



Developers Guide, API, Production and Deployment

NetBurner 3.5.6

NetBurner, Inc. 31-Oct-2025

1 NetBurner 3.5.6	1
1.1 Welcome to NetBurner Version 3!	1
1.2 Who Is This Guide For?	1
1.3 Recommended Starting Point for New Developers	2
1.4 Documentation Overview	2
1.4.1 The NetBurner Application Programming Interface (API)	2
1.4.2 Device Discovery and Configuration	3
1.4.3 The NBEclipse Getting Started Guide	3
1.4.4 Migration Guides	3
1.4.5 Example Programs	3
1.4.6 Programmers Guide	4
1.4.7 Platform References	4
1.4.8 System Diagnostics	4
1.4.9 Utilities	4
1.4.10 Production and Deployment	4
1.4.11 Location of Original Factory Program	5
1.4.12 Experienced NetBurner Customers	5
2 Build System	7
2.1 Introduction	7
2.2 GCC Compiler Flags	7
2.2.1 Generic	7
2.2.2 Architecture-Specific Compiler Flags	8
2.2.2.1 ARM Cortex-M7	8
2.2.2.2 ARM Cortex-M0+	8
2.2.2.3 ColdFire	8
2.2.3 CPU-Specific Compiler Flags	9
2.2.3.1 SAME70	9
2.2.3.2 MIMXRT11xx, MIMXRT10xx	9
2.2.3.3 SAMD20	9
2.2.3.4 MCF5441X	9
2.2.4 Platform-Specific Compiler Flags	9
2.2.4.1 MODM7AE70	9
2.2.4.2 SBE70LC	10
2.2.4.3 SOMRT1061	10
2.2.4.4 MOD5441X	10
2.2.4.5 NANO54415	10
2.2.4.6 SB800EX	10
2.3 Linker Flags	10
2.3.1 Generic	10
2.3.2 Architecture-Specific Linker Flags	11
2.3.3 CPU-Specific Linker Flags	11

2.3.3.1 SAME70	11
2.3.3.2 MIMXRT11xx, MIMXRT10xx	11
2.3.3.3 SAMD20	11
2.3.3.4 MCF5441X	12
2.3.4 Platform-Specific Linker Flags	12
2.4 Pack Binary	12
2.4.1 Generic	12
2.4.2 Architecture-Specific	12
2.4.3 CPU-Specific	13
2.4.3.1 MCF5441X	13
2.4.3.2 SAMD20	13
2.4.3.3 SAME70	13
2.4.3.4 MIMXRT11xx, MIMXRT10xx	13
2.4.4 Platform-Specific	13
2.4.4.1 MODM7AE70	13
2.4.4.2 SBE70LC	14
2.4.4.3 SOMRT1061	14
2.4.4.4 MOD5441X	14
2.4.4.5 NANO54415	14
2.4.4.6 SB800EX	14
3 Command Line Tools	15
3.1 Introduction	15
3.2 Command Line Summary	15
3.3 Command Line Environment Variables	15
3.4 Create a New Project From an Example	16
3.4.1 Edit the makefile	17
3.4.2 Edit main.cpp	17
3.5 Modifying System Files with Overload	18
3.5.1 Important Overload Rules	18
3.6 GNU Debugger (GDB)	19
3.6.1 Example GDB Session	19
3.7 Addr2line	21
3.7.1 ColdFire platforms	22
3.7.2 ARM platforms	22
4 Device Discovery and Configuration	23
4.1 Introduction	23
4.1.1 Device Discovery Methods:	23
4.1.2 Device Configuration methods:	23
4.2 Configuration Procedure	24
4.2.1 Configuration Settings	26
4.2.1.1 Config	26

4.2.1.2 Update Application	26
4.2.1.3 Config > Sys	26
4.2.1.4 Config > Sys > BOOT	26
4.2.1.5 Config > Sys > NetIf	27
4.2.1.6 Config > Sys > NetIf > Ethernet0	27
4.2.1.7 Config > Sys > NetIf > Ethernet0 > IPv4	27
4.2.1.8 Config > Sys > NetIf > Ethernet0 > IPv6	28
4.3 The JSON Object	29
4.3.1 Developer Notes	30
4.4 Configuration Security	31
4.5 Serial Configuration Server	31
4.5.1 Serial Configuration	32
4.5.2 Serial App Downloads	37
4.5.2.1 Start of Serial Configuration Server.	38
4.5.2.2 Optional, type "help" to see a list of commands.	38
4.5.2.3 Type "flash" command	38
4.5.2.4 Send Application File	39
4.5.3 Serial Config Settings	39
5 Migration Guides	41
5.1 Release 2.x to 3.x Overview	41
5.1.1 Hardware	42
5.1.2 Configuration	42
5.1.3 RTOS	42
5.1.4 TCP	43
5.1.5 Network Interfaces	43
5.1.6 SSL/TLS	43
5.1.7 System Initialization	43
5.1.8 NBEclipse	43
5.1.9 GCC	44
5.2 Release 2.x to 3.x Porting Guide	44
5.2.1 Device Configuration	44
5.2.2 Configuration Procedure	44
5.2.3 Recommended Examples	45
5.2.4 Configuration Security	45
5.2.4.1 Cautions When Using the Jumper Recover Procedure on a Device	46
5.2.5 Device Application Updates	46
5.2.6 Eclipse and Development Tools Updates	46
5.2.7 SSL/TLS	47
5.2.8 Notes on Porting From Previous Revisions	47
5.2.9 Porting I2C From MCF52xx Based Products to MODM7AE70	48
5.3 Upgrade 5441x Platforms to 3.x	50

5.3.1 Introduction	50
5.3.2 5441x Product Models Numbers	50
5.3.3 3.x Update Procedures	50
5.3.3.1 Procedure For a New 5441x Device Running The NetBurner Factory App	51
5.3.3.2 Procedure Using the 2.x to 3.x Conversion Application: xxxx-3p0-Update_APP.s19	51
5.3.3.3 Procedure Using Alternate Boot Monitor	51
5.3.4 Procedure to Revert Back to 2.x	52
5.3.4.1 Serial Port Procedure	52
5.3.4.2 Hardware Reset Jumper Procedure	52
5.3.5 Note on IPSetup utility and 3.x Devices	53
6 NBEclipse	55
6.1 Getting Started Guide	55
6.1.1 Introduction	55
6.1.1.1 How to Use This Guide	55
6.1.1.2 Source Code for Example Programs	56
6.1.1.3 Hardware Setup	56
6.1.1.4 Software Installation	56
6.1.1.5 Upgrading From a Previous Installation	56
6.1.1.6 Java Installation	56
6.1.1.7 Debug Port	57
6.1.2 Platform Overview	57
6.1.2.1 NetBurner Development Platform Choices	57
6.1.3 Project Management	57
6.1.3.1 NBEclipse Key Points	58
6.1.4 Creating Projects and Importing Files	58
6.1.4.1 Create a New Project	59
6.1.4.2 Select the NetBurner Project Type	59
6.1.4.3 Specify Project Name and Executable Option	60
6.1.4.4 Project Configurations	61
6.1.4.5 Select Your NetBurner Target Device	62
6.1.4.6 Application Wizard	64
6.1.4.7 New Project Created	65
6.1.4.8 Import Example Source Files to the Project SRC Folder	66
6.1.4.9 Open your main.cpp source file	70
6.1.4.10 Project Import Complete	71
6.1.4.11 Download the Application to Your Device	73
6.1.4.12 View the Device Web Page	76
6.1.4.13 View Serial Port Status Messages with MTTTY	77
6.1.5 Debugger	78
6.1.5.1 Starting a Debug Session	78
6.1.6 Rebuilding Projects & Libraries	81

6.1.6.1 Rebuild Project Source Files	81
6.1.6.2 Rebuild Library Files	82
6.1.7 Overload Directory	83
6.2 Add a Library to a Project	90
6.3 Change The IP Targeted By A Project	91
6.4 Change CompCode Settings for EDFS	91
7 PC Tools and Utilities	95
7.1 Multi-threaded TTY Serial Terminal (MTTTY)	95
7.1.1 MTTY FAQ	96
7.2 Wget	96
7.3 NBUdate	97
7.4 Smart Traps	97
7.4.1 Trap Information	98
7.4.2 Register Information	99
7.4.3 RTOS Information	99
7.4.4 Task Information	99
7.5 Task Scan	99
7.5.1 TaskScan FAQ	100
7.6 WinAddr2Line	101
7.6.1 Using WinAddr2Line	102
7.7 IPSetup	102
7.8 AutoUpdate	102
7.9 Virtual Comm Port	103
7.10 CompHtml	103
7.11 CompCode	103
7.12 flashpack	103
8 Platform References	105
8.1 MODM7AE70 Platform Reference	105
8.1.1 Introduction	105
8.1.2 Processor Information	105
8.1.2.1 Definitions	105
8.1.3 Datasheet	105
8.1.4 Development Board Schematic	106
8.1.5 Memory Operation	106
8.1.6 Memory Map	106
8.1.7 External Bus Interface	106
8.2 MOD54415, MOD54417 Platform Reference	107
8.2.1 Introduction	107
8.2.2 Processor Information	107
8.2.2.1 Definitions	107
8.2.3 Datasheet	107

8.2.4 Development Board Schematic	107
8.2.5 Memory Operation	107
8.2.6 Memory Map	108
8.3 NANO54415 Platform Reference	108
8.3.1 Introduction	108
8.3.2 Processor Information	108
8.3.2.1 Definitions	109
8.3.3 Datasheet	109
8.3.4 Development Board Schematic	109
8.3.5 Memory Operation	109
8.3.6 Processor Runtime Memory Map	109
8.3.7 Application SPI Flash Memory	110
8.4 SB800EX Platform Reference	110
8.4.1 Introduction	110
8.4.2 Processor Information	110
8.4.2.1 Definitions	110
8.4.3 Datasheet	110
8.4.4 Memory Operation	110
8.4.5 Processor Runtime Memory Map	111
8.4.6 Application SPI Flash Memory	111
8.5 SBE70LC Platform Reference	112
8.5.1 Introduction	112
8.5.2 Processor Information	112
8.5.2.1 Definitions	112
8.5.3 Datasheet	112
8.5.4 Development Board Schematic	112
8.5.5 Memory Operation	112
8.5.6 Memory Map	112
8.6 SOMRT1061 Platform Reference	113
8.6.1 Introduction	113
8.6.2 Processor Information	113
8.6.2.1 Definitions	113
8.6.3 Datasheet and Pins	113
8.6.4 Development Board Schematic	113
8.6.5 SOMRT1061 Platform General Information	114
8.6.5.1 The 8MB of Flash uses a Flash File System and is used for all storage purposes:	114
8.6.5.2 External Memory Bus	114
8.6.6 Boot Sequence	114
8.6.7 UART Signals and Port Number Mapping	115
8.7 Recovery: MODM7AE70, SBE70LC	115
8.7.1 Introduction	115
8.7.2 Repeating Traps or Resets	115

8.7.2.1 Step 1: Abort to the Configuration Server	115
8.7.2.2 Step 2: Download a New Application	115
8.7.3 Unresponsive Device or No Serial Port Access	116
8.7.3.1 Recovery Jumper Locations	116
8.8 Recovery: MOD5441x, NANO54415, SB800EX	118
8.8.1 Step 1: Abort to the Alternate Boot Monitor	118
8.8.2 Recovery Jumper Locations	119
8.8.2.1 NANO54415	119
8.8.2.2 MOD5441x	119
8.8.2.3 SB800EX	120
8.8.3 Step 2: Download a New Application	121
8.8.3.1 Network	121
8.8.3.2 Serial Port	121
8.9 Recovery: SOMRT1061	122
8.9.1 Introduction	122
8.9.2 Boot Components and Features Overview	122
8.9.3 Boot Sequence Overview	122
8.9.3.1 Normal Boot	122
8.9.3.2 Incorrect Network Settings	122
8.9.3.3 Application Starts, But Traps After Application Initialization	123
8.9.3.4 Application Starts, But Traps Before Application Initialization	123
8.9.4 Procedure to Abort to the Configuration Server	123
8.9.5 Uploading a New Application	123
8.9.5.1 Web Page Upload	123
8.9.5.2 NBUpdate	123
8.9.5.3 Serial Console Interface	123
8.9.6 Procedures to Abort to the Alternate Boot Monitor	124
8.9.6.1 No Valid Application	124
8.9.6.2 Configuration Server Access Operational	124
8.9.6.3 Application Traps Before UserMain() Starts	124
8.9.7 Hardware Jumper Recovery	124
8.9.7.1 Recovery Jumper Operation	124
8.9.7.2 Procedure	125
8.9.7.3 Recovery Jumper Location	125
8.10 ARM Cache Management	126
8.10.1 Details	126
8.10.2 Instructions	126
8.10.2.1 Turning cache on/off	126
8.10.2.2 Declaring static objects non cachable	126
8.10.2.3 Cleaning Cache	126
8.10.2.4 Invalidating Cache	126

9 Production & Deployment	127
9.1 Production Releases and Customer Deployment	127
9.2 Finding a Device on a Network	127
9.3 Application Programming	127
9.3.1 Configuration Web Page	128
9.3.2 nbupdate Utility	128
9.3.3 wget Utility	128
9.3.3.1 Program an Application	128
9.3.3.2 Read The Configuration	129
9.3.3.3 Program New Configuration Settings	129
9.4 NetBurner SSL Certificate Upload Commands	129
9.4.1 Overview	129
9.4.2 Requirements	129
9.4.3 Linux/Unix Commands	129
9.4.3.1 Using curl (Recommended)	129
9.4.3.2 Using wget (Simple approach)	129
9.4.4 Windows PowerShell Commands	130
9.4.4.1 Method 1: Using Invoke-RestMethod (Recommended)	130
9.4.4.2 Method 2: Using Invoke-WebRequest	130
9.4.4.3 Method 3: Manual multipart construction	130
9.4.5 Windows Command Prompt	130
9.4.5.1 Method 1: Using curl (Windows 10/11)	130
9.4.5.2 Method 2: PowerShell one-liner from Command Prompt	131
9.4.5.3 Method 3: Batch file approach	131
9.4.6 Usage Notes	131
9.4.7 Recommended Approaches	131
9.4.8 Troubleshooting	131
9.4.9 Security Considerations	132
9.5 Customer Configuration	132
10 Programmers Guide	133
10.1 Overview	133
10.2 Table of Contents	133
10.3 About This Guide	133
10.3.1 Guide Navigation	133
10.3.1.1 Quick Start Path	133
10.3.1.2 By Use Case	134
10.3.2 Document Descriptions	135
10.3.2.1 Introduction to NetBurner Programming	135
10.3.2.2 Configuration Server	135
10.3.2.3 Custom Branding	135
10.3.2.4 IPv4-IPv6 Dual Stack	136

10.3.2.5	EFFS File System	136
10.3.2.6	File Descriptors	137
10.3.2.7	HTML Processing	137
10.3.2.8	Interrupt Handling	137
10.3.2.9	JSON Lexer	138
10.3.2.10	NetBurner RTOS	138
10.3.2.11	Network Protocols	139
10.3.2.12	SSL-TLS Security	139
10.3.2.13	SSH Protocol	139
10.3.2.14	Web Server	140
10.3.3	Development Workflow	140
10.3.3.1	Typical Application Development Process	140
10.3.4	Common Implementation Patterns	141
10.3.4.1	Pattern 1: Web-Based Device Control	141
10.3.4.2	Pattern 2: Network Data Logger	141
10.3.4.3	Pattern 3: Secure Remote Monitoring	142
10.3.5	Cross-Reference Guide	142
10.3.5.1	By Task	142
10.3.6	Quick Start Checklist	143
10.3.6.1	Essential Reading for New Developers	143
10.3.6.2	Advanced Topics Preparation	143
10.3.7	Document Conventions	144
10.3.7.1	Code Examples	144
10.3.7.2	Platform References	144
10.3.8	Getting Help	144
10.3.8.1	Documentation Resources	144
10.3.9	Version Information	144
10.3.10	Next Steps	145
10.4	Introduction	145
10.4.1	Intro to NetBurner Programming	145
10.4.2	Overview	145
10.4.3	Prerequisites	145
10.4.4	Learning Approach	145
10.4.5	Source Code Location	145
10.4.6	Creating Your First Application	146
10.4.6.1	Using the NBEclipse Application Wizard	146
10.4.6.2	Template Application Structure	146
10.4.6.3	Application Execution Flow	146
10.4.7	Key System Functions	147
10.4.7.1	init()	147
10.4.7.2	WaitForActiveNetwork()	147
10.4.7.3	StartHttp()	147

10.4.8 Output Functions	147
10.4.8.1 printf() vs printf()	147
10.4.9 Application Design Principles	148
10.4.9.1 The Main Loop	148
10.4.9.2 Timing Considerations	148
10.4.9.3 Resource Management	148
10.4.10 Next Steps	148
10.4.11 Quick Reference	148
10.4.11.1 Critical Constants	148
10.4.11.2 Core Functions	148
10.4.12 Architecture Overview	149
10.5 Config Server Programming Guide	149
10.5.1 Configuration Server Programming Guide	149
10.5.2 Introduction	149
10.5.2.1 Configuration System Overview	149
10.5.2.2 Access Methods	149
10.5.2.3 Key Features	149
10.5.3 Configuration Structure	150
10.5.3.1 Configuration Tree Hierarchy	150
10.5.3.2 Root Level: Config	150
10.5.4 Config.AppData	150
10.5.4.1 Purpose and Usage	150
10.5.4.2 Available Configuration Objects	151
10.5.4.3 Example: Application Settings	151
10.5.4.4 Configuration Object Hierarchy	151
10.5.5 Config.Sys - System Settings	151
10.5.5.1 System Configuration Tree	151
10.5.6 Config.Sys.Boot - Boot Settings	151
10.5.6.1 Boot Configuration Parameters	151
10.5.6.2 Boot Process Flow	152
10.5.6.3 Serial Config Server Modes	152
10.5.6.4 Security Configuration	153
10.5.7 Config.Sys.NetIf - Network Interfaces	153
10.5.7.1 Network Interface Tree	153
10.5.7.2 Common Interface Parameters	153
10.5.7.3 Discovery Configuration	153
10.5.8 Config.Sys.NetIf.Ethernet0.IPv4	154
10.5.8.1 IPv4 Settings Overview	154
10.5.8.2 IPv4 Parameters	154
10.5.8.3 IPv4 Mode Selection	155
10.5.9 Config.Sys.NetIf.Ethernet0.IPv6	155
10.5.9.1 IPv6 Settings Overview	155

10.5.9.2 IPv6 Parameters	156
10.5.9.3 IPv6 Mode Selection	156
10.5.10 Configuration Web Interface	156
10.5.10.1 Accessing the Web Interface	156
10.5.10.2 Web Interface Features	157
10.5.10.3 Application Update Feature	157
10.5.11 Serial Configuration Interface	157
10.5.11.1 Accessing Serial Config	158
10.5.11.2 Serial Config Prompt	158
10.5.11.3 Navigation Model	158
10.5.11.4 Serial Config Commands	158
10.5.11.5 Command Examples	158
10.5.11.6 Uploading Application via Serial	159
10.5.12 Creating a Custom Web Interface	160
10.5.12.1 Customization Workflow	160
10.5.12.2 Steps to Customize	160
10.5.12.3 Custom Interface Example	160
10.5.13 Configuration API Examples	161
10.5.13.1 Example Projects Overview	161
10.5.13.2 Available Examples	161
10.5.13.3 Example Selection Guide	162
10.5.14 Best Practices	163
10.5.14.1 Configuration Design Principles	163
10.5.15 Troubleshooting	164
10.5.15.1 Common Issues and Solutions	164
10.5.15.2 Debug Checklist	165
10.5.16 Advanced Topics	166
10.5.16.1 Custom Configuration Objects	166
10.5.16.2 Configuration Callbacks	166
10.5.16.3 Configuration Persistence	167
10.5.17 API Quick Reference	167
10.5.17.1 Core Configuration Functions	167
10.5.17.2 Configuration Object Types	167
10.5.17.3 Accessing Values	167
10.5.18 Summary	168
10.5.18.1 Key Features Recap	168
10.5.18.2 Access Methods Summary	168
10.5.18.3 Configuration Hierarchy	168
10.5.18.4 Best Practices Checklist	168
10.5.18.5 Getting Started	169
10.6 Custom Branding	169
10.6.1 Custom Branding	169

10.6.2 Overview	169
10.6.3 Branding Capabilities Options	169
10.6.4 Customization Options	170
10.6.4.1 1. Configuration Server Logo Replacement	170
10.6.4.2 2. Custom Configuration Interface	170
10.6.4.3 3. Physical Device Labeling	171
10.6.4.4 4. Discovery Utility Customization	171
10.6.4.5 5. Custom Cloud Server Infrastructure	172
10.6.5 Implementation Strategy	173
10.6.5.1 Planning Your Customization	173
10.6.5.2 Recommended Path	174
10.6.6 Best Practices	174
10.6.6.1 Visual Consistency	174
10.6.6.2 Configuration Access Patterns	174
10.6.6.3 Testing Checklist	175
10.6.7 Resources and Support	175
10.6.7.1 Example Code Locations	175
10.6.7.2 Getting Started	175
10.6.7.3 Support Channels	175
10.6.8 Security Considerations	176
10.7 IPv4/IPv6 Dual Stack Guide	176
10.7.1 Overview	176
10.7.1.1 Key Changes in 2.8.0+	176
10.7.2 IPv6 Address Assignment	176
10.7.2.1 Address Assignment Flow	176
10.7.2.2 Link-Local Addresses	177
10.7.2.3 Router Auto-Configuration (SLAAC)	177
10.7.2.4 DHCPv6 Configuration	177
10.7.2.5 Static IP Addresses	178
10.7.2.6 Complete Address Assignment Hierarchy	178
10.7.3 IPADDR Class Migration	178
10.7.3.1 Storage Considerations	178
10.7.3.2 Null Value Checking	179
10.7.3.3 Null IP in Function Parameters	179
10.7.4 IPADDR Member Functions	179
10.7.4.1 Function Types	179
10.7.4.2 Setting IP Address Values	179
10.7.4.3 Reading and Checking IP Address Values	179
10.7.4.4 Output Functions	180
10.7.5 Domain Name Service (DNS)	180
10.7.5.1 DNS Function Variants	180
10.7.5.2 Universal DNS Code Pattern	181

10.7.5.3 DNS Record Types	181
10.7.5.4 IPv6 DNS Priority Control	181
10.7.5.5 DNS Resolution Strategy Matrix	182
10.7.6 Platform-Specific Considerations	182
10.7.6.1 macOS and BSD Systems	182
10.7.7 Migration Checklist	182
10.7.7.1 Code Audit	182
10.7.7.2 Storage Migration	183
10.7.7.3 Testing Strategy	183
10.7.8 Best Practices	183
10.7.8.1 Portable Code Patterns	183
10.7.8.2 Storage Design	183
10.7.8.3 DNS Resolution	184
10.7.9 Summary	184
10.7.9.1 Key Takeaways	184
10.7.9.2 Migration Path	184
10.8 EDFS Programming Guide	184
10.8.1 NetBurner EDFS Programming Guide	184
10.8.2 Overview	184
10.8.2.1 Topics Covered	185
10.8.3 System Architecture	185
10.8.4 Supported Platforms	185
10.8.4.1 Platform Capabilities Matrix	185
10.8.4.2 Interface Types	185
10.8.5 Example Programs	186
10.8.5.1 EDFS-FAT Examples	186
10.8.5.2 EDFS-FAT Security Examples	186
10.8.5.3 EDFS-STD Examples	186
10.8.6 Flash Card Hardware Interface	187
10.8.6.1 SD/MMC Hardware Connection Diagram	187
10.8.6.2 Signal Connection Table	187
10.8.6.3 Hardware Specifications	187
10.8.6.4 GPIO Pin Configuration	188
10.8.6.5 Interrupt-Driven SPI (ColdFire Platforms Only)	188
10.8.7 EDFS-FAT File System Operation	188
10.8.7.1 Typical Operation Flow	188
10.8.7.2 Common EDFS-FAT Function Calls	189
10.8.7.3 File Time and Date Stamps	189
10.8.7.4 File System Utilities	189
10.8.8 EDFS-FAT Project Configuration	190
10.8.8.1 NBEclipse Project Setup	190
10.8.8.2 Makefile Configuration	190

10.8.9	FFFS-STD File System (On-Chip Flash)	190
10.8.9.1	FFFS-FAT vs EDFS-STD Comparison	190
10.8.9.2	Flash Memory Layout	190
10.8.9.3	Memory Allocation Considerations	190
10.8.10	FFFS-STD Project Configuration	191
10.8.10.1	NBEclipse Configuration	191
10.8.11	FFFS-STD Makefile Configuration	192
10.8.11.1	Step 1: Add EDFS-STD Library	192
10.8.11.2	Step 2: Platform-Specific Configuration	192
10.8.12	Flash Memory Address Configuration	192
10.8.12.1	Platform Memory Map	192
10.8.12.2	Memory Region Diagram	192
10.8.12.3	Configuration File Reference	192
10.8.13	Debug Port Configuration	193
10.8.13.1	Overview	193
10.8.13.2	Default Configuration	193
10.8.13.3	Supported Functions	193
10.8.13.4	Custom Configuration	193
10.8.14	API Reference Documentation	193
10.8.14.1	FFFS-FAT API Reference	193
10.8.14.2	FFFS-STD API Reference	193
10.8.14.3	Key Function Mapping	194
10.8.15	Best Practices	194
10.8.15.1	General Guidelines	194
10.8.15.2	FFFS-FAT Specific	194
10.8.15.3	FFFS-STD Specific	194
10.8.15.4	Performance Optimization	194
10.8.16	Troubleshooting	195
10.8.16.1	Common EDFS-FAT Issues	195
10.8.16.2	Common EDFS-STD Issues	195
10.8.17	Additional Resources	196
10.8.17.1	Documentation References	196
10.8.17.2	Support Files	196
10.8.17.3	Example Locations	196
10.8.18	Summary	196
10.9	File Descriptors	196
10.9.1	File Descriptors	196
10.9.2	Overview	196
10.9.3	File Descriptor Architecture	197
10.9.4	File Descriptor Allocation	197
10.9.4.1	Allocation Map	197
10.9.4.2	Detailed Breakdown	197

10.9.4.3 Expansion Pool Usage	197
10.9.5 Creating Custom I/O Drivers	198
10.9.5.1 Driver Architecture	198
10.9.5.2 Core Header File	198
10.9.5.3 State Management Functions	198
10.9.6 IoExpandStruct Definition	198
10.9.6.1 Structure Members	198
10.9.6.2 Data Flow	199
10.9.7 File Descriptor Management API	199
10.9.7.1 Core Functions	199
10.9.7.2 Function Details	199
10.9.8 TCP Socket Notifications	200
10.9.8.1 Callback System	200
10.9.8.2 Callback Definition	200
10.9.8.3 Callback Parameters	200
10.9.8.4 Implementation Example	201
10.9.9 Using select() with File Descriptors	201
10.9.9.1 Multiplexing Multiple I/O Sources	201
10.9.9.2 select() Usage Pattern	201
10.9.9.3 Mixing Device Types	202
10.9.10 Complete Implementation Example	202
10.9.10.1 Custom Ring Buffer Device	202
10.9.10.2 Usage of Custom Device	203
10.9.11 Best Practices	204
10.9.11.1 Resource Management	204
10.9.11.2 Guidelines	204
10.9.12 Troubleshooting	205
10.9.12.1 Common Issues	205
10.9.12.2 Debugging Tips	205
10.9.13 Summary	205
10.9.13.1 Key Concepts	205
10.9.13.2 Quick Reference	206
10.10 HTML Processing	206
10.10.1 HTML Processing	206
10.10.2 Introduction	206
10.10.2.1 Quick Start	206
10.10.2.2 Web Server Initialization	206
10.10.2.3 Content Sources	206
10.10.3 Dynamic Content Overview	206
10.10.3.1 Dynamic Content Methods	207
10.10.4 HTML Tags	207
10.10.4.1 Tag Overview	207

10.10.4.2 Tag Syntax	207
10.10.5 The CPPCALL Tag	208
10.10.5.1 Function Flow	208
10.10.5.2 Function Signature	208
10.10.5.3 Example: Display Client Information	208
10.10.6 The VARIABLE Tag	208
10.10.6.1 Use Case 1: Display Variables	208
10.10.6.2 Supported Variable Types	209
10.10.6.3 Use Case 2: Parameterized Function Calls	209
10.10.7 The INCLUDE Tag and htmlvar.h	209
10.10.7.1 Two Methods for Include Resolution	209
10.10.7.2 Method 1: INCLUDE Tag in HTML	209
10.10.7.3 Method 2: htmlvar.h Header File	210
10.10.7.4 Include Resolution Flow	210
10.10.8 Extending VARIABLE Tag Support	210
10.10.8.1 Custom Type Example	210
10.10.8.2 System Example: MAC Address Display	210
10.10.9 Configuration System Tags	211
10.10.9.1 Configuration Tag Types	211
10.10.9.2 Configuration Tree Structure	211
10.10.9.3 Example: Display Configuration Values	211
10.10.9.4 Configuration Access Flow	211
10.10.10 HTTP GET Request Handler	211
10.10.10.1 When to Use GET Handlers	211
10.10.10.2 GET Handler Flow	212
10.10.10.3 Creating a GET Handler	212
10.10.10.4 CallbackFunctionPageHandler Parameters	212
10.10.10.5 URL Encoding Example	213
10.10.11 HTTP Form POST Handler	213
10.10.11.1 Creating a POST Handler	213
10.10.11.2 Step 1: Create HTML Form	213
10.10.11.3 Step 2: Create Callback Function	213
10.10.11.4 Step 3: Register Handler	214
10.10.11.5 POST Event Flow	214
10.10.11.6 POST Events	215
10.10.11.7 Multiple Form Handling	215
10.10.12 Complete Examples	215
10.10.12.1 Example 1: System Status Page with Mixed Tags	215
10.10.12.2 Example 2: Interactive Control Panel	215
10.10.13 Best Practices	216
10.10.13.1 Tag Usage Guidelines	216
10.10.13.2 Performance Considerations	217

10.10.13.3 Security Guidelines	217
10.10.13.4 Code Organization	217
10.10.14 Troubleshooting	218
10.10.14.1 Common Issues and Solutions	218
10.10.14.2 Debug Checklist	218
10.10.15 Additional Resources	218
10.10.15.1 Related Documentation	218
10.10.15.2 Key Header Files	219
10.10.15.3 Quick Reference	219
10.10.16 Summary	219
10.11 Interrupt Handling	220
10.11.1 interrupt_handling Interrupt Handling in NetBurner Applications	220
10.11.2 interrupt_overview Overview	220
10.11.2.1 interrupt_sources Common Interrupt Sources	220
10.11.2.2 interrupt_platform_variations Platform Variations	220
10.11.3 core_interrupt_functions Core Interrupt Functions	221
10.11.3.1 interrupt_header_requirements Header Requirements	221
10.11.3.2 interrupt_function_overview Function Overview	221
10.11.3.3 interrupt_setpinirq SetPinIrq() - Configure Interrupt Pin	221
10.11.3.4 interrupt_enableirq EnableIrq() - Activate Interrupt	222
10.11.3.5 interrupt_disableirq DisableIrq() - Deactivate Interrupt	222
10.11.4 isr_restrictions ISR Restrictions	222
10.11.4.1 isr_critical_constraints Critical Constraints	222
10.11.4.2 isr_prohibited_rtos Prohibited RTOS Functions	222
10.11.4.3 isr_prohibited_io Prohibited I/O Functions	223
10.11.4.4 isr_prohibited_memory Prohibited Memory Functions	223
10.11.4.5 isr_recommended_pattern Recommended ISR Pattern	223
10.11.5 platform_specific_implementations Platform-Specific Implementations	223
10.11.6 same70_platforms SAME70 Platforms (MODM7AE70 and SBE70LC)	224
10.11.6.1 same70_architecture Architecture Overview	224
10.11.6.2 same70_port_priority Port Priority Configuration	224
10.11.6.3 same70_port_mapping Port Priority Mapping	225
10.11.6.4 same70_workflow Configuration Workflow	225
10.11.6.5 same70_example Complete SAME70 Example	225
10.11.7 mcf5441x_platforms MCF5441x Platforms (MOD54415, MOD54417, NANO54415 and SB800EX)	225
10.11.7.1 mcf5441x_architecture Architecture Overview	225
10.11.7.2 mcf5441x_interrupt_macro INTERRUPT() Macro	226
10.11.7.3 mcf5441x_setintc SetIntc() Function	227
10.11.7.4 mcf5441x_example Complete MCF5441x Example: 1-Wire Interface	228
10.11.7.5 mcf5441x_workflow MCF5441x Configuration Workflow	229
10.11.7.6 mcf5441x_best_practices Best Practices for MCF5441x	229

10.11.8 somrt1061_platform SOMRT1061 Platform	230
10.11.8.1 somrt1061_overview Overview	230
10.11.8.2 somrt1061_configuration Configuration Approach	230
10.11.8.3 somrt1061_notes Platform-Specific Notes	230
10.11.8.4 somrt1061_example SOMRT1061 Example	230
10.11.9 interrupt_summary Summary and Best Practices	231
10.11.9.1 interrupt_cross_platform Cross-Platform Compatibility	231
10.11.9.2 interrupt_platform_selection Platform Selection Guide	231
10.11.9.3 interrupt_general_best_practices General Best Practices	231
10.11.9.4 interrupt_debugging Debugging Interrupts	232
10.11.9.5 interrupt_error_prevention Error Prevention	232
10.11.10 interrupt_resources Additional Resources	232
10.12 JSON Lexer	232
10.12.1 JSON Lexer Programming Guide	232
10.12.2 Overview	232
10.12.2.1 Core Components	232
10.12.2.2 Version Notes	233
10.12.3 Loading JSON Data	233
10.12.3.1 Method 1: From Text Blob	233
10.12.3.2 Method 2: From Web Server	233
10.12.3.3 Method 3: From File Descriptor	233
10.12.4 Data Flow Diagram	233
10.12.5 Accessing JSON Data	234
10.12.5.1 Example JSON Structure	234
10.12.5.2 Basic Access Patterns	234
10.12.6 Access Operator Summary	234
10.12.6.1 Navigation Example	234
10.12.7 Type Conversion Operators	235
10.12.8 Validation Functions	235
10.12.8.1 IsValid() Method	235
10.12.8.2 Type Checking Functions	235
10.12.8.3 Validation Flow	235
10.12.9 Example References	235
10.12.9.1 Earthquake Example Application	235
10.12.9.2 Source Code	236
10.12.9.3 JSON Structure Navigation	236
10.12.9.4 Data Extraction Flow	237
10.12.9.5 Example Output	237
10.12.10 USGS JSON Response Structure	237
10.12.10.1 Response Schema	237
10.12.10.2 Metadata Section	237
10.12.10.3 Features Array	237

10.12.10.4 Parsing Strategy	238
10.12.11 Complete JSON Sample	238
10.12.11.1 Sample Structure	238
10.12.12 Best Practices	239
10.12.12.1 Memory Management	239
10.12.12.2 Error Handling	239
10.12.12.3 Performance Considerations	239
10.12.12.4 Type Safety	239
10.12.13 Advanced Usage	239
10.12.13.1 Nested Array Navigation	239
10.12.13.2 Dynamic Key Access	239
10.12.13.3 Array Iteration Patterns	239
10.12.14 Summary	240
10.13 NetBurner RTOS	240
10.13.1 Introduction	240
10.13.2 Priority Levels	240
10.13.3 What is a Preemptive RTOS?	241
10.13.4 Core Principle	241
10.13.5 Task Execution Flow	241
10.13.6 Blocking Scenarios	241
10.13.7 Function Categories	242
10.13.7.1 RTOS Blocking Functions	242
10.13.7.2 I/O Functions That Block	242
10.13.7.3 Network Functions That Block	242
10.13.7.4 Functions That Unblock Tasks	242
10.13.8 System Task Priorities	242
10.13.9 Task Creation	243
10.13.10 Entry Point: UserMain()	243
10.13.11 Task Creation Methods	243
10.13.11.1 Method 1: OSTaskCreatewName() - Full Control	243
10.13.11.2 Method 2: OSSimpleTaskCreatewName() - Simplified	243
10.13.12 Priority Selection Helper	243
10.13.13 Complete Example	244
10.13.14 Protecting Shared Data	245
10.13.15 Mechanism Selection Guide	245
10.13.16 Protection Mechanisms (Ordered by System Impact)	245
10.13.17 Detailed Mechanism Comparison	245
10.13.18 Important Considerations	246
10.13.18.1 Queue vs. FIFO	246
10.13.19 Semaphores	246
10.13.20 Concept	246
10.13.21 Operation Flow	246

10.13.22 Usage Pattern	246
10.13.23 Timeout Specification	247
10.13.24 Resource Protection Example	247
10.13.25 Message Queues	247
10.13.26 Concept	247
10.13.27 Queue Structure	247
10.13.28 Message Flow	247
10.13.29 Usage Example	248
10.13.30 FIFOs	248
10.13.31 Concept	248
10.13.32 Structure Requirements	248
10.13.33 FIFO Architecture	249
10.13.34 Memory Management Strategies	249
10.13.34.1 Strategy 1: Static Pool (Recommended)	249
10.13.34.2 Strategy 2: Dynamic Allocation (Not Recommended)	249
10.13.35 Queue vs. FIFO Decision	249
10.13.36 FIFO Functions	249
10.13.37 Critical Sections	250
10.13.38 OSCrit - Counted Critical Section	250
10.13.39 Operation Flow	250
10.13.40 Counting Behavior	250
10.13.41 Usage Examples	250
10.13.41.1 Manual Enter/Exit	250
10.13.41.2 C++ Scoped Object (Recommended)	250
10.13.42 OSCrit vs. OSLock	251
10.13.43 OSLock Usage Pattern	251
10.13.44 OS Flags	251
10.13.45 Concept	251
10.13.46 Flag Bitmap Structure	251
10.13.47 Flag Operations	251
10.13.48 Pending Modes	252
10.13.49 Flag Functions Summary	252
10.13.50 Usage Example	252
10.13.51 System Configuration	253
10.13.52 Predef.h and Predef-overload.h	253
10.13.53 Configuration Architecture	253
10.13.54 Why Use Predef-overload.h?	253
10.13.55 Setup Methods	254
10.13.55.1 NBEclipse Setup	254
10.13.55.2 Command Line Tools Setup	254
10.13.56 Predef-overload.h Best Practices	254
10.13.57 Configuration Workflow	254

10.13.58 Common Configuration Overrides	254
10.13.59 Verification After Changes	255
10.13.60 Summary	255
10.13.61 Key Takeaways	255
10.13.62 Quick Reference	256
10.13.63 Further Reading	256
10.14 Network Protocols TCP and UDP	256
10.14.1 TCP vs UDP Comparison	256
10.14.1.1 Protocol Characteristics	256
10.14.1.2 Protocol Comparison Diagram	257
10.14.1.3 Real-World Analogies	257
10.14.1.4 Decision Factors	257
10.14.1.5 Common Protocol Usage	258
10.14.1.6 Performance Considerations	258
10.14.1.7 Key Takeaways	258
10.14.1.8 Additional Resources	259
10.14.1.9 Quick Reference	259
10.14.2 TCP Protocol	260
10.14.2.1 TCP Protocol	260
10.14.2.2 TCP Stream Behavior	260
10.14.2.3 TCP Connection Requirements	260
10.14.2.4 Well-Known Port Numbers	261
10.14.2.5 TCP Server Implementation	261
10.14.2.6 TCP Client Implementation	268
10.14.2.7 TCP Error Handling	271
10.14.3 UDP Protocol	277
10.14.3.1 UDP Protocol	277
10.14.3.2 UDP Implementation Comparison	277
10.14.3.3 UDP Class Interface	278
10.14.3.4 UDP Sockets Interface	280
10.15 SSL/TLS and Certificates	289
10.15.1 Introduction	289
10.15.2 Understanding SSL/TLS Fundamentals	289
10.15.3 The Security Problem	290
10.15.4 How SSL/TLS Solves These Problems	290
10.15.5 The Trust Chain: How Authentication Works	290
10.15.5.1 Real-World Analogy: Trusting a Doctor	290
10.15.5.2 SSL/TLS Certificate Chain	290
10.15.6 Public Key Cryptography Simplified	291
10.15.7 Certificate Types at a Glance	291
10.15.8 Quick Start Guide	291
10.15.9 For SSL Servers (Your NetBurner Device)	291

10.15.9.1 Option 1: Automatic (Recommended for Production)	291
10.15.10 For SSL Clients (Connecting to Servers)	292
10.15.11 Certificate Types and Use Cases	292
10.15.12 Decision Matrix	292
10.15.13 1. Self-Signed Certificates	292
10.15.14 2. Private Certificate Authority (CA)	293
10.15.15 3. Public CA Certificates	294
10.15.16 4. ACME (Automated Certificate Management)	295
10.15.17 Use Case Decision Table	296
10.15.18 Implementation Methods	296
10.15.19 Method 1: Automatic Onboard Certificate Generation	296
10.15.19.1 How It Works	297
10.15.19.2 Basic Implementation	297
10.15.19.3 Advanced: Auto-Regeneration	297
10.15.19.4 Customizing Certificate Generation	297
10.15.20 Method 2: ACME Automated Certificates	298
10.15.20.1 Complete ACME Example	298
10.15.20.2 ACME Troubleshooting Checklist	298
10.15.21 Method 3: Manual Certificate Installation	298
10.15.21.1 Step-by-Step Process	298
10.15.21.2 Installing Certificate Chain	299
10.15.22 Method 4: SSL Client Implementation	300
10.15.22.1 Basic SSL Client	300
10.15.22.2 Advanced: Certificate Verification	300
10.15.22.3 Loading CAs from Storage	301
10.15.23 Troubleshooting	301
10.15.24 Common Browser Warning Messages	301
10.15.24.1 1. "Your Connection is Not Private"	301
10.15.24.2 2. "Certificate Name Mismatch"	301
10.15.24.3 3. "Certificate Expired"	302
10.15.24.4 4. "Certificate Not Yet Valid"	302
10.15.25 SSL Handshake Failures	302
10.15.25.1 Debugging Connection Issues	302
10.15.25.2 Common Error Codes	303
10.15.26 Certificate Verification Issues	303
10.15.26.1 Verification Flowchart	303
10.15.26.2 Bypass Verification (Development Only!)	304
10.15.27 Memory and Resource Issues	304
10.15.27.1 Memory Requirements	304
10.15.27.2 Checking Memory	304
10.15.28 Performance Optimization	304
10.15.28.1 Cipher Suite Selection	304

10.15.28.2 Connection Reuse	305
10.15.29 Best Practices	305
10.15.30 Security Principles	305
10.15.30.1 1. Private Key Protection	305
10.15.30.2 2. Certificate Validation	306
10.15.30.3 3. Protocol and Cipher Configuration	306
10.15.31 Production Checklist	306
10.15.31.1 Pre-Deployment	306
10.15.31.2 Post-Deployment Monitoring	306
10.15.32 Development Workflow	307
10.15.32.1 Recommended Development Cycle	307
10.15.32.2 Configuration Management	307
10.15.33 Common Pitfalls to Avoid	308
10.15.33.1 1. Time Synchronization	308
10.15.33.2 2. Certificate Chain Issues	308
10.15.33.3 3. Resource Leaks	308
10.15.33.4 4. Hardcoded Credentials	308
10.15.34 Additional Resources	309
10.15.35 NetBurner Examples	309
10.15.36 Documentation and References	309
10.15.36.1 Official Documentation	309
10.15.36.2 Educational Resources	309
10.15.36.3 Wikipedia Articles	309
10.15.37 Tools and Utilities	310
10.15.37.1 Certificate Management	310
10.15.37.2 Testing and Debugging	310
10.15.37.3 Online Testing	310
10.15.38 Quick Reference	310
10.15.38.1 Essential SSL Functions	310
10.15.38.2 Configuration Constants	310
10.16 SSH Programming Guide	311
10.16.1 SSH Programming Guide for NetBurner Devices	311
10.16.2 Overview	311
10.16.2.1 NetBurner SSH Implementation	311
10.16.2.2 Key Benefits of wolfSSH	311
10.16.3 What's New in NNDK 3.3.6+	312
10.16.3.1 Major Changes Summary	312
10.16.3.2 Architecture Change	312
10.16.4 Key Types and Formats	313
10.16.4.1 Supported Key Types	313
10.16.4.2 Supported Key Formats	313
10.16.5 Migration Guide	313

10.16.5.1 Project Configuration	313
10.16.5.2 Removed Functions	314
10.16.5.3 Updated Functions	314
10.16.6 Key Management	315
10.16.6.1 Key Installation Methods	315
10.16.6.2 Method 1: Runtime Callback (Highest Priority)	315
10.16.6.3 Method 2: Compiled-In Key (Medium Priority)	316
10.16.6.4 Method 3: Auto-Generated Key (Lowest Priority)	316
10.16.7 User Authentication	317
10.16.7.1 UserAuthManager Class	317
10.16.7.2 Key Features	317
10.16.7.3 Basic Usage	318
10.16.7.4 Storage Callbacks	318
10.16.7.5 Integration with SSH Authentication	319
10.16.7.6 Security Best Practices	320
10.16.8 Configuration	320
10.16.8.1 SSH Server Configuration	320
10.16.8.2 SSH Client Configuration	320
10.16.8.3 Advanced Configuration	321
10.16.8.4 Configuration Constants	321
10.16.9 Examples and Best Practices	321
10.16.9.1 Example 1: Basic SSH Server	321
10.16.9.2 Example 2: SSH Server with Persistent Users	322
10.16.9.3 Example 3: SSH Client	323
10.16.9.4 Example 4: Custom Key Management	323
10.16.9.5 Best Practices	324
10.16.10 Quick Reference	324
10.16.10.1 Essential Functions	324
10.16.10.2 Common Patterns	325
10.16.10.3 Troubleshooting	326
10.16.10.4 Debug Logging	326
10.16.10.5 Migration Checklist	326
10.16.11 Summary	327
10.16.11.1 Key Takeaways	327
10.17 Web Server	328
10.17.1 NetBurner Web Server Guide	328
10.17.2 Overview	328
10.17.2.1 Key Features	328
10.17.3 Basic Web Server Setup	328
10.17.3.1 Project Structure	328
10.17.3.2 Initial index.html	329
10.17.3.3 Static Content	329

10.17.3.4 Adding Resources	329
10.17.3.5 Editing HTML Files in NBEclipse	330
10.17.4 Dynamic Web Content	330
10.17.4.1 CPPCALL Tag - C++ Callback Functions	330
10.17.4.2 VARIABLE Tag - Displaying Variables	331
10.17.4.3 INCLUDE Tag - Linking Variables	333
10.17.5 Advanced Techniques	334
10.17.5.1 Function Callbacks with Parameters	334
10.17.5.2 Custom Data Types	337
10.17.6 Best Practices	339
10.17.6.1 Project Organization	339
10.17.6.2 Code Organization	339
10.17.6.3 Performance Optimization	339
10.17.6.4 Security Considerations	340
10.17.6.5 Error Handling	340
10.17.6.6 HTML Best Practices	341
10.17.6.7 Testing Strategy	341
10.17.6.8 Debugging Tips	342
10.17.6.9 Common Pitfalls to Avoid	342
10.17.7 Quick Reference	342
10.17.7.1 Tag Summary	342
10.17.7.2 Function Signatures	343
10.17.7.3 Common Patterns	343
10.17.8 Summary	343
10.17.8.1 Key Takeaways	343
10.18 Web Sockets	344
10.18.1 WebSocket Protocol Implementation for Real-Time Bidirectional Communication	344
10.18.2 Overview	344
10.18.3 Typical WebSocket Workflow	344
10.18.4 WebSocket Frame Types	345
10.18.5 Buffering and Flow Control	345
10.18.6 Connection Health Monitoring	345
10.18.7 Graceful Connection Closure	345
10.18.8 Thread Safety	346
10.18.9 Resource Limits	346
10.18.10 Error Handling	346
10.18.11 Security Considerations	347
10.18.12 Performance Optimization	347
10.18.13 Debugging WebSocket Issues	347
10.18.14 Configuration and Return Code Definitions	352
10.18.14.1 Number of buffer segments allocated for each WebSocket connection.	352
10.18.14.2 Maximum number of simultaneous WebSocket connections supported.	353

10.18.14.3 Minimum buffer space threshold before triggering buffer management actions.	354
10.18.14.4 Accumulation delay for batching small writes before transmission.	354
10.18.14.5 Timeout value for flush operations (in system ticks).	355
10.18.14.6 WebSocket frame FIN bit mask indicating final fragment.	355
10.18.14.7 WebSocket continuation frame opcode.	356
10.18.14.8 WebSocket text frame opcode for UTF-8 encoded text data.	357
10.18.14.9 WebSocket binary frame opcode for raw binary data.	357
10.18.14.10 WebSocket close frame opcode for connection termination.	359
10.18.14.11 WebSocket ping frame opcode for connection liveness check.	360
10.18.14.12 WebSocket pong frame opcode for ping response.	361
10.18.14.13 WebSocket socket option to enable text frame mode.	361
11 System Diagnostics	365
11.1 Introduction	365
11.2 Server Page with Diagnostics Enabled	365
11.3 Configuration Web Page Display	366
11.4 Adding Your Own Diagnostic Information	369
12 Example Applications	371
12.1 Example Applications Main Page	371
12.2 AES	373
12.3 Application Image Security	374
12.3.1 Examples for application image security, including:	374
12.3.1.1 Examples	374
12.3.2 Application Image Checksum	374
12.3.2.1 Application Image Checksum	374
12.3.3 Board Lock	376
12.3.3.1 Lock an Application to a Specific Device (board)	376
12.3.3.2 Check Lock	376
12.3.3.3 Key Blob	378
12.3.3.4 Sign Board	380
12.3.4 Signed Application	382
12.3.4.1 Signed Application Example	382
12.4 Clear Configuration Flash	385
12.4.1 NetBurner Clear Configuration Flash	385
12.4.1.1 Overview	385
12.4.1.2 Purpose	385
12.4.1.3 Important Limitations	385
12.4.1.4 Reboot Requirement	385
12.4.1.5 How It Works	386
12.4.1.6 Usage Instructions	386
12.4.1.7 Technical Details	386
12.4.1.8 Safety Considerations	386

12.4.1.9 Development Notes	386
12.5 Configuration	387
12.5.1 Application Data	387
12.5.1.1 Basic Config Variables	387
12.5.1.2 Config Class	391
12.5.1.3 Custom Range Limit Config Variables	393
12.5.1.4 Nesting Config Objects	396
12.5.1.5 FS_BulkStore	399
12.5.2 Config and System Params	402
12.5.2.1 Disable System Features	402
12.5.2.2 Modify Boot	405
12.5.2.3 Modify Ethernet Interface	407
12.5.2.4 Show Configuration	414
12.5.3 Config Web Interfaces	416
12.5.3.1 Basic Web Config	417
12.5.3.2 Basic Web Config with SSL/TLS	417
12.5.3.3 Config Tags	418
12.5.3.4 Custom Web Config	422
12.5.3.5 Custom Web System Config	425
12.5.3.6 HTML Only Configuration	427
12.5.4 Detached Config Object	430
12.5.4.1 Detached Config Example	430
12.5.5 Change Notify	432
12.5.5.1 Config Notify Application	432
12.5.6 Config Class Receive Notification	434
12.5.6.1 Config Notify Class Example	434
12.6 DNS Device Name	437
12.6.1 DNS Device Name Application	437
12.6.1.1 Overview	437
12.6.1.2 Features	437
12.6.1.3 How It Works	437
12.6.1.4 Device Naming	437
12.6.1.5 Network Integration	438
12.6.1.6 Sample Output	438
12.6.1.7 Key Functions	438
12.6.1.8 Important Notes	439
12.6.1.9 Building and Deployment	439
12.6.1.10 Use Cases	439
12.6.1.11 Additional Information on DNS Naming	439
12.7 DHCP	440
12.7.1 DHCP Client - Change IP Address	440
12.7.1.1 DHCP Client - Change IP Address Application	440

12.7.2 DHCP Client - Test DHCP	443
12.7.2.1 DHCP Test Application	443
12.7.3 DHCP Server (DHCPD)	446
12.7.3.1 DHCP Server Application (DHCPD)	446
12.7.4 DHCP Client - Change IP Address Via Webpage	448
12.7.4.1 DHCP Client - Change IP Address Via Webpage	448
12.8 DNS Client	451
12.8.1 DNS Client Application	451
12.8.1.1 Overview	451
12.8.1.2 Features	452
12.8.1.3 Application Functions	452
12.8.1.4 Prerequisites	452
12.8.1.5 Usage Instructions	453
12.8.1.6 Error Handling	453
12.8.1.7 Network Interface Diagnostics	453
12.8.1.8 Compilation Options	453
12.8.1.9 Technical Details	453
12.8.1.10 Troubleshooting	454
12.8.1.11 Application Flow	454
12.9 Embedded Flash File System (EFFS)	454
12.9.1 FAT File System (EFFS-FAT)	455
12.9.1.1 Application Update	455
12.9.1.2 Basic	455
12.9.1.3 FTP	458
12.9.1.4 HTTP	461
12.9.1.5 HTML Variables	465
12.9.1.6 Multiple Tasks	466
12.9.1.7 Performance Tests	466
12.9.1.8 RAM Drive	467
12.9.2 Standard File System (EFFS-STD)	467
12.9.2.1 EFFS-STD HTTP	467
12.10 Ethernet	470
12.10.1 Manual Configuration	471
12.10.1.1 Ethernet Manual Configuration Test Application	471
12.11 Exception Try/Catch	473
12.11.1 Exception Try/Catch Application	473
12.11.1.1 Overview	473
12.11.1.2 Application Details	473
12.11.1.3 Key Features	473
12.11.1.4 Output	474
12.11.1.5 Compilation Requirements	474
12.11.1.6 Dependencies	474

12.11.1.7 Usage	474
12.11.1.8 Development Notes	474
12.12 External IRQ	474
12.12.1 External IRQ Example	474
12.12.1.1 Overview	474
12.12.1.2 Supported Platforms	475
12.12.1.3 Hardware Setup	475
12.12.1.4 How It Works	476
12.12.1.5 User Interface	476
12.12.1.6 Configuration Options	476
12.12.1.7 Building and Running	477
12.12.1.8 Key Functions Used	477
12.12.1.9 Important Notes	477
12.12.1.10 Troubleshooting	477
12.13 Extra FD Circular Buffer	477
12.13.1 NetBurner Extra File Descriptor Circular Buffer	477
12.13.1.1 Overview	477
12.13.1.2 Key Features	478
12.13.1.3 Architecture	478
12.13.1.4 Configuration Options	478
12.13.1.5 Usage Example	479
12.13.1.6 Test Functions	479
12.13.1.7 File Descriptor Status Management	479
12.13.1.8 Thread Safety	479
12.13.1.9 Debug Features	479
12.13.1.10 Configuration	479
12.13.1.11 Integration Notes	480
12.13.1.12 Files	480
12.14 FileDescriptor	480
12.15 Extra FD Circular Buffer	480
12.15.1 NetBurner Extra File Descriptor Circular Buffer	480
12.15.1.1 Overview	480
12.15.1.2 Key Features	480
12.15.1.3 Architecture	481
12.15.1.4 Configuration Options	481
12.15.1.5 Usage Example	481
12.15.1.6 Test Functions	482
12.15.1.7 File Descriptor Status Management	482
12.15.1.8 Thread Safety	482
12.15.1.9 Debug Features	482
12.15.1.10 Configuration	482
12.15.1.11 Integration Notes	482

12.15.1.12 Files	483
12.15.2 FDBuffer	483
12.15.2.1 NetBurner FDBuffer Example	483
12.15.3 fdprintf - printf to file descriptor	485
12.15.3.1 fdprintf Example Application	485
12.15.4 FdToNBString	487
12.15.4.1 File Descriptor to NBString Example	487
12.16 FTP	489
12.16.1 FTP Client	490
12.16.1.1 FTP Client Example	490
12.16.2 Display HTML Files (FTPD)	493
12.16.2.1 FTPD HTML Expose Application	493
12.16.3 Simple FTP Server (FTPD)	496
12.16.3.1 Simple FTP Server (FTPD) - Trivial Implementation	496
12.17 GDB Debugger	499
12.17.1 NetBurner GDB Demo Application	499
12.17.1.1 Overview	499
12.17.1.2 Application Description	499
12.17.1.3 Features	499
12.17.1.4 Hardware Requirements	499
12.17.1.5 Software Requirements	500
12.17.1.6 Application Behavior	500
12.17.1.7 Console Output	500
12.17.1.8 Usage	500
12.17.1.9 Development Notes	500
12.17.1.10 File Structure	500
12.17.1.11 Building and Deployment	500
12.18 General Purpose I/O	501
12.18.1 GPIO Blink	501
12.18.1.1 GPIO Blink Application	501
12.18.2 General Purpose I/O Service	503
12.18.2.1 GPIO Service Application	503
12.19 I2C	506
12.19.1 PAC193X	506
12.19.1.1 PAC193x Energy Monitor	506
12.19.2 PicKit Serial I2C Demo Board	509
12.19.2.1 PicKit I2C Serial Demo Board Application	509
12.19.3 RV3032 I2C Real-Time Clock	512
12.19.3.1 There are multiple I2C peripherals on each NetBurner platform as indicated in the tables below. The first	512
12.19.4 Wire Address Scan	513
12.19.4.1 I2C Wire Scan Application	513

12.19.5 SparkFun Qwiic Alphanumeric Display	515
12.19.5.1 SparkFun Qwiic Alphanumeric Display Application	515
12.19.6 SparkFun Qwiic Joystick	517
12.19.6.1 SparkFun Qwiic Alphanumeric Display Application	517
12.19.7 SparkFun Qwiic KeyPad	521
12.19.7.1 SparkFun Qwiic KeyPad Application	521
12.19.8 SparkFun Qwiic OLED Display	523
12.19.8.1 SparkFun Qwiic OLED Display - Analog Clock	523
12.20 IP Address Object (IPADDR)	526
12.20.1 FunWithIPADDR - NetBurner IP Address Object Example	526
12.20.1.1 Overview	526
12.20.1.2 What is IPADDR?	526
12.20.1.3 Application Functionality	526
12.20.1.4 Code Examples	527
12.20.1.5 Important Notes	527
12.20.1.6 Application Flow	527
12.20.1.7 Build Requirements	527
12.20.1.8 System Features Used	528
12.21 IPv6	528
12.21.1 DHCPv6	528
12.21.1.1 DHCPv6 Client Application	528
12.22 JSON	531
12.22.1 JSON Lexer	532
12.22.1.1 JSON Array	532
12.22.1.2 JSON Array with Objects	534
12.22.1.3 Parse Test	535
12.22.2 NetBurner Demo Server for JSON Client Examples	538
12.22.2.1 NetBurner Demo Server	538
12.22.3 Get JSON Object From Server	541
12.22.3.1 Get JSON From Server Application	541
12.22.4 Get Non-JSON Object From Server	544
12.22.4.1 Get Non-JSON From Server	544
12.22.5 JSON Linter Test	546
12.22.5.1 JSON Linter Test Application	546
12.22.6 Post JSON to Server and Display Result	550
12.22.6.1 JSON POST Application	550
12.22.7 Post JSON to Server	552
12.22.7.1 NetBurner JSON POST Application	552
12.22.8 Get Application Image and Load it From a Web Server	555
12.22.8.1 LoadCodeFromServer	555
12.22.9 Simple JSON HTML Example	557
12.22.9.1 NetBurner Simple JSON HTML Example	557

12.22.10 Simple JSON Post Receiver Example	560
12.22.10.1 Simple JSON Post Receiver Example	560
12.23 malloc	563
12.23.1 Malloc Example Application	563
12.23.1.1 Overview	563
12.23.1.2 Purpose	563
12.23.1.3 Key Features	563
12.23.1.4 Application Flow	564
12.23.1.5 Memory Fragmentation Example	564
12.23.1.6 Key Observations	564
12.23.1.7 Code Structure	565
12.23.1.8 Usage Notes	565
12.23.1.9 Educational Value	565
12.23.1.10 Platform Requirements	565
12.24 MQTT Examples	565
12.24.1 MQTT Config Object Example	566
12.24.1.1 MQTT Config Object Application	566
12.24.2 MQTT Custom Object Example	569
12.24.2.1 MQTT Custom Object Example	569
12.24.3 MQTT Object I/O Example	571
12.24.3.1 MQTT Object I/O Application	571
12.24.4 MQTT Simple Object Pub/Sub Example	574
12.24.4.1 MQTT Simple Object Pub/Sub Application	574
12.24.5 MQTT SubHandler GPIO Example	576
12.24.6 Advanced MQTT Periodic Example	576
12.24.6.1 MQTT Periodic Publish Example	576
12.24.7 Advanced MQTT Pub/Sub Example	578
12.24.7.1 MQTT Advanced Pub/Sub Example	578
12.24.8 Advanced MQTT Echo Example	581
12.24.8.1 MQTT Echo Application	581
12.25 Multicast	584
12.25.1 Multicast Example Application	584
12.25.1.1 Overview	584
12.25.1.2 Features	584
12.25.1.3 Application Components	584
12.25.1.4 How It Works	585
12.25.1.5 Usage Instructions	585
12.25.1.6 Technical Details	585
12.25.1.7 Important Notes	586
12.25.1.8 Troubleshooting	586
12.25.1.9 Dependencies	586
12.26 Multihome	586

12.26.1 Multihome Application	586
12.26.1.1 Overview	586
12.26.1.2 Features	587
12.26.1.3 Prerequisites	587
12.26.1.4 Application Behavior	587
12.26.1.5 Code Structure	588
12.26.1.6 Usage	588
12.26.1.7 Network Configuration	588
12.26.1.8 Error Handling	588
12.26.1.9 System Requirements	588
12.27 NB Approve Shutdown	589
12.27.1 NBApproveShutdown Application	589
12.27.1.1 Overview	589
12.27.1.2 Purpose	589
12.27.1.3 System Requirements	589
12.27.1.4 Application Features	589
12.27.1.5 How It Works	590
12.27.1.6 Usage Instructions	590
12.27.1.7 Code Structure	590
12.27.1.8 Implementation Notes	591
12.27.1.9 Example Output	591
12.27.1.10 Development Guidelines	591
12.28 NSString Class	591
12.29 Network Time Server (NTP) Client	592
12.29.1 NTP Client Example Application	592
12.29.1.1 Overview	592
12.29.1.2 Features	592
12.29.1.3 Prerequisites	592
12.29.1.4 Time Zone Features	592
12.29.1.5 Application Flow	593
12.29.1.6 Time Display Format	593
12.29.1.7 Output Examples	593
12.29.1.8 Key Functions	593
12.29.1.9 Technical Details	594
12.29.1.10 Usage Instructions	594
12.29.1.11 Status Information	594
12.29.1.12 Troubleshooting	594
12.29.1.13 Development Notes	594
12.30 NTP & Real-Time Clock	595
12.30.1 NTP-System-RTC Demo Application	595
12.30.1.1 Overview	595
12.30.1.2 Hardware Requirements	595

12.30.1.3 Features	595
12.30.1.4 Application Menu	596
12.30.1.5 Technical Details	596
12.30.1.6 Network Requirements	596
12.30.1.7 Time Zone Configuration	597
12.30.1.8 Build and Deployment	597
12.30.1.9 Usage Scenarios	597
12.30.1.10 Limitations	597
12.30.1.11 Applications	597
12.31 Overload Directory & System Files	598
12.31.1 NetBurner Overload Directory Example	598
12.31.1.1 Overview	598
12.31.1.2 Application Description	598
12.31.1.3 Key Features	598
12.31.1.4 Directory Structure	599
12.31.1.5 Files in This Example	599
12.31.1.6 Setup Instructions for NBEclipse Projects	599
12.31.1.7 Building and Maintenance	600
12.31.1.8 Application Behavior	600
12.31.1.9 Important Notes	600
12.32 Ping	600
12.32.1 Ping Example	600
12.32.1.1 Overview	600
12.32.1.2 Features	600
12.32.1.3 How It Works	601
12.32.1.4 Key Functions	601
12.32.1.5 Technical Details	601
12.32.1.6 Usage Example	602
12.32.1.7 Error Handling	602
12.32.1.8 System Requirements	602
12.32.1.9 Development Notes	602
12.32.1.10 Purpose	602
12.33 PLATFORM SPECIFIC	603
12.33.1 MOD5441x, NANO54415, SB800EX	603
12.33.1.1 1-Wire	604
12.33.1.2 Analog to Digital	607
12.33.1.3 CAN to Serial	617
12.33.1.4 Digital to Analog (DAC)	624
12.33.1.5 DSPI to Serial	629
12.33.1.6 Load Application From Flash Card	635
12.33.1.7 SDHC Flash Card	640
12.33.1.8 EMI Emissions Reduction	648

12.33.1.9 External NMI IRQ 7	650
12.33.1.10 RTC - External	655
12.33.1.11 I2C	655
12.33.1.12 RTC - On Chip	669
12.33.1.13 Periodic Interrupt Timer	669
12.33.1.14 Pulse Generator and Counter	669
12.33.1.15 PWM	670
12.33.1.16 Core Watchdog Timer (CWT)	670
12.33.1.17 WAV File Audio Player	670
12.33.1.18 MOD5441x Only	671
12.33.1.19 NANO54415 Only	703
12.33.1.20 SB800EX Only	703
12.33.2 MODM7AE70, SBE70LC	704
12.33.2.1 GPIO Command Server	704
12.33.2.2 I2C	704
12.33.2.3 Serial Callback Function	711
12.33.2.4 Timers	711
12.33.2.5 Watchdog Timer	720
12.33.2.6 MODM7AE70	720
12.33.2.7 SBE70LC	725
12.33.3 SOMRT1061	726
12.33.3.1 CAN	726
12.33.3.2 Clock Module Example	729
12.33.3.3 EDFS Multiple Partitions	734
12.33.3.4 External Real Time Clock	734
12.33.3.5 Periodic Interrupt Timer (PIT)	734
12.33.3.6 Open All UARTs	735
12.33.3.7 SOMRT1061 Pin IRQ Example	736
12.33.3.8 PWM	738
12.33.3.9 SEMC Simple	739
12.33.3.10 SEMC to SRAM, FPGA, LCD	742
12.33.3.11 Analog to Digital (ADC)	773
12.33.3.12 SPI PickIt Demo Board	773
12.33.3.13 On-Chip Temperature Sensor (TEMPMON)	776
12.33.3.14 Dev Board Temperature Sensor and ADC	776
12.34 PPP	776
12.34.1 PPP Server Application	776
12.34.1.1 Overview	776
12.34.1.2 Features	776
12.34.1.3 Hardware Requirements	776
12.34.1.4 Connection Types	777
12.34.1.5 Configuration	777

12.34.1.6 Authentication	777
12.34.1.7 Interactive Commands	777
12.34.1.8 PPP State Machine	778
12.34.1.9 Error Handling	778
12.34.1.10 IP Address Utilities	778
12.34.1.11 Web Interface	779
12.34.1.12 File Structure	779
12.34.1.13 Build Requirements	779
12.34.1.14 Usage Instructions	779
12.34.1.15 Testing PPP Connections	780
12.34.1.16 Development Notes	780
12.34.1.17 Troubleshooting	780
12.35 Profiler	781
12.35.1 Profiler Example Application	781
12.35.1.1 Overview	781
12.35.1.2 Features	781
12.35.1.3 Profiling Functionality	781
12.35.1.4 Menu Commands	781
12.35.1.5 Setup and Configuration	781
12.35.1.6 Application Flow	782
12.35.1.7 Network Features	782
12.35.1.8 Development Notes	782
12.35.1.9 Output Examples	782
12.35.1.10 Compilation Requirements	782
12.35.1.11 Usage	783
12.36 RTOS	783
12.36.1 MaxTasks	783
12.36.1.1 MaxTasks RTOS Example	783
12.36.2 OSCrit - Critical Section	786
12.36.2.1 OSCrit - Critical Section Example	786
12.36.3 OSFifo - FIFO	788
12.36.3.1 RTOS FIFO Example Application	788
12.36.4 OSFlags	790
12.36.4.1 OSFlags RTOS Example	790
12.36.5 OSMailbox	793
12.36.5.1 RTOS Mailbox Example	793
12.36.6 OSQueue	794
12.36.7 OSSemaphore	795
12.36.8 OSTaskCreate	795
12.36.9 Round Robin Task Sheduler	795
12.37 Serial	795
12.37.1 Dual TCP To Serial	795

12.37.1.1 DualTCP2Serial Example	795
12.37.2 Serial To Ethernet	798
12.37.2.1 Serial to Ethernet Bridge Application	798
12.37.3 Serial HTTP Get Request	801
12.37.4 Serial Receive Callback	801
12.37.4.1 Serial Receive Callback Example	801
12.37.5 Serial to Serial	803
12.37.5.1 Serial to Serial Example	803
12.37.6 Serial Application Update	805
12.37.6.1 Overview	805
12.37.6.2 Methods	805
12.37.7 TCP to Serial	805
12.37.7.1 TCP to Serial Bridge	805
12.37.8 Serial Webserver	808
12.37.8.1 Serial Webserver	808
12.38 SHA1 Digest	809
12.39 Show Network Interfaces	810
12.39.1 Show Network Interfaces Example	810
12.39.1.1 Overview	810
12.39.1.2 Application Architecture	810
12.39.1.3 Key Features	810
12.39.1.4 Implementation Details	811
12.39.1.5 Web Interface Structure	814
12.39.1.6 Network Information Display	814
12.39.1.7 Usage Instructions	815
12.39.1.8 Advanced Features	816
12.39.1.9 Troubleshooting	817
12.39.1.10 Performance Considerations	818
12.39.1.11 Build Configuration	819
12.39.1.12 Learning Objectives	819
12.39.1.13 Related Examples	820
12.40 SOCKS5 Client	820
12.41 SPI	820
12.41.1 DSPI Multiplex	820
12.41.2 Serial to SPI	820
12.41.3 SPI PicKit Demo Board	820
12.42 SSH	821
12.42.1 SSH Server	821
12.42.1.1 SSH Minimal Server Example	821
12.42.2 SSH Server with User Key	824
12.42.2.1 SSH Server with User Key Example	824
12.42.3 SSH Client	827

12.42.3.1 SSH Minimal Client Example	827
12.42.4 SSH Server with User Authorization	830
12.42.4.1 SSH Server with User Authorization	830
12.42.5 SecureSerToEthFactoryApp	834
12.42.5.1 NetBurner Secure Serial-to-Ethernet Factory Application	834
12.43 SSL/TLS	838
12.43.1 Easy SSL/TLS Onboard Certificates	839
12.43.1.1 ACME Certificate Generation (Easy Servlet)	839
12.43.1.2 On-board Cert Generation - Simple	842
12.43.1.3 On-board Cert Generation - Advanced	845
12.43.1.4 On-board Cert Generation - Compiled Certificate Authority	849
12.43.1.5 ACME Certificate Generation (Easy Servlet)	852
12.43.2 FTPS Server	855
12.43.2.1 FTPS Server Example	855
12.43.3 HTML File Post	859
12.43.3.1 SSL/TLS File Post with Multipart Form	859
12.43.4 HTML Form Post	862
12.43.4.1 NetBurner SSL HTML Form Post Example	862
12.43.5 HTTPS Dual Cert	865
12.43.5.1 NetBurner HTTPS Dual Certificate Server	865
12.43.6 HTTPS Upload Cert	869
12.43.6.1 NetBurner SSL Certificate Upload Application	869
12.43.7 HTTPS Web Server Demo	875
12.43.7.1 SSL/TLS Web Server Demo	875
12.43.8 HTTPS GET Request	878
12.43.8.1 SSL HTTPS GET Example	878
12.43.9 SSL/TLS Client	880
12.43.9.1 SSL/TLS Client Example	880
12.43.10 SSL/TLS Client Certificate	882
12.43.10.1 SSL Client Certificate Example	882
12.43.11 SSL/TLS Client Verify Peer - Basic	885
12.43.11.1 SSL/TLS Client Verify Peer - Basic	885
12.43.12 SSL/TLS Client Verify Peer - EDFS-STD	888
12.43.12.1 SSL/TLS Client Verify Peer - EDFS-STD	888
12.43.13 SSL/TLS Receive Mail (POP3)	892
12.43.13.1 SSL POP3 Mail Example	892
12.43.14 SSL/TLS Send Mail	894
12.43.14.1 SSL/TLS Send Mail Example	894
12.43.15 SSL/TLS Send Mail w/ Attachment	898
12.43.15.1 SSL/TLS SendMail with Attachment	898
12.43.16 SSL/TLS Send Mail w/ EDFS Atchment	901
12.43.16.1 SSL Send Mail with EDFS Attachment	901

12.43.17 SSL/TLS Server	904
12.43.17.1 NetBurner SSL/TLS Server Example	904
12.43.18 HTTPS Configuration Mirror	907
12.43.18.1 SSL Configuration Mirror	907
12.44 Stack Protection	910
12.44.1 Stack Protection Example	910
12.44.1.1 Overview	911
12.44.1.2 Features	911
12.44.1.3 Architecture Support	911
12.44.1.4 Prerequisites	911
12.44.1.5 Application Structure	912
12.44.1.6 Usage	912
12.44.1.7 Expected Output	912
12.44.1.8 Safety Features	912
12.44.1.9 Development Notes	912
12.44.1.10 Important Warnings	913
12.44.1.11 Build Instructions	913
12.44.1.12 Troubleshooting	913
12.45 Syslog	913
12.45.1 NetBurner Syslog Demo	913
12.45.1.1 Overview	913
12.45.1.2 Features	913
12.45.1.3 How It Works	913
12.45.1.4 Setup Instructions	914
12.45.1.5 Code Structure	914
12.45.1.6 Expected Output	915
12.45.1.7 Network Requirements	915
12.45.1.8 Production Considerations	915
12.45.1.9 Troubleshooting	915
12.46 System Diagnostics	915
12.46.1 System Diagnostics Example	915
12.46.1.1 Overview	915
12.46.1.2 Features	915
12.46.1.3 Application Components	916
12.46.1.4 Code Structure	916
12.46.1.5 Usage Instructions	916
12.46.1.6 Adding Custom Variables	916
12.46.1.7 Important Notes	917
12.46.1.8 Output Example	917
12.47 TCP	917
12.47.1 TCP Client	918
12.47.1.1 TCP Client Example Application	918

12.47.2 TCP No Block Connect Test	920
12.47.2.1 TCP No Block Connect Test	920
12.47.3 TCP Keepalive Simple	923
12.47.3.1 TCP Keep-Alive Server	923
12.47.4 TCP Keepalive Multiple	925
12.47.4.1 TCP Multi-Socket Server with Keep-Alive	925
12.47.5 Multiple TCP Interface Test	929
12.47.5.1 TCP Multiple Interface Test	929
12.47.6 TCP Multi-Socket Server	931
12.47.6.1 TCP Multi-Socket Server	931
12.47.7 TCP Resource Information	933
12.47.7.1 TCP Resource Information	933
12.47.8 TCP Server	935
12.47.8.1 Simple TCP Server Example	935
12.47.9 TCP Server Using Via	937
12.47.9.1 TCP Server Via Example	937
12.47.10 TCP Speed Test	940
12.47.10.1 TCP Speed Test Application	940
12.47.11 TCP Stress Test	943
12.47.11.1 TCP Stress Test Application	943
12.48 Telnet Command	945
12.48.1 NetBurner Telnet Command Example	945
12.48.1.1 Overview	945
12.48.1.2 Features	945
12.48.1.3 Configuration	946
12.48.1.4 Callback Functions	946
12.48.1.5 Connection Types and Prompts	947
12.48.1.6 Disconnect Handling	947
12.48.1.7 SSH Support (Optional)	947
12.48.1.8 Usage	947
12.48.1.9 Build Requirements	948
12.48.1.10 System Integration	948
12.48.1.11 Customization	948
12.48.1.12 Security Notes	948
12.48.1.13 Related Documentation	948
12.49 TFTP Client	948
12.49.1 TFTP Client Application	948
12.49.1.1 Overview	948
12.49.1.2 Features	949
12.49.1.3 Requirements	949
12.49.1.4 Application Flow	949
12.49.1.5 Code Structure	949

12.49.1.6 Return Codes	950
12.49.1.7 Usage Instructions	950
12.49.1.8 Test Files	951
12.49.1.9 Network Requirements	951
12.49.1.10 Troubleshooting	951
12.49.1.11 Customization Options	951
12.49.1.12 Development Notes	951
12.50 TFTP Server	952
12.50.1 TFTP Server Application	952
12.50.1.1 Overview	952
12.50.1.2 Application Description	952
12.50.1.3 Key Components	952
12.50.1.4 Functionality	953
12.50.1.5 System Requirements	953
12.50.1.6 Usage	953
12.50.1.7 Development Notes	954
12.50.1.8 File Structure	954
12.50.1.9 Output Information	954
12.50.1.10 Protocol Support	954
12.51 Time Functions	954
12.51.1 Time Functions Application	954
12.51.1.1 Overview	954
12.51.1.2 Features	954
12.51.1.3 File Structure	955
12.51.1.4 Key Time Functions Demonstrated	956
12.51.1.5 Time Zone Support	956
12.51.1.6 Usage Requirements	956
12.51.1.7 Time Precision and Accuracy	957
12.51.1.8 Error Handling	957
12.51.1.9 Example Output	957
12.51.1.10 Development Notes	957
12.51.1.11 Integration	958
12.52 Timers	958
12.52.1 Timer	958
12.52.1.1 HiRes Delay Timer Example	958
12.52.2 Interval Timer	961
12.52.2.1 Interval Timer Example Application	961
12.52.3 Stopwatch Timer	963
12.52.3.1 Stopwatch Example Application	963
12.53 UDP	965
12.53.1 UDP to Serial	965
12.53.1.1 NetBurner UDP to Serial Bridge	965

12.53.2 UDP Notify Callback Function	969
12.53.2.1 UDP Notify Example	969
12.53.3 UDP C++ Packet Class	971
12.53.4 UDP Echo	971
12.53.4.1 UDP C++ Packet Class Example	971
12.53.5 UDP Send/Receive	973
12.53.5.1 UDP Send/Receive Example	973
12.53.6 UDP Sockets	976
12.53.6.1 UDP Sockets Example Application	976
12.53.7 UDP Sockets Via	978
12.53.7.1 UDP Sockets Via Example	978
12.54 VLAN	981
12.54.1 VLAN Example Application	981
12.54.1.1 Overview	981
12.54.1.2 Features	981
12.54.1.3 VLAN Configuration	981
12.54.1.4 Prerequisites	981
12.54.1.5 Serial Console Commands	981
12.54.1.6 Network Interface Information	982
12.54.1.7 Web Interface	983
12.54.1.8 File Structure	983
12.54.1.9 Key Functions	983
12.54.1.10 Building and Deployment	983
12.54.1.11 Network Setup Requirements	983
12.54.1.12 Troubleshooting	984
12.54.1.13 Platform Compatibility	984
12.54.1.14 IPv6 Support	984
12.55 Web Server	984
12.55.1 Ajax Graph	985
12.55.1.1 NetBurner AJAX Graph Application	985
12.55.2 Bootstrap Hello World	987
12.55.2.1 NetBurner Bootstrap HelloWorld Example	987
12.55.3 HTML File Post	990
12.55.3.1 File Post with Multipart Form	990
12.55.4 HTML File Post Task Handoff	992
12.55.4.1 File Post Task Handoff Example	992
12.55.5 Flash Form	995
12.55.5.1 FlashForm Example Application	995
12.55.6 GIF Canvas	997
12.55.6.1 GIF Canvas Example	997
12.55.7 HTML Application File Post	1000
12.55.7.1 NetBurner Application Update Server	1000

12.55.8 HTML Cookie	1002
12.55.8.1 NetBurner HTML Cookie Example	1002
12.55.9 HTML Form Post	1005
12.55.9.1 HtmlFormPost Example	1005
12.55.10 HTML Form Post 2.X Compatible	1014
12.55.10.1 NetBurner HTML Form POST Example	1014
12.55.11 HTML Password	1017
12.55.11.1 Methods to Set an Access Group	1017
12.55.11.2 ACCESSGROUP HTML Tags	1018
12.55.12 HtmlPostDateTime	1018
12.55.12.1 HTML Date Time Post Application	1018
12.55.13 HTML Server GET Request	1021
12.55.13.1 HtmlServerGetRequest Example	1021
12.55.14 HTMLVariables	1030
12.55.14.1 HtmlVariables Example	1030
12.55.15 Ping From Web	1038
12.55.15.1 NetBurner Ping From Web Example	1038
12.55.16 Raw Form Post 2pX	1041
12.55.16.1 Raw Post Callback 2pX	1041
12.55.17 Remote Web Console for Serial/stdout	1043
12.55.17.1 NetBurner Remote Web Console Example	1043
12.55.18 Serial Webserver	1046
12.55.18.1 Serial Webserver	1046
12.55.19 Simple HTML	1048
12.55.19.1 NetBurner Simple HTML Example	1048
12.55.20 TicTacToe	1050
12.55.20.1 NetBurner TicTacToe Web Application	1050
12.56 Web Client	1053
12.56.1 Earthquake	1053
12.56.1.1 Earthquake Monitor Application	1053
12.56.2 Find My IP	1055
12.56.2.1 FindMyIP Application	1055
12.56.3 Find My IP Task	1056
12.56.3.1 Find My IP Application	1056
12.56.4 WebClient Http Request Methods	1058
12.56.4.1 HTTP Methods Test Application	1058
12.56.5 Message Passer	1062
12.56.5.1 MessagePasser	1062
12.56.6 MessagePasserTask JSON	1064
12.56.6.1 MessagePasserTask Application	1064
12.57 Web Sockets	1067
12.57.1 Web Socket Connect	1067

12.57.1.1 WebSocket Connect Application	1067
12.57.2 Web Socket Console	1069
12.57.2.1 WebSocket Console	1069
12.57.3 Web Socket DIP Switch	1072
12.57.3.1 Real-Time DIP Switch States via WebSocket	1072
12.57.4 Web Socket Echo	1075
12.57.4.1 WebSocket Echo Application	1075
12.57.5 HTTPS WebSocket Demo	1078
12.57.5.1 SSL/TLS WebSocket Demo	1078
12.58 WiFi Examples	1081
12.58.1 Wifi Config AP	1081
12.58.1.1 NetBurner WiFi Configuration Access Point	1081
12.58.2 Configure AP with JSON	1084
12.58.2.1 WiFi Configuration Access Point with JSON	1084
12.58.3 Wifi Firmware Update	1087
12.58.3.1 WiFi Firmware Update Application	1087
12.58.4 Wifi Access Point	1088
12.58.4.1 WiFi Access Point Application	1088
12.58.5 Wifi Client	1091
12.58.5.1 WiFi Client Application	1091
12.58.6 Wifi Scan	1093
12.58.6.1 WiFi Scan Application	1093
13 Deprecated List	1097
14 NetBurner API	1099
14.1 AES - Advanced Encryption Standard	1099
14.1.1 Detailed Description	1099
14.1.2 Function Documentation	1099
14.1.2.1 aes_cbc_decrypt()	1099
14.1.2.2 aes_cbc_encrypt()	1101
14.1.2.3 aes_decrypt()	1101
14.1.2.4 aes_encrypt()	1101
14.1.2.5 aes_self_test()	1102
14.1.2.6 aes_set_key()	1102
14.2 ARP - Address Resolution Protocol	1102
14.2.1 Detailed Description	1103
14.2.2 How ARP Works	1104
14.2.3 ARP Cache Management	1104
14.2.4 Function Categories	1104
14.2.5 Example Usage	1104
14.2.5.1 Querying the ARP Cache	1104
14.2.5.2 Displaying ARP Cache Contents	1104

14.2.5.3 Sending Gratuitous ARP	1105
14.2.5.4 IP Address Conflict Detection	1105
14.2.5.5 Network Device Monitoring	1105
14.2.6 Best Practices	1106
14.2.7 Troubleshooting	1106
14.2.8 Important Notes	1106
14.2.9 Protocol Specifications	1106
14.2.10 Typedef Documentation	1107
14.2.10.1 PARP	1107
14.2.11 Function Documentation	1107
14.2.11.1 fShowArp()	1107
14.2.11.2 GetArpMacFromIp()	1108
14.2.11.3 GetProperInterface4()	1110
14.2.11.4 IsMyAddressUsedArpDetect()	1111
14.2.11.5 sendGratuitousArp()	1113
14.2.11.6 ShowArp()	1115
14.2.12 Variable Documentation	1117
14.2.12.1 hard_size	1117
14.2.12.2 hard_Type	1117
14.2.12.3 op_code	1117
14.2.12.4 prot_size	1117
14.2.12.5 prot_Type	1117
14.2.12.6 sender_ip	1117
14.2.12.7 sender_phy	1117
14.2.12.8 target_ip	1117
14.2.12.9 target_phy	1118
14.3 AT Commands	1118
14.3.1 Detailed Description	1119
14.3.2 Typedef Documentation	1119
14.3.2.1 ProcessATcommandFunc	1119
14.4 Base64	1119
14.4.1 Detailed Description	1119
14.4.2 Function Documentation	1120
14.4.2.1 Base64Decode()	1120
14.4.2.2 Base64Encode()	1120
14.4.2.3 Base64StreamEncode()	1121
14.4.2.4 Base64UrlDecode()	1121
14.4.2.5 Base64UrlEncode()	1122
14.4.2.6 Base64UrlStreamEncode()	1122
14.5 Buffers - System Buffer Pool	1123
14.5.1 Detailed Description	1123
14.5.2 Function Documentation	1123

14.5.2.1 FreeBuffer()	1123
14.5.2.2 FreeBufferList()	1124
14.5.2.3 GetBuffer()	1124
14.5.2.4 GetFastBuffer()	1124
14.5.2.5 GetFreeCount()	1125
14.5.2.6 IncUsageCount()	1125
14.5.2.7 InitBuffers()	1125
14.5.2.8 ShowBuffer()	1125
14.5.3 Buffer Ownership States	1126
14.5.3.1 Detailed Description	1126
14.5.4 Buffer State Flags	1126
14.5.4.1 Detailed Description	1126
14.6 CAN	1127
14.6.1 Detailed Description	1127
14.6.2 FlexCAN SOMRT1061	1127
14.6.2.1 Detailed Description	1132
14.6.2.2 Macro Definition Documentation	1132
14.6.2.3 Typedef Documentation	1137
14.6.2.4 Enumeration Type Documentation	1138
14.6.2.5 Function Documentation	1141
14.6.3 MOD5441x, NANO54415, SB800EX	1153
14.6.3.1 Detailed Description	1153
14.6.4 MODM7AE70	1153
14.6.4.1 Detailed Description	1154
14.6.4.2 Macro Definition Documentation	1155
14.6.4.3 MCAN Driver Internals	1155
14.7 Command Processor	1159
14.7.1 Detailed Description	1160
14.7.2 Function Documentation	1160
14.7.2.1 CmdAddCommandFd()	1160
14.7.2.2 CmdListenOnSshPort()	1161
14.7.2.3 CmdListenOnTcpPort()	1161
14.7.2.4 CmdListenQuietOnSshPort()	1161
14.7.2.5 CmdListenQuietOnTcpPort()	1162
14.7.2.6 CmdRemoveCommandFd()	1162
14.7.2.7 CmdStartCommandProcessor()	1163
14.7.2.8 CmdStopListeningOnSshPort()	1164
14.7.2.9 CmdStopListeningOnTcpPort()	1164
14.7.2.10 SendToAll()	1164
14.7.3 Variable Documentation	1165
14.7.3.1 CmdAuthenticateFunc	1165
14.7.3.2 CmdAuthenticateSshFunc	1165

14.7.3.3 CmdChar_func	1166
14.7.3.4 CmdCmd_func	1166
14.7.3.5 CmdConnect_func	1167
14.7.3.6 CmdDisconnect_func	1167
14.7.3.7 CmdIdleTimeout	1167
14.7.3.8 CmdPrompt_func	1167
14.7.4 Command Processor Disconnect Causes	1168
14.7.4.1 Detailed Description	1168
14.7.5 Command Processor Listen Channels	1168
14.7.5.1 Detailed Description	1168
14.7.5.2 Enumeration Type Documentation	1168
14.7.6 Command Processor Response Codes	1168
14.7.6.1 Detailed Description	1169
14.8 Configuration System	1169
14.8.1 Detailed Description	1169
14.8.2 Configuration Network Objects	1169
14.8.2.1 Detailed Description	1169
14.8.3 Configuration Server	1170
14.8.3.1 Detailed Description	1170
14.8.3.2 Function Documentation	1170
14.8.4 Configuration Time Objects	1171
14.8.4.1 Detailed Description	1171
14.8.5 Configuration Variables	1171
14.8.5.1 Detailed Description	1172
14.8.5.2 Function Documentation	1172
14.8.5.3 Configuration Variable Flags	1173
14.9 Constants (constants.h)	1173
14.9.1 Detailed Description	1175
14.9.2 Macro Definition Documentation	1175
14.9.2.1 TICKS_PER_SECOND	1175
14.10 DHCP	1175
14.10.1 Detailed Description	1175
14.10.2 DHCP Server	1176
14.10.2.1 Detailed Description	1176
14.10.2.2 Function Documentation	1176
14.10.3 DHCPv4 Client	1177
14.10.3.1 Detailed Description	1177
14.10.3.2 Function Documentation	1177
14.10.3.3 DHCP State	1178
14.10.4 DHCPv6 Client	1178
14.10.4.1 Detailed Description	1179
14.10.4.2 Function Documentation	1179

14.11 DNS - Domain Name System	1180
14.11.1 Detailed Description	1181
14.11.2 Function Documentation	1181
14.11.2.1 AnyDNSInterFaceActive()	1181
14.11.2.2 CheckDnsCache()	1182
14.11.2.3 fd_dns_part1()	1182
14.11.2.4 fd_dns_processresult()	1182
14.11.2.5 fd_outstanding_Responses()	1183
14.11.2.6 GetHostByName()	1183
14.11.2.7 GetHostByNameVialfNum()	1184
14.11.2.8 IsNameIPAddress()	1185
14.11.3 DNS Record Types	1185
14.11.3.1 Detailed Description	1185
14.11.4 DNS Return Codes	1185
14.11.4.1 Detailed Description	1186
14.12 Device Interrupts	1186
14.12.1 Detailed Description	1186
14.12.2 Enumeration Type Documentation	1186
14.12.2.1 _DeviceInterruptTrigger	1186
14.12.3 Function Documentation	1187
14.12.3.1 DeviceAckInterrupt()	1187
14.12.3.2 DeviceDisableInterrupt()	1187
14.12.3.3 DeviceEnableInterrupt()	1187
14.12.3.4 DeviceGetI2cSem()	1187
14.12.3.5 DeviceGetQspiSem()	1188
14.12.3.6 DevicesValid()	1188
14.12.3.7 DeviceSetupI2c()	1188
14.13 Diagnostics (DIAG)	1189
14.13.1 Detailed Description	1189
14.13.2 Typedef Documentation	1189
14.13.2.1 diagcallback	1189
14.13.3 Function Documentation	1190
14.13.3.1 InitPlatformDiag()	1190
14.13.3.2 SendDiagReport() [1/2]	1190
14.13.3.3 SendDiagReport() [2/2]	1190
14.14 EDFS - Embedded Flash File System	1191
14.14.1 Detailed Description	1191
14.14.2 EDFS-STD Flash File System	1191
14.14.2.1 Detailed Description	1192
14.14.2.2 Macro Definition Documentation	1192
14.14.2.3 STD File System Seek Codes	1198
14.14.3 FAT File System	1199

14.14.3.1 Detailed Description	1200
14.14.3.2 Macro Definition Documentation	1200
14.14.3.3 Function Documentation	1206
14.14.3.4 FAT File System Seek Codes	1207
14.15 Ethernet	1207
14.15.1 Detailed Description	1208
14.15.2 Macro Definition Documentation	1208
14.15.2.1 NO_AUTOMATIC_2ND_ETHERNET	1208
14.15.3 Function Documentation	1208
14.15.3.1 AddEthernetInterfaces()	1208
14.15.3.2 DisablePHY()	1209
14.15.3.3 EnablePHY()	1209
14.15.3.4 ManualEthernetConfig()	1209
14.15.4 Ethernet I/O Counters	1210
14.15.4.1 Detailed Description	1211
14.16 Extra File Descriptors	1211
14.16.1 Detailed Description	1211
14.16.2 Function Documentation	1211
14.16.2.1 FreeExtraFd()	1211
14.16.2.2 GetExtraData()	1211
14.16.2.3 GetExtraFD()	1212
14.16.2.4 GetFreeExtraFDCount()	1212
14.16.2.5 GetFreeSocketCount()	1212
14.16.2.6 GetIoExpandStruct()	1213
14.17 FTP Client	1214
14.17.1 Detailed Description	1215
14.17.2 Function Documentation	1215
14.17.2.1 FTP_CloseSession()	1215
14.17.2.2 FTP_InitializeSession()	1215
14.17.2.3 FTPActiveMode()	1216
14.17.2.4 FTPDeleteDir()	1216
14.17.2.5 FTPDeleteFile()	1216
14.17.2.6 FTPGetCommandResult()	1217
14.17.2.7 FTPGetDir()	1217
14.17.2.8 FTPGetFile()	1218
14.17.2.9 FTPGetFileNames()	1218
14.17.2.10 FTPGetList()	1219
14.17.2.11 FTPMakeDir()	1219
14.17.2.12 FTPPassiveMode()	1220
14.17.2.13 FTPRawCommand()	1220
14.17.2.14 FTPRawStreamCommand()	1220
14.17.2.15 FTPRenameFile()	1221

14.17.2.16 FTPSendFile()	1221
14.17.2.17 FTPSetDir()	1222
14.17.2.18 FTPUpDir()	1222
14.18 FTP Server	1222
14.18.1 Detailed Description	1223
14.18.2 FTP Server Callback	1223
14.18.2.1 Detailed Description	1224
14.18.2.2 Typedef Documentation	1225
14.18.2.3 Function Documentation	1225
14.18.2.4 FTP Server Core Return Codes	1234
14.18.2.5 FTP Server File Return Codes	1234
14.18.3 FTP Server with EDFS-FAT	1234
14.18.4 FTP Server with EDFS-STD	1235
14.19 File Descriptor Printf - fdprintf	1235
14.19.1 Detailed Description	1235
14.19.2 Overview	1235
14.19.3 Function Variants	1235
14.19.4 Example Usage	1236
14.19.4.1 Serial Port Output	1236
14.19.4.2 Network Socket Output	1236
14.19.4.3 Creating Wrapper Functions	1236
14.19.4.4 Integer-Only for Code Size	1236
14.19.5 Format Specifiers	1236
14.19.6 Performance Considerations	1237
14.19.7 Best Practices	1237
14.19.8 Important Notes	1237
14.19.9 Example Code	1238
14.19.10 Function Documentation	1238
14.19.10.1 fdprintf()	1238
14.19.10.2 fdprintf()	1239
14.19.10.3 vfdprintf()	1241
14.19.10.4 vfdprintf()	1243
14.20 GDB Stub (Remote Debugging)	1244
14.20.1 Detailed Description	1245
14.20.2 Function Documentation	1245
14.20.2.1 InitGDBStub()	1245
14.20.2.2 InitGDBStubNoBreak()	1245
14.21 GPIO	1246
14.21.1 Detailed Description	1246
14.21.2 PinIO	1246
14.21.2.1 Detailed Description	1246
14.21.2.2 Variable Documentation	1246

14.21.3 SOMRT1061 Pin Function Constants	1247
14.21.3.1 Detailed Description	1247
14.22 HAL - Hardware Abstraction Layer	1247
14.22.1 Detailed Description	1248
14.22.2 Function Documentation	1248
14.22.2.1 FlashErase()	1248
14.22.2.2 FlashProgram()	1249
14.22.2.3 FlashProgramApplImage()	1249
14.22.2.4 ForceReboot()	1249
14.22.2.5 HalDeviceCertValid()	1250
14.22.2.6 HalEraseDeviceCertAndKey()	1250
14.22.2.7 HalGetDeviceCert()	1250
14.22.2.8 HalGetDeviceCertLen()	1250
14.22.2.9 HalGetDeviceKey()	1251
14.22.2.10 HalGetDeviceKeyLen()	1251
14.22.2.11 HalGetTickFraction()	1251
14.22.2.12 HalSaveNewDeviceCert()	1251
14.22.2.13 HalSaveNewDeviceKey()	1252
14.22.2.14 HalStorage_AddressOffset()	1252
14.22.2.15 HalStorage_Erase()	1252
14.22.2.16 HalStorage_Finalize()	1253
14.22.2.17 HalStorage_GetAllocated()	1253
14.22.2.18 HalStorage_GetMaxAllocation()	1253
14.22.2.19 HalStorage_Prepare()	1254
14.22.2.20 HalStorage_RemainingSpace()	1254
14.22.2.21 HalStorage_Save()	1254
14.22.2.22 HalStorage_SavePartial()	1255
14.22.2.23 HalStorage_WriteOffset()	1255
14.22.2.24 HardwareSetup()	1255
14.22.2.25 PostConfigHardwareInit()	1256
14.22.2.26 spaceleft()	1256
14.22.2.27 StdioCheckIntc()	1256
14.22.2.28 SysLogCheckIntc()	1256
14.22.3 Variable Documentation	1257
14.22.3.1 HalTickMaxCount	1257
14.22.3.2 watchdog_service_function	1257
14.23 HTTP and HTML Functions	1257
14.23.1 Detailed Description	1259
14.23.2 Typedef Documentation	1259
14.23.2.1 http_gethandlerfunc	1259
14.23.2.2 http_posthandler	1260
14.23.3 Enumeration Type Documentation	1260

14.23.3.1 HTTP_ACCESS	1260
14.23.3.2 HTTP_REQ_RD_RESULT	1260
14.23.3.3 HTTP_RequestTypes	1261
14.23.4 Function Documentation	1261
14.23.4.1 BadRequestResponse()	1261
14.23.4.2 CheckHttpAccess()	1261
14.23.4.3 ConfigRenderFunc() [1/2]	1262
14.23.4.4 ConfigRenderFunc() [2/2]	1262
14.23.4.5 EmptyResponse()	1262
14.23.4.6 EnableRemoteConsole()	1263
14.23.4.7 ForbiddenResponse()	1263
14.23.4.8 GetRecordFromName()	1263
14.23.4.9 httpstricmp()	1264
14.23.4.10 NoContentResponse()	1264
14.23.4.11 NotAvailableResponse()	1264
14.23.4.12 NotFoundResponse()	1265
14.23.4.13 RedirectResponse()	1265
14.23.4.14 SendEmailResponse()	1265
14.23.4.15 SendFileFragment()	1266
14.23.4.16 SendFullResponse() [1/2]	1266
14.23.4.17 SendFullResponse() [2/2]	1267
14.23.4.18 SendGifHeader()	1267
14.23.4.19 SendHeaderResponse() [1/2]	1267
14.23.4.20 SendHeaderResponse() [2/2]	1269
14.23.4.21 SendHTMLHeader()	1269
14.23.4.22 SendHTMLHeaderWCookie()	1270
14.23.4.23 SendTextHeader()	1270
14.23.4.24 StartHttp()	1270
14.23.4.25 StartHttps()	1271
14.23.4.26 StopHttp()	1271
14.23.4.27 writeallsafestring()	1271
14.23.4.28 WriteHtmlVariable()	1271
14.23.4.29 writesafestring()	1272
14.24 I2C	1272
14.24.1 Detailed Description	1272
14.24.2 Bit Bang I2C (GPIO)	1273
14.24.2.1 Detailed Description	1273
14.24.3 MOD5441x, NANO54415, SB800EX	1273
14.24.3.1 Detailed Description	1274
14.24.3.2 Function Documentation	1275
14.24.4 MODM7AE70, SBE70LC	1279
14.24.4.1 Detailed Description	1279

14.24.5 Wire Interface	1279
14.24.5.1 Detailed Description	1279
14.25 IOSYS - I/O System	1279
14.25.1 Detailed Description	1282
14.25.2 Typedef Documentation	1282
14.25.2.1 FDCallBack	1282
14.25.3 Enumeration Type Documentation	1282
14.25.3.1 FDChangeType	1282
14.25.4 Function Documentation	1283
14.25.4.1 charavail()	1283
14.25.4.2 close()	1283
14.25.4.3 CurrentStdioFD()	1283
14.25.4.4 dataavail()	1284
14.25.4.5 FD_CLR()	1284
14.25.4.6 FD_CLRFROMSET()	1284
14.25.4.7 FD_COPY()	1285
14.25.4.8 FD_ISSET()	1285
14.25.4.9 FD_OVERLAP()	1286
14.25.4.10 FD_SET()	1286
14.25.4.11 FD_SETFROMSET()	1286
14.25.4.12 FD_ZERO()	1287
14.25.4.13 getchar()	1288
14.25.4.14 haserror()	1288
14.25.4.15 ioctl()	1288
14.25.4.16 peek()	1289
14.25.4.17 read()	1289
14.25.4.18 readall()	1290
14.25.4.19 ReadAllWithTickTimeout()	1290
14.25.4.20 ReadAllWithTimeout()	1291
14.25.4.21 ReadWithTickTimeout()	1292
14.25.4.22 ReadWithTimeout()	1292
14.25.4.23 RegisterFDCallBack()	1293
14.25.4.24 ReplaceStdio()	1293
14.25.4.25 select()	1294
14.25.4.26 SockPeekWithTimeout()	1295
14.25.4.27 write()	1295
14.25.4.28 writeall()	1296
14.25.4.29 writeavail()	1296
14.25.4.30 writestring()	1297
14.25.4.31 ZeroWaitSelect()	1297
14.26 IP	1298
14.26.1 Detailed Description	1302

14.26.2 Typedef Documentation	1302
14.26.2.1 FragmentProcessFunction	1302
14.26.2.2 IpHeaderIPv4	1303
14.26.2.3 PromiscuousPacketFunc	1303
14.26.3 Function Documentation	1304
14.26.3.1 AddInterface()	1304
14.26.3.2 FixHeaderAndSend()	1305
14.26.3.3 FixHeaderAndSendViaInterface()	1305
14.26.3.4 GetEphemeralPort()	1306
14.26.3.5 GetIpPkt() [1/3]	1306
14.26.3.6 GetIpPkt() [2/3]	1306
14.26.3.7 GetIpPkt() [3/3]	1307
14.26.3.8 GetLocalIPInterface()	1307
14.26.3.9 GetLocalIPInterface4()	1307
14.26.3.10 GetLocalIPInterface6()	1308
14.26.3.11 GetMultiHomeInterface()	1308
14.26.3.12 GetProperInterface4()	1308
14.26.3.13 GetSrcIp()	1309
14.26.3.14 GetSrcIp4()	1309
14.26.3.15 GetSrcIp6()	1310
14.26.3.16 GetSrcIp6Wlf()	1310
14.26.3.17 GetSrcIpWlf()	1311
14.26.3.18 GetSrcIpWlf4()	1311
14.26.3.19 GetSum()	1312
14.26.3.20 GetSum20()	1312
14.26.3.21 GetSumHdr()	1313
14.26.3.22 GetUdpPkt()	1313
14.26.3.23 InitializeStack()	1314
14.26.3.24 IpProcessEthernetPacket()	1314
14.26.3.25 IsLocal4()	1314
14.26.3.26 IsMyIp4()	1315
14.26.3.27 IsMyIp6()	1315
14.26.3.28 Ping()	1316
14.26.3.29 Ping4()	1316
14.26.3.30 Ping4ViaInterface()	1317
14.26.3.31 Ping6()	1318
14.26.3.32 Ping6ViaInterface()	1318
14.26.3.33 PingViaInterface()	1319
14.26.3.34 SendICMPError()	1320
14.26.3.35 ShowIPBuffer()	1320
14.26.4 Variable Documentation	1321
14.26.4.1 pFragFunc	1321

14.26.4.2 pPromiscuousPacketFunc	1321
14.26.4.3 pTaskPacketFunc	1321
14.26.5 IPADDR4 Class	1321
14.26.5.1 Detailed Description	1322
14.26.6 IPADDR6 Class	1322
14.26.6.1 Detailed Description	1322
14.27 IPv6 Interface Management	1323
14.27.1 Detailed Description	1323
14.27.2 Function Documentation	1323
14.27.2.1 ShowIP6Counters()	1323
14.27.3 IPv6 DNS	1323
14.27.3.1 Detailed Description	1323
14.27.4 IPv6 Data Structures	1323
14.27.4.1 Detailed Description	1324
14.27.4.2 Enumeration Type Documentation	1324
14.27.5 IPv6 Frame Processing	1324
14.27.5.1 Detailed Description	1324
14.27.6 IPv6 Multicast Constants	1325
14.27.6.1 Detailed Description	1325
14.27.7 IPv6 Neighbor Discovery	1325
14.27.7.1 Detailed Description	1325
14.27.7.2 Enumeration Type Documentation	1326
14.27.8 IPv6 Routing	1326
14.27.8.1 Detailed Description	1326
14.27.8.2 Enumeration Type Documentation	1326
14.28 Initialization - System Initialization Functions	1327
14.28.1 Detailed Description	1328
14.28.2 Function Documentation	1328
14.28.2.1 EnableSecureConfigServer()	1328
14.28.2.2 EnableSystemDiagnostics()	1328
14.28.2.3 init()	1328
14.28.2.4 StartHttp()	1329
14.28.2.5 StartHttpCustom()	1329
14.28.2.6 StartHttps()	1329
14.28.2.7 WaitForActiveNetwork()	1330
14.29 JSON Lexer	1330
14.29.1 Detailed Description	1331
14.29.2 Typedef Documentation	1331
14.29.2.1 CharOutputFn	1331
14.29.3 Enumeration Type Documentation	1331
14.29.3.1 json_primitive_type	1331
14.30 MQTT	1332

14.30.1 Detailed Description	1333
14.31 Multicast	1333
14.31.1 Detailed Description	1333
14.31.2 Function Documentation	1333
14.31.2.1 RegisterMulticastFifo4()	1333
14.31.2.2 RegisterMulticastFifo6()	1334
14.31.2.3 UnregisterMulticastFifo4()	1334
14.31.2.4 UnregisterMulticastFifo6()	1335
14.32 Multihome and VLAN	1335
14.32.1 Detailed Description	1335
14.32.2 Function Documentation	1335
14.32.2.1 AddInterface() [1/3]	1335
14.32.2.2 AddInterface() [2/3]	1336
14.32.2.3 AddInterface() [3/3]	1336
14.32.2.4 AddVlanInterface() [1/3]	1336
14.32.2.5 AddVlanInterface() [2/3]	1337
14.32.2.6 AddVlanInterface() [3/3]	1337
14.33 NBRTOS Real Time Operating System	1338
14.34 NBString - NetBurner String Class	1338
14.34.1 Detailed Description	1338
14.35 Network Interfaces	1338
14.35.1 Detailed Description	1339
14.35.2 Function Documentation	1340
14.35.2.1 DisableMulticast()	1340
14.35.2.2 EnableMulticast()	1340
14.35.2.3 GetFirstInterface()	1340
14.35.2.4 GetInterfaceBlock()	1340
14.35.2.5 GetInterfaceForMyAddress4()	1341
14.35.2.6 GetInterfaceLink()	1341
14.35.2.7 GetInterfaceNumber()	1341
14.35.2.8 GetNextInterface()	1342
14.35.2.9 InterfaceLinkActive()	1342
14.35.2.10 InterfaceLinkDuplex()	1342
14.35.2.11 InterfaceLinkSpeed()	1342
14.35.2.12 InterfaceMAC()	1343
14.35.2.13 InterfaceName()	1343
14.35.2.14 Removeinterface()	1343
14.35.3 IPADDR4 Functions	1344
14.35.3.1 Detailed Description	1344
14.35.3.2 Function Documentation	1344
14.35.4 Low Level Processing (NetDoRx)	1346
14.35.4.1 Detailed Description	1347

14.35.4.2 Typedef Documentation	1347
14.35.4.3 Function Documentation	1348
14.35.4.4 Variable Documentation	1350
14.36 PEM Conversion Utilities	1350
14.36.1 Detailed Description	1351
14.36.1.1 Supported PEM Types	1351
14.36.1.2 Typical Use Cases	1351
14.36.2 Macro Definition Documentation	1351
14.36.2.1 CONVERT_BUFFER_LENGTH_MAX	1351
14.36.3 Function Documentation	1352
14.36.3.1 convertLittleEndianDword()	1352
14.36.3.2 Byte Order Formats	1352
14.36.3.3 convertLittleEndianWord()	1353
14.36.3.4 Byte Order Formats	1353
14.36.3.5 ConvertPEMFormattedData()	1354
14.36.3.6 PEM Format Structure	1355
14.36.4 PEM Data Type Identifiers	1358
14.36.4.1 Detailed Description	1358
14.36.4.2 Macro Definition Documentation	1359
14.36.5 PEM Format Footers	1360
14.36.5.1 Detailed Description	1360
14.36.6 PEM Format Headers	1360
14.36.6.1 Detailed Description	1360
14.37 POP3 - Post Office Protocol	1360
14.37.1 Detailed Description	1361
14.37.2 Function Documentation	1361
14.37.2.1 GetPOPErrorString()	1361
14.37.2.2 POP3_CloseSession()	1361
14.37.2.3 POP3_DeleteCmd()	1362
14.37.2.4 POP3_InitializeSession()	1363
14.37.2.5 POP3_ListCmd()	1363
14.37.2.6 POP3_RetrieveMessage()	1364
14.37.2.7 POP3_StatCmd()	1364
14.37.2.8 POPGetResultCode()	1365
14.37.2.9 SSL_POP3_InitializeSession()	1365
14.37.3 POP3 Return Codes	1365
14.37.3.1 Detailed Description	1366
14.38 PPP - Point to Point Protocol	1366
14.38.1 Detailed Description	1366
14.38.2 PPP Connection State	1366
14.38.2.1 Detailed Description	1366
14.38.2.2 Enumeration Type Documentation	1367

14.38.3 PPP Return Codes	1367
14.38.3.1 Detailed Description	1368
14.39 Pit	1368
14.39.1 Detailed Description	1368
14.39.2 Typedef Documentation	1368
14.39.2.1 pit_chnl_t	1368
14.39.2.2 pit_config_t	1369
14.39.3 Enumeration Type Documentation	1369
14.39.3.1 _pit_chnl	1369
14.39.3.2 _pit_interrupt_enable	1369
14.39.3.3 _pit_status_flags	1369
14.39.4 Function Documentation	1370
14.39.4.1 PIT_Deinit()	1370
14.39.4.2 PIT_Init()	1370
14.40 Platform Specific API	1370
14.40.1 Detailed Description	1370
14.40.2 MOD54415, MOD54417, NANO54415, SB800EX	1371
14.40.2.1 Detailed Description	1371
14.40.2.2 Interrupt Macro: MOD5441x,NANO54415, SB800EX	1371
14.40.3 MODM7AE70, SBE70LC	1385
14.40.3.1 Detailed Description	1385
14.40.3.2 EBI - External Bus Interface	1385
14.40.4 SOMRT1061	1404
14.40.4.1 Detailed Description	1404
14.40.4.2 QTMR (Quad Timer)	1404
14.40.4.3 SEMC	1417
14.40.4.4 SOMRT1061 Temperature Monitor	1432
14.41 SMTP - Send Email	1434
14.41.1 Detailed Description	1435
14.41.2 Enumeration Type Documentation	1435
14.41.2.1 CONTENT_TYPE_ENUM	1435
14.41.3 Function Documentation	1435
14.41.3.1 IsMailError()	1435
14.41.3.2 PrintNBEError()	1435
14.41.3.3 PrintServerLog()	1436
14.41.3.4 SendMail()	1436
14.41.3.5 SendMailAuth()	1437
14.41.3.6 SendMailAuthAddMIME()	1437
14.41.3.7 SendMailAuthEndMIME()	1438
14.41.3.8 SendMailAuthStartMIME()	1438
14.41.3.9 SendMailEx()	1439
14.41.4 MIME Content Types	1440

14.41.5 SMTP Error Codes	1440
14.41.5.1 Detailed Description	1440
14.41.6 SSL SMTP Functions	1440
14.41.6.1 Detailed Description	1441
14.41.6.2 Function Documentation	1441
14.42 SOCKS	1444
14.42.1 Detailed Description	1444
14.42.2 Enumeration Type Documentation	1444
14.42.2.1 SocksAdrType	1444
14.42.2.2 SocksAuthType	1445
14.42.2.3 SocksClientCmd	1445
14.42.3 Function Documentation	1445
14.42.3.1 AuthWithGssApi()	1445
14.42.3.2 GetSocksProxySettings()	1445
14.42.3.3 SetSocksProxySettings()	1446
14.42.4 SOCKS Error Codes	1446
14.42.4.1 Detailed Description	1446
14.43 SPI	1447
14.43.1 Detailed Description	1447
14.43.2 MCF5441x (DSPI)	1447
14.43.2.1 Detailed Description	1450
14.43.2.2 Macro Definition Documentation	1450
14.43.2.3 Enumeration Type Documentation	1453
14.43.2.4 Function Documentation	1454
14.43.3 MCF5441x (Queued SPI 2.x compatibility layer)	1458
14.43.3.1 Detailed Description	1458
14.43.3.2 Function Documentation	1459
14.43.3.3 QSPI state	1461
14.43.4 MIMX10xx (DSPI)	1462
14.43.4.1 Detailed Description	1463
14.43.4.2 Enumeration Type Documentation	1463
14.43.4.3 Function Documentation	1464
14.43.4.4 DspiModuleNumber	1467
14.43.4.5 DspiState	1467
14.43.5 SAME70 (DSPI)	1467
14.43.5.1 Detailed Description	1469
14.43.5.2 Enumeration Type Documentation	1469
14.43.5.3 Function Documentation	1470
14.43.5.4 Variable Documentation	1473
14.43.6 SAME70 (QuadSPI)	1474
14.43.6.1 Detailed Description	1474
14.43.7 SAME70 (USART)	1474

14.43.7.1 Detailed Description	1475
14.43.7.2 UsartModuleNumber	1475
14.44 SSH	1475
14.44.1 Detailed Description	1477
14.44.2 Typedef Documentation	1477
14.44.2.1 sshGetUserKeyFn	1477
14.44.2.2 sshGetUserPwFn	1477
14.44.2.3 sshUserAuthenticateFn	1477
14.44.2.4 sshUserAuthenticateWithTypeFn	1478
14.44.2.5 sshUserGetKeyFn	1478
14.44.3 Function Documentation	1478
14.44.3.1 NbSshInit()	1478
14.44.3.2 SshAccept()	1479
14.44.3.3 SshClientGetUserKeyFn()	1479
14.44.3.4 SshClientGetUserPaswordFn()	1480
14.44.3.5 SshClientSetGetUserKeyFn()	1480
14.44.3.6 SshClientSetGetUserPaswordFn()	1480
14.44.3.7 SshClrSockOption()	1480
14.44.3.8 SshConnect()	1481
14.44.3.9 SshGetKeySize()	1481
14.44.3.10 SshGetSockOption()	1482
14.44.3.11 SshGetUserAuthenticate()	1482
14.44.3.12 SshGetUserAuthenticateWithType()	1482
14.44.3.13 SshGetUserGetKey()	1482
14.44.3.14 SshNegotiateSession()	1483
14.44.3.15 SshNegotiateSessionClient()	1483
14.44.3.16 SshPrintStatistics()	1483
14.44.3.17 SshSetBannerText()	1483
14.44.3.18 SshSetSockOption()	1484
14.44.3.19 SshSetUserAuthenticate()	1484
14.44.3.20 SshSetUserAuthenticateWithType()	1484
14.44.3.21 SshSetUserGetKey()	1485
14.44.3.22 SshValidateKey()	1485
14.44.3.23 SshWritePublicKey()	1485
14.44.4 SSH Error Codes	1486
14.44.4.1 Detailed Description	1486
14.45 SSL/TLS	1486
14.45.1 Detailed Description	1489
14.45.2 Enumeration Type Documentation	1489
14.45.2.1 ExpireRetCodes	1489
14.45.3 Function Documentation	1490
14.45.3.1 FreeSSLSockets()	1490

14.45.3.2 GetCertificateLen()	1490
14.45.3.3 GetCertificatePEM()	1490
14.45.3.4 GetPrivateKeyLen()	1490
14.45.3.5 GetPrivateKeyPEM()	1491
14.45.3.6 IsSSL_CertNKeyValid()	1491
14.45.3.7 SetMaxSessionReuseTime()	1491
14.45.3.8 SSL_accept()	1492
14.45.3.9 SSL_AddCertToClientCaList()	1493
14.45.3.10 SSL_AddCertToServerCaList()	1493
14.45.3.11 SSL_ClearClientCaList()	1493
14.45.3.12 SSL_ClearServerCaList()	1494
14.45.3.13 SSL_ClientReadyCert()	1494
14.45.3.14 SSL_connect() [1/2]	1494
14.45.3.15 SSL_connect() [2/2]	1495
14.45.3.16 SSL_GetAltNameFromCert()	1496
14.45.3.17 SSL_GetCnFromCert()	1496
14.45.3.18 SSL_GetExpirationDate()	1497
14.45.3.19 SSL_GetIssuerFromCert()	1497
14.45.3.20 SSL_getsocketerror()	1498
14.45.3.21 SSL_GetSocketInterface()	1498
14.45.3.22 SSL_GetSocketLocalAddr()	1498
14.45.3.23 SSL_GetSocketLocalPort()	1499
14.45.3.24 SSL_GetSocketRemoteAddr()	1499
14.45.3.25 SSL_GetSocketRemotePort()	1500
14.45.3.26 SSL_GetSubjectNameFromCert()	1500
14.45.3.27 SSL_GetTcpRtxCount()	1501
14.45.3.28 SSL_GetTimeToExpire()	1501
14.45.3.29 SSL_IsCertAddedToClientCaList()	1501
14.45.3.30 SSL_IsCertAddedToServerCaList()	1502
14.45.3.31 SSL_IsCertExpired()	1502
14.45.3.32 SSL_makeSecure()	1502
14.45.3.33 SSL_negotiate()	1503
14.45.3.34 SSL_ServerReadyCert()	1504
14.45.3.35 SSL_TcpGetLastRxInterval()	1504
14.45.3.36 SSL_TcpGetLastRxTime()	1505
14.45.3.37 SSL_TcpSendKeepAlive()	1505
14.45.3.38 SSLGetRxBufferSizeUsed()	1505
14.45.3.39 SSLGetTxBufferAvailSpace()	1505
14.45.3.40 SslInit()	1506
14.45.3.41 SSLReadWithTimeout()	1506
14.45.3.42 SslSetVerifyPeerClient()	1507
14.45.3.43 SslSetVerifyPeerServer()	1507

14.45.3.44 SSLSocketsAvail()	1507
14.45.4 Variable Documentation	1507
14.45.4.1 IsSSLfd	1507
14.45.4.2 SSL_clrsockoption	1508
14.45.4.3 SSL_GetRawSocket	1508
14.45.4.4 SSL_getsockoption	1508
14.45.4.5 SSL_setsockoption	1509
14.45.5 ACME Certificate Generation	1509
14.45.5.1 Detailed Description	1509
14.45.6 Onboard Self-Signed Certificate Generation	1510
14.45.6.1 Detailed Description	1511
14.45.6.2 Enumeration Type Documentation	1511
14.45.6.3 Function Documentation	1512
14.45.7 SSL Error Codes	1513
14.45.7.1 Detailed Description	1514
14.46 Serial Interfaces	1514
14.46.1 Detailed Description	1515
14.46.2 Macro Definition Documentation	1515
14.46.2.1 SimpleOpenSerial	1515
14.46.3 Enumeration Type Documentation	1515
14.46.3.1 parity_mode	1515
14.46.4 Function Documentation	1515
14.46.4.1 GetUartErrorReg()	1515
14.46.4.2 OpenDefaultSerial()	1516
14.46.4.3 OpenSerial()	1516
14.46.4.4 SendBreak()	1517
14.46.4.5 Serial485HalfDupMode()	1517
14.46.4.6 SerialClose()	1517
14.46.4.7 SerialEnableHwRxFlow()	1518
14.46.4.8 SerialEnableHwTxFlow()	1518
14.46.4.9 SerialEnableRxFlow()	1518
14.46.4.10 SerialEnableTxFlow()	1518
14.46.4.11 SerialExpandRxBuffer()	1519
14.46.4.12 SerialSendComplete()	1519
14.46.4.13 serwriteaddress()	1519
14.46.4.14 SetRTS()	1519
14.46.5 Serial Port Error Codes	1519
14.46.5.1 Detailed Description	1519
14.47 Shutdown Notifications	1520
14.47.1 Detailed Description	1520
14.47.2 Function Documentation	1520
14.47.2.1 NBApproveShutdown()	1520

14.47.2.2 NBFaultNotify()	1522
14.47.3 ShutdownReasons	1522
14.47.3.1 Detailed Description	1523
14.48 Signed Application Update	1523
14.48.1 Detailed Description	1523
14.48.2 Function Documentation	1523
14.48.2.1 ProgramApplication()	1523
14.48.2.2 RegisterAppSigningPublicKey()	1524
14.48.2.3 UpdateFromStream()	1524
14.48.3 NBUpdate Function Return Values	1524
14.48.3.1 Detailed Description	1525
14.49 Stream Update	1525
14.49.1 Detailed Description	1525
14.49.2 Function Documentation	1525
14.49.2.1 ReadBinaryApplicationCodeFromStream()	1525
14.49.2.2 ReadS19ApplicationCodeFromStream()	1526
14.49.3 Stream Update Return Values	1526
14.49.3.1 Detailed Description	1526
14.50 Synchronous Audio Interface (SOMRT1061)	1526
14.50.1 Detailed Description	1527
14.50.2 Variable Documentation	1527
14.50.2.1 sai	1527
14.50.3 SAI Callback Types	1527
14.50.3.1 Detailed Description	1527
14.50.3.2 Typedef Documentation	1527
14.50.4 SAI Configuration Enumerations	1528
14.50.4.1 Detailed Description	1529
14.50.4.2 Enumeration Type Documentation	1529
14.50.5 SAI Configuration Structures	1532
14.50.5.1 Detailed Description	1532
14.50.6 SAI Driver Class	1532
14.50.6.1 Detailed Description	1532
14.50.7 SAI Driver Constants	1532
14.50.7.1 Detailed Description	1533
14.50.7.2 Macro Definition Documentation	1533
14.50.8 SAI Predefined Configurations	1533
14.50.8.1 Detailed Description	1534
14.50.8.2 Variable Documentation	1534
14.51 Syslog	1535
14.51.1 Detailed Description	1536
14.51.2 Function Documentation	1536
14.51.2.1 SysLog()	1536

14.51.2.2 SysLogVia()	1538
14.51.3 Variable Documentation	1540
14.51.3.1 SysLogAddress	1540
14.52 System Functions	1540
14.52.1 Detailed Description	1541
14.52.2 Function Documentation	1541
14.52.2.1 GetReleaseTag()	1541
14.52.2.2 GetUserParameters()	1541
14.52.2.3 SaveUserParameters()	1541
14.53 TCP	1542
14.53.1 Detailed Description	1544
14.53.2 Overview	1544
14.53.3 Basic Usage Patterns	1544
14.53.3.1 Server Pattern (Listen/Accept)	1544
14.53.3.2 Client Pattern (Connect)	1545
14.53.3.3 Non-Blocking Pattern	1545
14.53.4 Important Concepts	1545
14.53.4.1 Entry Point	1545
14.53.4.2 Timeout Values	1545
14.53.4.3 File Descriptors	1545
14.53.4.4 Buffer Management	1545
14.53.4.5 Socket States	1546
14.53.4.6 Resource Management	1546
14.53.5 Function Categories	1546
14.53.5.1 Connection Management	1546
14.53.5.2 Data Transfer	1546
14.53.5.3 Socket Query Functions	1546
14.53.5.4 Configuration Functions	1547
14.53.5.5 Monitoring and Diagnostics	1547
14.53.5.6 Event Notification	1547
14.53.5.7 Throughput Monitoring (#define TCP_THROUGHPUT_INFO)	1547
14.53.6 Error Handling	1547
14.53.7 Advanced Features	1548
14.53.7.1 Multi-Interface Control	1548
14.53.7.2 Keepalive Implementation	1548
14.53.7.3 Nagle Algorithm Control	1548
14.53.7.4 Event-Driven Programming	1548
14.53.8 Best Practices	1548
14.53.8.1 DO:	1548
14.53.8.2 DON'T:	1549
14.53.9 IPv6 Support	1549
14.53.10 Performance Tuning	1549

14.53.10.1 Bulk Data Transfer	1549
14.53.10.2 Low Latency	1549
14.53.10.3 Many Simultaneous Connections	1549
14.53.11 Thread Safety	1550
14.53.12 Related APIs	1550
14.53.13 Further Reading	1550
14.53.14 Function Documentation	1550
14.53.14.1 abortsocket()	1550
14.53.14.2 accept() [1/2]	1552
14.53.14.3 accept() [2/2]	1557
14.53.14.4 clrsockoption()	1560
14.53.14.5 connect() [1/2]	1563
14.53.14.6 connect() [2/2]	1570
14.53.14.7 connectvia() [1/4]	1577
14.53.14.8 connectvia() [2/4]	1585
14.53.14.9 connectvia() [3/4]	1594
14.53.14.10 connectvia() [4/4]	1600
14.53.14.11 connectwlocal() [1/2]	1608
14.53.14.12 connectwlocal() [2/2]	1616
14.53.14.13 DumpTcpDebug()	1623
14.53.14.14 EnableTcpDebug()	1624
14.53.14.15 getsocketerror()	1626
14.53.14.16 GetSocketLocalAddr()	1631
14.53.14.17 GetSocketLocalPort()	1632
14.53.14.18 GetSocketRemoteAddr()	1632
14.53.14.19 GetSocketRemotePort()	1633
14.53.14.20 getsockoption()	1634
14.53.14.21 GetTcpRtxCount()	1638
14.53.14.22 listen()	1643
14.53.14.23 listenvia() [1/2]	1650
14.53.14.24 listenvia() [2/2]	1653
14.53.14.25 NoBlockConnect()	1661
14.53.14.26 NoBlockConnectVia() [1/2]	1665
14.53.14.27 NoBlockConnectVia() [2/2]	1665
14.53.14.28 NoBlockConnectwlocal()	1670
14.53.14.29 SetOutOfOrderBuffers()	1676
14.53.14.30 SetSocketRxBuffers()	1681
14.53.14.31 SetSocketTxBuffers()	1682
14.53.14.32 SetSocketUnackBuffers()	1683
14.53.14.33 setsockoption()	1684
14.53.14.34 SocketPeek()	1688
14.53.14.35 SockReadWithTimeout()	1689

14.53.14.36	TcpAllDataAcked()	1691
14.53.14.37	TcpGetLastRxInterval()	1692
14.53.14.38	TcpGetLastRxTime()	1696
14.53.14.39	TcpGetRxBufferSpaceUsed()	1700
14.53.14.40	TcpGetSocketInterface()	1700
14.53.14.41	TcpGetSocketState()	1703
14.53.14.42	TcpGetTxBufferAvailSpace()	1703
14.53.14.43	TcpGetTxDataWaiting()	1703
14.53.14.44	TcpSendKeepAlive()	1704
14.53.14.45	WaitForSocketFlush()	1706
14.53.15	TCP Notification System	1707
14.53.15.1	Detailed Description	1707
14.53.15.2	Typedef Documentation	1707
14.53.15.3	Function Documentation	1709
14.53.16	TCP Socket Options	1713
14.53.16.1	Detailed Description	1714
14.53.17	TCP Socket State	1714
14.53.17.1	Detailed Description	1714
14.53.18	TCP Socket Status	1715
14.53.18.1	Detailed Description	1715
14.54	TFTP	1715
14.54.1	Detailed Description	1715
14.54.2	TFTP Client	1715
14.54.2.1	Detailed Description	1716
14.54.2.2	Function Documentation	1716
14.54.2.3	TFTP Return Codes	1717
14.54.3	TFTP Server	1717
14.54.3.1	Detailed Description	1717
14.55	Time	1718
14.55.1	Detailed Description	1718
14.55.2	Function Documentation	1719
14.55.2.1	set_time()	1719
14.55.2.2	time()	1719
14.55.2.3	timegm()	1720
14.55.2.4	tzsetchar()	1720
14.55.3	Variable Documentation	1720
14.55.3.1	TZRecords	1720
14.56	Timers	1722
14.56.1	Detailed Description	1722
14.56.2	High Resolution Delay Timer	1722
14.56.2.1	Detailed Description	1722
14.56.2.2	Usage Overview	1722

14.56.2.3 Timing Precision	1723
14.56.2.4 Example Usage	1723
14.56.2.5 Important Notes	1723
14.56.2.6 Comparison with Other Timing Methods	1723
14.56.3 Interval Timer	1723
14.56.3.1 Detailed Description	1724
14.56.3.2 Function Documentation	1727
14.56.4 Stopwatch Timer	1739
14.56.4.1 Detailed Description	1739
14.57 UDP	1740
14.57.1 Detailed Description	1741
14.57.1.1 Features	1741
14.57.1.2 When to Use UDP	1741
14.57.1.3 API Selection Guide	1742
14.57.1.4 Quick Start Examples	1742
14.57.1.5 Memory Management	1743
14.57.1.6 Port Number Guidelines	1744
14.57.1.7 Quality of Service (QoS)	1744
14.57.1.8 Performance Considerations	1745
14.57.1.9 Error Handling	1745
14.57.1.10 Thread Safety	1746
14.57.1.11 Advanced Features	1746
14.57.1.12 Debugging Tips	1746
14.57.1.13 Common Pitfalls	1746
14.57.2 Typedef Documentation	1747
14.57.2.1 udp_data_notify	1747
14.57.3 Function Documentation	1748
14.57.3.1 RegisterEphemeralFifo()	1748
14.57.3.2 RegisterUDPFifo()	1750
14.57.3.3 RegisterUDPFifoVia()	1751
14.57.3.4 RegisterUDPFifoWithNotify()	1754
14.57.3.5 RegisterUDPFifoWithNotifyVia()	1755
14.57.3.6 UnregisterUDPFifo()	1757
14.57.4 UDP Error Codes	1758
14.57.4.1 Detailed Description	1758
14.57.5 UDP Packet API	1759
14.57.6 UDP Socket API	1759
14.57.6.1 Detailed Description	1759
14.57.6.2 Send UDP packet from command line	1766
14.57.6.3 Listen for UDP packets	1766
14.57.6.4 Function Documentation	1766
14.58 User Authorization Manager	1777

14.58.1 Detailed Description	1777
14.58.2 Typedef Documentation	1777
14.58.2.1 LoadAuthRecordsFn	1777
14.58.2.2 SaveAuthRecordsFn	1777
14.58.3 Authorization Responses	1778
14.58.3.1 Detailed Description	1778
14.58.3.2 Enumeration Type Documentation	1778
14.58.4 Authorization Types	1778
14.58.4.1 Detailed Description	1779
14.58.4.2 Enumeration Type Documentation	1779
14.59 Utilities	1779
14.59.1 Detailed Description	1781
14.59.2 Macro Definition Documentation	1781
14.59.2.1 DB_TCPIP	1781
14.59.3 Function Documentation	1782
14.59.3.1 _exit()	1782
14.59.3.2 _fini()	1782
14.59.3.3 AsciiToIp4()	1782
14.59.3.4 AsciiToIp6()	1782
14.59.3.5 AsciiToMac()	1782
14.59.3.6 bufnstr()	1783
14.59.3.7 convertBinaryToHexAscii()	1783
14.59.3.8 convertHexAsciiToBinary()	1787
14.59.3.9 FdShowData()	1792
14.59.3.10 fdShowMac()	1792
14.59.3.11 FdShowRingData()	1792
14.59.3.12 Get_msec()	1792
14.59.3.13 GetPreciseTime()	1793
14.59.3.14 kill()	1793
14.59.3.15 MacToId()	1793
14.59.3.16 outbyte()	1793
14.59.3.17 print()	1794
14.59.3.18 putbytenum()	1794
14.59.3.19 putnum()	1794
14.59.3.20 ShowData()	1794
14.59.3.21 ShowIP4()	1794
14.59.3.22 ShowIP6()	1795
14.59.3.23 ShowMac() [1/2]	1795
14.59.3.24 ShowMac() [2/2]	1795
14.59.3.25 ShowRingData()	1795
14.59.3.26 snShowIP4()	1796
14.59.3.27 snShowIP6()	1796

14.59.3.28 snShowMac() [1/2]	1796
14.59.3.29 snShowMac() [2/2]	1797
14.59.3.30 sShowCounters()	1797
14.59.3.31 ValidIPv4()	1797
14.60 Web Client	1797
14.60.1 Detailed Description	1805
14.60.2 Function Documentation	1806
14.60.2.1 DoActualClientRequest()	1806
14.60.2.2 DoGet() [1/6]	1806
14.60.2.3 DoGet() [2/6]	1807
14.60.2.4 DoGet() [3/6]	1808
14.60.2.5 DoGet() [4/6]	1808
14.60.2.6 DoGet() [5/6]	1809
14.60.2.7 DoGet() [6/6]	1810
14.60.2.8 DoGetEx() [1/6]	1811
14.60.2.9 DoGetEx() [2/6]	1811
14.60.2.10 DoGetEx() [3/6]	1813
14.60.2.11 DoGetEx() [4/6]	1814
14.60.2.12 DoGetEx() [5/6]	1815
14.60.2.13 DoGetEx() [6/6]	1816
14.60.2.14 DoGetUpdate() [1/2]	1816
14.60.2.15 DoGetUpdate() [2/2]	1817
14.60.2.16 DoHttp() [1/6]	1817
14.60.2.17 DoHttp() [2/6]	1818
14.60.2.18 DoHttp() [3/6]	1819
14.60.2.19 DoHttp() [4/6]	1819
14.60.2.20 DoHttp() [5/6]	1820
14.60.2.21 DoHttp() [6/6]	1821
14.60.2.22 DoHttpEx() [1/6]	1822
14.60.2.23 DoHttpEx() [2/6]	1822
14.60.2.24 DoHttpEx() [3/6]	1823
14.60.2.25 DoHttpEx() [4/6]	1824
14.60.2.26 DoHttpEx() [5/6]	1825
14.60.2.27 DoHttpEx() [6/6]	1826
14.60.2.28 DoJsonHttp() [1/8]	1827
14.60.2.29 DoJsonHttp() [2/8]	1828
14.60.2.30 DoJsonHttp() [3/8]	1829
14.60.2.31 DoJsonHttp() [4/8]	1830
14.60.2.32 DoJsonHttp() [5/8]	1831
14.60.2.33 DoJsonHttp() [6/8]	1832
14.60.2.34 DoJsonHttp() [7/8]	1833
14.60.2.35 DoJsonHttp() [8/8]	1834

14.60.2.36 DoJsonHttpFile() [1/4]	1835
14.60.2.37 DoJsonHttpFile() [2/4]	1836
14.60.2.38 DoJsonHttpFile() [3/4]	1836
14.60.2.39 DoJsonHttpFile() [4/4]	1838
14.60.2.40 DoJsonPost() [1/8]	1839
14.60.2.41 DoJsonPost() [2/8]	1840
14.60.2.42 DoJsonPost() [3/8]	1841
14.60.2.43 DoJsonPost() [4/8]	1841
14.60.2.44 DoJsonPost() [5/8]	1842
14.60.2.45 DoJsonPost() [6/8]	1843
14.60.2.46 DoJsonPost() [7/8]	1844
14.60.2.47 DoJsonPost() [8/8]	1845
14.60.2.48 DoJsonPostHttpFile() [1/4]	1846
14.60.2.49 DoJsonPostHttpFile() [2/4]	1847
14.60.2.50 DoJsonPostHttpFile() [3/4]	1848
14.60.2.51 DoJsonPostHttpFile() [4/4]	1848
14.60.2.52 DoMultipartBoundary()	1849
14.60.2.53 DoMultipartFinished() [1/2]	1850
14.60.2.54 DoMultipartFinished() [2/2]	1851
14.60.2.55 DoMultipartItem()	1851
14.60.2.56 DoMultipartStartHttp() [1/2]	1852
14.60.2.57 DoMultipartStartHttp() [2/2]	1853
14.60.2.58 DoMultipartStartPost() [1/2]	1854
14.60.2.59 DoMultipartStartPost() [2/2]	1855
14.60.2.60 DoUrlEncodedFormHttp() [1/3]	1856
14.60.2.61 DoUrlEncodedFormHttp() [2/3]	1857
14.60.2.62 DoUrlEncodedFormHttp() [3/3]	1858
14.60.2.63 DoUrlEncodedFormPost() [1/4]	1859
14.60.2.64 DoUrlEncodedFormPost() [2/4]	1860
14.60.2.65 DoUrlEncodedFormPost() [3/4]	1860
14.60.2.66 DoUrlEncodedFormPost() [4/4]	1861
14.60.2.67 PopulateAuthHeader()	1862
14.60.2.68 SetHttpDiag() [1/2]	1863
14.60.2.69 SetHttpDiag() [2/2]	1863
14.60.2.70 StartConnection()	1863
14.60.2.71 StartTransaction() [1/2]	1864
14.60.2.72 StartTransaction() [2/2]	1865
14.60.2.73 StartWebClient() [1/3]	1865
14.60.2.74 StartWebClient() [2/3]	1866
14.60.2.75 StartWebClient() [3/3]	1866
14.60.2.76 WriteHttpRequestHeaders()	1867
14.60.3 Web Client Error Codes	1867

14.60.3.1 Detailed Description	1868
14.61 Web Sockets	1868
14.61.1 Detailed Description	1868
14.61.2 Function Documentation	1869
14.61.2.1 WSGetPingReplyTick()	1869
14.61.2.2 WSPing()	1872
14.61.3 WSPing	1872
14.61.3.1 Description	1872
14.61.3.2 Parameters	1872
14.61.3.3 Return Value	1872
14.61.3.4 Usage Notes	1872
14.61.3.5 See Also	1874
14.61.3.6 WSUpgrade()	1874
14.61.3.7 WSWaitForPingReply()	1879
14.61.4 Web Socket Definitions	1882
14.61.4.1 Detailed Description	1883
14.61.4.2 Overview	1883
14.62 Wifi	1883
14.62.1 Detailed Description	1884
14.62.2 Function Documentation	1884
14.62.2.1 InitAP_SPI()	1884
14.62.2.2 InitWifi_Serial()	1885
14.62.2.3 InitWifi_SPI()	1886
14.62.2.4 ScanAndShowNetworks()	1886
14.62.2.5 ScanForNetworks()	1886
14.62.2.6 SetWifiSPISpeed()	1886
14.62.2.7 WifiInitScan_Serial()	1887
14.62.2.8 WifiInitScan_SPI()	1887
14.62.2.9 WifiInitScanAndShow_Serial()	1888
14.62.2.10 WifiInitScanAndShow_SPI()	1888
14.62.3 BSS Options	1888
14.62.3.1 Detailed Description	1889
14.62.4 Cipher Options	1889
14.62.4.1 Detailed Description	1889
14.62.5 Configuration Errors	1889
14.62.5.1 Detailed Description	1890
14.62.6 Connect Request Errors	1890
14.62.6.1 Detailed Description	1890
14.62.7 Save Config Record Errors	1890
14.62.7.1 Detailed Description	1890
14.62.8 Scan Request Errors	1890
14.62.8.1 Detailed Description	1891

14.62.9 Security Options	1891
14.62.9.1 Detailed Description	1891
15 Namespace Documentation	1893
15.1 canMCF5441x Namespace Reference	1893
15.1.1 Detailed Description	1893
15.2 DHCP Namespace Reference	1893
15.2.1 Detailed Description	1894
15.2.2 Enumeration Type Documentation	1894
15.2.2.1 LeaseState	1894
15.3 mcanMODM7AE70 Namespace Reference	1894
15.3.1 Detailed Description	1895
15.4 MQTT Namespace Reference	1895
15.4.1 Detailed Description	1897
15.4.2 Typedef Documentation	1897
15.4.2.1 MsgCallback_t	1897
15.4.3 Enumeration Type Documentation	1898
15.4.3.1 anonymous enum	1898
15.4.3.2 eObj_Serialize_t	1898
15.4.3.3 eResult_t	1899
15.4.4 Function Documentation	1900
15.4.4.1 GetConnectionStatusString()	1900
15.4.4.2 GetPacketTypeString()	1900
15.4.4.3 GetReturnCodeString()	1901
15.5 MQTT::PacketInfo Namespace Reference	1901
15.5.1 Detailed Description	1901
15.6 NB::Error Namespace Reference	1901
15.6.1 Detailed Description	1901
15.6.2 Enumeration Type Documentation	1901
15.6.2.1 GeneralErrors	1901
15.6.3 GeneralErrors	1901
15.7 NB::Error::Connect Namespace Reference	1902
15.7.1 Detailed Description	1902
15.7.2 Enumeration Type Documentation	1902
15.7.2.1 ConnectErrors	1902
15.7.3 ConnectErrors	1902
15.8 NB::Error::Init Namespace Reference	1902
15.8.1 Detailed Description	1903
15.8.2 Enumeration Type Documentation	1903
15.8.2.1 InitializationErrors	1903
15.8.3 InitializationErrors	1903
15.9 NB::Error::Scan Namespace Reference	1903

15.9.1 Detailed Description	1903
15.9.2 Enumeration Type Documentation	1903
15.9.2.1 ScanErrors	1903
15.9.3 ScanErrors	1903
15.10 ReceiveUDP Namespace Reference	1903
15.10.1 Detailed Description	1904
15.10.2 Function Documentation	1904
15.10.2.1 send_test_packet()	1904
15.10.2.2 signal_handler()	1904
16 Class Documentation	1905
16.1 _edma_channel_Preemption_config Struct Reference	1905
16.1.1 Detailed Description	1905
16.1.2 Member Data Documentation	1905
16.1.2.1 channelPriority	1905
16.1.2.2 enableChannelPreemption	1905
16.1.2.3 enablePreemptAbility	1905
16.2 _edma_config Struct Reference	1905
16.2.1 Detailed Description	1906
16.2.2 Member Data Documentation	1906
16.2.2.1 enableContinuousLinkMode	1906
16.2.2.2 enableDebugMode	1906
16.2.2.3 enableHaltOnError	1906
16.2.2.4 enableRoundRobinArbitration	1906
16.3 _edma_handle Struct Reference	1906
16.3.1 Detailed Description	1906
16.3.2 Member Data Documentation	1907
16.3.2.1 base	1907
16.3.2.2 callback	1907
16.3.2.3 channel	1907
16.3.2.4 flags	1907
16.3.2.5 header	1907
16.3.2.6 tail	1907
16.3.2.7 tcdPool	1907
16.3.2.8 tcdSize	1907
16.3.2.9 tcdUsed	1907
16.3.2.10 userData	1907
16.4 _edma_minor_offset_config Struct Reference	1907
16.4.1 Detailed Description	1908
16.4.2 Member Data Documentation	1908
16.4.2.1 enableDestMinorOffset	1908
16.4.2.2 enableSrcMinorOffset	1908

16.4.2.3 minorOffset	1908
16.5 _edma_tcd Struct Reference	1908
16.5.1 Detailed Description	1908
16.5.2 Member Data Documentation	1909
16.5.2.1 ATTR	1909
16.5.2.2 BITER	1909
16.5.2.3 CITER	1909
16.5.2.4 CSR	1909
16.5.2.5 DADDR	1909
16.5.2.6 DLAST_SGA	1909
16.5.2.7 DOFF	1909
16.5.2.8 NBYTES	1909
16.5.2.9 SADDR	1909
16.5.2.10 SLAST	1909
16.5.2.11 SOFF	1909
16.6 _edma_transfer_config Struct Reference	1910
16.6.1 Detailed Description	1910
16.6.2 Member Data Documentation	1910
16.6.2.1 destAddr	1910
16.6.2.2 destOffset	1910
16.6.2.3 destTransferSize	1910
16.6.2.4 majorLoopCounts	1910
16.6.2.5 minorLoopBytes	1910
16.6.2.6 srcAddr	1910
16.6.2.7 srcOffset	1910
16.6.2.8 srcTransferSize	1911
16.7 _EthernetFrameHeader Struct Reference	1911
16.7.1 Detailed Description	1911
16.8 _flexcan_config Struct Reference	1911
16.8.1 Detailed Description	1911
16.8.2 Member Data Documentation	1912
16.8.2.1 baudRate	1912
16.8.2.2 bitRate	1912
16.8.2.3 clkSrc	1912
16.8.2.4 disableSelfReception	1912
16.8.2.5 enableIndividMask	1912
16.8.2.6 enableListenOnlyMode	1912
16.8.2.7 enableLoopBack	1912
16.8.2.8 enableSelfWakeup	1912
16.8.2.9 enableSupervisorMode	1912
16.8.2.10 enableTimerSync	1912
16.8.2.11 maxMbNum	1912

16.8.2.12 wakeupSrc	1913
16.9 _flexcan_fifo_transfer Struct Reference	1913
16.9.1 Detailed Description	1913
16.9.2 Member Data Documentation	1913
16.9.2.1 frame	1913
16.9.2.2 frameNum	1913
16.10 _flexcan_frame Struct Reference	1913
16.10.1 Detailed Description	1913
16.10.2 Member Data Documentation	1913
16.10.2.1 __pad0__	1913
16.10.2.2 dataByte0	1913
16.10.2.3 dataByte1	1914
16.10.2.4 dataByte2	1914
16.10.2.5 dataByte3	1914
16.10.2.6 dataByte4	1914
16.10.2.7 dataByte5	1914
16.10.2.8 dataByte6	1914
16.10.2.9 dataByte7	1914
16.10.2.10 dataWord0	1914
16.10.2.11 dataWord1	1914
16.10.2.12 format	1914
16.10.2.13 id	1914
16.10.2.14 idhit	1914
16.10.2.15 length	1915
16.10.2.16 timestamp	1915
16.10.2.17 type	1915
16.11 _flexcan_handle Struct Reference	1915
16.11.1 Detailed Description	1915
16.11.2 Member Data Documentation	1915
16.11.2.1 callback	1915
16.11.2.2 mbFrameBuf	1915
16.11.2.3 mbState	1915
16.11.2.4 rxFifoFrameBuf	1915
16.11.2.5 rxFifoFrameNum	1916
16.11.2.6 rxFifoState	1916
16.11.2.7 rxFifoTransferTotalNum	1916
16.11.2.8 timestamp	1916
16.11.2.9 userData	1916
16.12 _flexcan_mb_transfer Struct Reference	1916
16.12.1 Detailed Description	1916
16.12.2 Member Data Documentation	1916
16.12.2.1 frame	1916

16.12.2.2 mbldx	1916
16.13 _flexcan_rx_fifo_config Struct Reference	1917
16.13.1 Detailed Description	1917
16.13.2 Member Data Documentation	1917
16.13.2.1 idFilterNum	1917
16.13.2.2 idFilterTable	1917
16.13.2.3 idFilterType	1917
16.13.2.4 priority	1917
16.14 _flexcan_rx_mb_config Struct Reference	1917
16.14.1 Detailed Description	1917
16.14.2 Member Data Documentation	1918
16.14.2.1 format	1918
16.14.2.2 id	1918
16.14.2.3 type	1918
16.15 _flexcan_timing_config Struct Reference	1918
16.15.1 Detailed Description	1918
16.15.2 Member Data Documentation	1918
16.15.2.1 phaseSeg1	1918
16.15.2.2 phaseSeg2	1918
16.15.2.3 preDivider	1918
16.15.2.4 propSeg	1918
16.15.2.5 rJumpwidth	1919
16.16 _FlexSPICongig Struct Reference	1919
16.16.1 Detailed Description	1920
16.16.2 Member Data Documentation	1920
16.16.2.1 busyBitPolarity	1920
16.16.2.2 columnAddressWidth	1920
16.16.2.3 controllerMiscOption	1921
16.16.2.4 deviceModeArg	1921
16.16.2.5 deviceModeCfgEnable	1921
16.16.2.6 deviceModeSeq	1921
16.16.2.7 deviceModeType	1921
16.16.2.8 deviceType	1921
16.16.2.9 lookupTable	1921
16.16.2.10 lutCustomSeqEnable	1921
16.16.2.11 reserved3	1921
16.16.2.12 serialClkFreq	1921
16.16.2.13 waitTimeCfgCommands	1922
16.17 _IpHeaderIPv4 Struct Reference	1922
16.17.1 Detailed Description	1922
16.18 _lpspi_master_config Struct Reference	1923
16.18.1 Detailed Description	1923

16.18.2 Member Data Documentation	1923
16.18.2.1 baudRate	1923
16.18.2.2 betweenTransferDelayInNanoSec	1923
16.18.2.3 bitsPerFrame	1923
16.18.2.4 cpha	1923
16.18.2.5 cpol	1923
16.18.2.6 dataOutConfig	1923
16.18.2.7 direction	1924
16.18.2.8 enableInputDelay	1924
16.18.2.9 lastSckToPcsDelayInNanoSec	1924
16.18.2.10 pcsActiveHighOrLow	1924
16.18.2.11 pcsToSckDelayInNanoSec	1924
16.18.2.12 pinCfg	1924
16.18.2.13 whichPcs	1924
16.19 _lpspi_master_edma_handle Struct Reference	1924
16.19.1 Detailed Description	1925
16.19.2 Member Data Documentation	1925
16.19.2.1 bytesEachRead	1925
16.19.2.2 bytesEachWrite	1925
16.19.2.3 bytesLastRead	1925
16.19.2.4 callback	1925
16.19.2.5 edmaRxRegToRxDataHandle	1925
16.19.2.6 edmaTxDataToTxRegHandle	1925
16.19.2.7 fifoSize	1925
16.19.2.8 isByteSwap	1925
16.19.2.9 isPcsContinuous	1926
16.19.2.10 isThereExtraRxBytes	1926
16.19.2.11 lpspiSoftwareTCD	1926
16.19.2.12 nbytes	1926
16.19.2.13 readRegRemainingTimes	1926
16.19.2.14 rxBuffIfNull	1926
16.19.2.15 rxData	1926
16.19.2.16 rxRemainingByteCount	1926
16.19.2.17 rxWatermark	1926
16.19.2.18 state	1926
16.19.2.19 totalByteCount	1926
16.19.2.20 transmitCommand	1926
16.19.2.21 txBuffIfNull	1927
16.19.2.22 txData	1927
16.19.2.23 txRemainingByteCount	1927
16.19.2.24 userData	1927
16.19.2.25 writeRegRemainingTimes	1927

16.20 _lpspi_master_edma_private_handle Struct Reference	1927
16.20.1 Detailed Description	1927
16.20.2 Member Data Documentation	1927
16.20.2.1 base	1927
16.20.2.2 handle	1927
16.21 _lpspi_master_handle Struct Reference	1928
16.21.1 Detailed Description	1928
16.21.2 Member Data Documentation	1928
16.21.2.1 bytesEachRead	1928
16.21.2.2 bytesEachWrite	1928
16.21.2.3 bytesPerFrame	1928
16.21.2.4 callback	1928
16.21.2.5 fifoSize	1928
16.21.2.6 isByteSwap	1929
16.21.2.7 isPcsContinuous	1929
16.21.2.8 isTxMask	1929
16.21.2.9 readRegRemainingTimes	1929
16.21.2.10 rxData	1929
16.21.2.11 rxRemainingByteCount	1929
16.21.2.12 rxWatermark	1929
16.21.2.13 state	1929
16.21.2.14 totalByteCount	1929
16.21.2.15 txBuffIfNull	1929
16.21.2.16 txData	1929
16.21.2.17 txRemainingByteCount	1929
16.21.2.18 userData	1930
16.21.2.19 writeRegRemainingTimes	1930
16.21.2.20 writeTcrInlSr	1930
16.22 _lpspi_slave_config Struct Reference	1930
16.22.1 Detailed Description	1930
16.22.2 Member Data Documentation	1930
16.22.2.1 bitsPerFrame	1930
16.22.2.2 cpha	1930
16.22.2.3 cpol	1930
16.22.2.4 dataOutConfig	1930
16.22.2.5 direction	1931
16.22.2.6 pcsActiveHighOrLow	1931
16.22.2.7 pinCfg	1931
16.22.2.8 whichPcs	1931
16.23 _lpspi_slave_edma_handle Struct Reference	1931
16.23.1 Detailed Description	1931
16.23.2 Member Data Documentation	1932

16.23.2.1 bytesEachRead	1932
16.23.2.2 bytesEachWrite	1932
16.23.2.3 bytesLastRead	1932
16.23.2.4 callback	1932
16.23.2.5 edmaRxRegToRxDataHandle	1932
16.23.2.6 edmaTxDataToTxRegHandle	1932
16.23.2.7 errorCount	1932
16.23.2.8 fifoSize	1932
16.23.2.9 isByteSwap	1932
16.23.2.10 isThereExtraRxBytes	1932
16.23.2.11 lpspiSoftwareTCD	1932
16.23.2.12 nbytes	1932
16.23.2.13 readRegRemainingTimes	1933
16.23.2.14 rxBuffIfNull	1933
16.23.2.15 rxData	1933
16.23.2.16 rxRemainingByteCount	1933
16.23.2.17 rxWatermark	1933
16.23.2.18 state	1933
16.23.2.19 totalByteCount	1933
16.23.2.20 txBuffIfNull	1933
16.23.2.21 txData	1933
16.23.2.22 txRemainingByteCount	1933
16.23.2.23 userData	1933
16.23.2.24 writeRegRemainingTimes	1933
16.24 _lpspi_slave_handle Struct Reference	1934
16.24.1 Detailed Description	1934
16.24.2 Member Data Documentation	1934
16.24.2.1 bytesEachRead	1934
16.24.2.2 bytesEachWrite	1934
16.24.2.3 callback	1934
16.24.2.4 errorCount	1934
16.24.2.5 fifoSize	1934
16.24.2.6 isByteSwap	1934
16.24.2.7 readRegRemainingTimes	1935
16.24.2.8 rxData	1935
16.24.2.9 rxRemainingByteCount	1935
16.24.2.10 rxWatermark	1935
16.24.2.11 state	1935
16.24.2.12 totalByteCount	1935
16.24.2.13 txData	1935
16.24.2.14 txRemainingByteCount	1935
16.24.2.15 userData	1935

16.24.2.16 writeRegRemainingTimes	1935
16.25 _lpspi_transfer Struct Reference	1935
16.25.1 Detailed Description	1936
16.25.2 Member Data Documentation	1936
16.25.2.1 configFlags	1936
16.25.2.2 dataSize	1936
16.25.2.3 rxData	1936
16.25.2.4 txData	1936
16.26 _lut_sequence Struct Reference	1936
16.26.1 Detailed Description	1936
16.27 _PinVector Class Reference	1937
16.27.1 Detailed Description	1937
16.27.2 Member Function Documentation	1937
16.27.2.1 config() [1/2]	1937
16.27.2.2 config() [2/2]	1937
16.27.2.3 operator uint32_t()	1938
16.27.2.4 operator=()	1938
16.27.2.5 operator[]()	1938
16.28 _pit_config Struct Reference	1938
16.28.1 Detailed Description	1939
16.28.2 Member Data Documentation	1939
16.28.2.1 enableRunInDebug	1939
16.29 _qtmr_config Struct Reference	1939
16.29.1 Detailed Description	1939
16.29.2 Member Data Documentation	1939
16.29.2.1 debugMode	1939
16.29.2.2 enableExternalForce	1939
16.29.2.3 enableMasterMode	1939
16.29.2.4 faultFilterCount	1939
16.29.2.5 faultFilterPeriod	1940
16.29.2.6 primarySource	1940
16.29.2.7 secondarySource	1940
16.30 _tempmon_config Struct Reference	1940
16.30.1 Detailed Description	1940
16.30.2 Member Data Documentation	1940
16.30.2.1 frequency	1940
16.30.2.2 highAlarmTemp	1940
16.30.2.3 lowAlarmTemp	1940
16.30.2.4 panicAlarmTemp	1940
16.31 AcmeServletObject Class Reference	1941
16.31.1 Detailed Description	1941
16.31.2 Constructor & Destructor Documentation	1941

16.31.2.1 AcmeServletObject()	1941
16.31.3 Member Function Documentation	1941
16.31.3.1 Delete_Everything_Restart()	1941
16.31.3.2 GetStateCC()	1941
16.31.4 Friends And Related Symbol Documentation	1942
16.31.4.1 GetDataForCertGen	1942
16.32 aes_context Struct Reference	1942
16.32.1 Detailed Description	1942
16.32.2 Member Data Documentation	1942
16.32.2.1 drk	1942
16.32.2.2 erk	1942
16.32.2.3 nr	1942
16.33 ARM_MPU_Region_t Struct Reference	1942
16.33.1 Detailed Description	1943
16.34 ARP Struct Reference	1943
16.34.1 Detailed Description	1943
16.35 BBI2C Class Reference	1943
16.35.1 Detailed Description	1944
16.35.2 Member Enumeration Documentation	1944
16.35.2.1 Result_t	1944
16.35.3 Constructor & Destructor Documentation	1945
16.35.3.1 BBI2C()	1945
16.35.4 Member Function Documentation	1945
16.35.4.1 readReg8()	1945
16.35.4.2 readRegN()	1945
16.35.4.3 SetMaxBusDelay()	1946
16.35.4.4 setNumAddressBytes()	1947
16.35.4.5 setup()	1947
16.35.4.6 writeReg8()	1947
16.35.4.7 writeRegN()	1947
16.36 DHCP::BlockAllocator Class Reference	1948
16.36.1 Detailed Description	1948
16.37 buffer_list Class Reference	1948
16.37.1 Detailed Description	1949
16.37.2 Member Function Documentation	1949
16.37.2.1 GetCount()	1949
16.37.2.2 InsertAfter()	1949
16.37.2.3 InsertBefore()	1949
16.37.2.4 InsertHead()	1949
16.37.2.5 InsertTail()	1950
16.37.2.6 Remove()	1950
16.37.2.7 RemoveHead()	1950

16.37.2.8 RemoveTail()	1950
16.38 buffer_object Class Reference	1950
16.38.1 Detailed Description	1950
16.39 BuyPassAcmeServletObject Class Reference	1951
16.39.1 Detailed Description	1951
16.40 CallBackFunctionPageHandler Class Reference	1951
16.40.1 Detailed Description	1952
16.40.2 Constructor & Destructor Documentation	1952
16.40.2.1 CallBackFunctionPageHandler() [1/2]	1952
16.40.2.2 CallBackFunctionPageHandler() [2/2]	1953
16.40.3 Member Function Documentation	1953
16.40.3.1 ProcessRaw()	1953
16.41 CallBackFunctionPostHandler Class Reference	1954
16.41.1 Detailed Description	1954
16.41.2 Member Function Documentation	1954
16.41.2.1 ProcessRaw()	1954
16.42 canMCF5441x::CanRxMessage Class Reference	1954
16.42.1 Detailed Description	1955
16.42.2 Constructor & Destructor Documentation	1955
16.42.2.1 CanRxMessage() [1/2]	1955
16.42.2.2 CanRxMessage() [2/2]	1955
16.42.3 Member Function Documentation	1956
16.42.3.1 GetData()	1956
16.42.3.2 GetNewMessage()	1957
16.42.3.3 IsValid()	1957
16.43 mcanMODM7AE70::CanRxMessage Class Reference	1957
16.43.1 Detailed Description	1958
16.43.2 Constructor & Destructor Documentation	1958
16.43.2.1 CanRxMessage()	1958
16.43.3 Member Function Documentation	1958
16.43.3.1 CopyData()	1958
16.43.3.2 GetData()	1958
16.43.3.3 GetId()	1958
16.43.3.4 GetLength()	1959
16.43.3.5 GetNewMessage()	1959
16.43.3.6 GetTimeStamp()	1959
16.43.3.7 IsValid()	1959
16.44 CertData Class Reference	1960
16.44.1 Detailed Description	1960
16.45 CertGenData Struct Reference	1960
16.45.1 Detailed Description	1960
16.46 MQTT::Client Class Reference	1960

16.46.1 Detailed Description	1963
16.46.2 Member Enumeration Documentation	1963
16.46.2.1 eConnectionStatus	1963
16.46.2.2 eRateLimitAdjustMethod	1963
16.46.2.3 eRateLimitType	1963
16.46.2.4 eSessionLimit	1963
16.46.3 Constructor & Destructor Documentation	1964
16.46.3.1 Client() [1/2]	1964
16.46.3.2 Client() [2/2]	1964
16.46.4 Member Function Documentation	1965
16.46.4.1 Connect() [1/2]	1965
16.46.4.2 Connect() [2/2]	1965
16.46.4.3 Disconnect()	1967
16.46.4.4 GetClientCert()	1967
16.46.4.5 GetConnectionStatus()	1967
16.46.4.6 GetFirstClient()	1967
16.46.4.7 Publish() [1/3]	1968
16.46.4.8 Publish() [2/3]	1968
16.46.4.9 Publish() [3/3]	1969
16.46.4.10 QueueAllSubscriptions()	1969
16.46.4.11 QueuePublish() [1/2]	1969
16.46.4.12 QueuePublish() [2/2]	1969
16.46.4.13 QueueSubscribe() [1/2]	1970
16.46.4.14 QueueSubscribe() [2/2]	1970
16.46.4.15 QueueUnsubscribe() [1/2]	1970
16.46.4.16 QueueUnsubscribe() [2/2]	1971
16.46.4.17 RegisterMqttObj()	1971
16.46.4.18 RegisterTopicHandler()	1971
16.46.4.19 RemoveObj()	1972
16.46.4.20 RemoveTopicHandler() [1/2]	1972
16.46.4.21 RemoveTopicHandler() [2/2]	1972
16.46.4.22 SendNextPubReq()	1973
16.46.4.23 SendNextSubReq()	1973
16.46.4.24 SetClientCert()	1973
16.46.4.25 SetConnectNotify()	1973
16.46.4.26 SetRateLimit()	1974
16.46.4.27 SetRateLimit_Publish()	1974
16.46.4.28 SetRateLimit_Subscribe()	1974
16.46.4.29 StartTask()	1975
16.46.4.30 Subscribe() [1/2]	1975
16.46.4.31 Subscribe() [2/2]	1975
16.46.4.32 Unsubscribe() [1/2]	1976

16.46.4.33 Unsubscribe() [2/2]	1976
16.47 config_bool Class Reference	1976
16.47.1 Detailed Description	1977
16.47.2 Constructor & Destructor Documentation	1977
16.47.2.1 config_bool() [1/2]	1977
16.47.2.2 config_bool() [2/2]	1978
16.47.3 Member Function Documentation	1978
16.47.3.1 GetTextValue()	1978
16.47.3.2 GetTypeValue()	1978
16.47.3.3 operator bool()	1978
16.47.3.4 operator=() [1/3]	1978
16.47.3.5 operator=() [2/3]	1979
16.47.3.6 operator=() [3/3]	1979
16.48 config_chooser Class Reference	1979
16.48.1 Detailed Description	1980
16.48.2 Constructor & Destructor Documentation	1980
16.48.2.1 config_chooser() [1/2]	1980
16.48.2.2 config_chooser() [2/2]	1981
16.48.3 Member Function Documentation	1981
16.48.3.1 GetChoices()	1981
16.48.3.2 GetTypeValue()	1981
16.48.3.3 IsInChoices() [1/2]	1981
16.48.3.4 IsInChoices() [2/2]	1982
16.48.3.5 IsSelected() [1/2]	1982
16.48.3.6 IsSelected() [2/2]	1982
16.48.3.7 operator NBString()	1983
16.48.3.8 operator=() [1/3]	1983
16.48.3.9 operator=() [2/3]	1983
16.48.3.10 operator=() [3/3]	1983
16.48.3.11 SetChoices()	1983
16.49 config_double Class Reference	1984
16.49.1 Detailed Description	1984
16.49.2 Constructor & Destructor Documentation	1985
16.49.2.1 config_double() [1/2]	1985
16.49.2.2 config_double() [2/2]	1985
16.49.3 Member Function Documentation	1985
16.49.3.1 GetTextValue()	1985
16.49.3.2 GetTypeValue()	1985
16.49.3.3 operator double()	1986
16.49.3.4 operator float()	1986
16.49.3.5 operator int()	1986
16.49.3.6 operator=() [1/2]	1986

16.49.3.7 operator=() [2/2]	1986
16.50 config_double_limit Class Reference	1987
16.50.1 Detailed Description	1988
16.50.2 Constructor & Destructor Documentation	1988
16.50.2.1 config_double_limit() [1/2]	1988
16.50.2.2 config_double_limit() [2/2]	1988
16.50.3 Member Function Documentation	1989
16.50.3.1 GetTypeValue()	1989
16.50.3.2 operator=() [1/2]	1989
16.50.3.3 operator=() [2/2]	1989
16.51 config_int Class Reference	1989
16.51.1 Detailed Description	1990
16.51.2 Constructor & Destructor Documentation	1990
16.51.2.1 config_int() [1/2]	1990
16.51.2.2 config_int() [2/2]	1990
16.51.3 Member Function Documentation	1991
16.51.3.1 GetTextValue()	1991
16.51.3.2 GetTypeValue()	1991
16.51.3.3 operator int()	1991
16.51.3.4 operator=() [1/2]	1991
16.51.3.5 operator=() [2/2]	1992
16.52 config_int_limit Class Reference	1993
16.52.1 Detailed Description	1994
16.52.2 Constructor & Destructor Documentation	1994
16.52.2.1 config_int_limit() [1/2]	1994
16.52.2.2 config_int_limit() [2/2]	1994
16.52.3 Member Function Documentation	1994
16.52.3.1 operator=() [1/2]	1994
16.52.3.2 operator=() [2/2]	1995
16.53 config_IPADDR Class Reference	1995
16.53.1 Detailed Description	1996
16.53.2 Constructor & Destructor Documentation	1996
16.53.2.1 config_IPADDR() [1/4]	1996
16.53.2.2 config_IPADDR() [2/4]	1996
16.53.2.3 config_IPADDR() [3/4]	1996
16.53.2.4 config_IPADDR() [4/4]	1997
16.53.3 Member Function Documentation	1997
16.53.3.1 GetTextValue()	1997
16.53.3.2 GetTypeValue()	1997
16.53.3.3 IsNull()	1997
16.53.3.4 NotNull()	1998
16.53.3.5 operator IPADDR()	1998

16.53.3.6 operator=()	[1/2]	1998
16.53.3.7 operator=()	[2/2]	1998
16.54 config_IPADDR4 Class Reference		1999
16.54.1 Detailed Description		1999
16.54.2 Constructor & Destructor Documentation		2000
16.54.2.1 config_IPADDR4()	[1/4]	2000
16.54.2.2 config_IPADDR4()	[2/4]	2000
16.54.2.3 config_IPADDR4()	[3/4]	2000
16.54.2.4 config_IPADDR4()	[4/4]	2001
16.54.3 Member Function Documentation		2001
16.54.3.1 GetTextValue()		2001
16.54.3.2 GetTypeValue()		2001
16.54.3.3 IsNull()		2001
16.54.3.4 NotNull()		2002
16.54.3.5 operator IPADDR4()		2003
16.54.3.6 operator=()	[1/2]	2003
16.54.3.7 operator=()	[2/2]	2003
16.54.3.8 SetNull()		2003
16.55 config_localname Class Reference		2003
16.55.1 Detailed Description		2005
16.55.2 Constructor & Destructor Documentation		2005
16.55.2.1 config_localname()	[1/4]	2005
16.55.2.2 config_localname()	[2/4]	2005
16.55.2.3 config_localname()	[3/4]	2005
16.55.2.4 config_localname()	[4/4]	2006
16.55.3 Member Function Documentation		2006
16.55.3.1 operator NBString()		2006
16.55.3.2 operator=()	[1/4]	2006
16.55.3.3 operator=()	[2/4]	2006
16.55.3.4 operator=()	[3/4]	2007
16.55.3.5 operator=()	[4/4]	2007
16.56 config_MACADR Class Reference		2007
16.56.1 Detailed Description		2008
16.56.2 Constructor & Destructor Documentation		2008
16.56.2.1 config_MACADR()	[1/4]	2008
16.56.2.2 config_MACADR()	[2/4]	2008
16.56.2.3 config_MACADR()	[3/4]	2009
16.56.2.4 config_MACADR()	[4/4]	2009
16.56.3 Member Function Documentation		2009
16.56.3.1 GetTextValue()		2009
16.56.3.2 GetTypeValue()		2009
16.56.3.3 operator MACADR()		2011

16.56.3.4 operator=() [1/2]	2011
16.56.3.5 operator=() [2/2]	2011
16.57 config_obj Class Reference	2011
16.57.1 Detailed Description	2012
16.57.2 Constructor & Destructor Documentation	2012
16.57.2.1 config_obj() [1/2]	2012
16.57.2.2 config_obj() [2/2]	2013
16.57.3 Member Function Documentation	2013
16.57.3.1 GetTextValue()	2013
16.57.3.2 GetTypeValue()	2013
16.58 config_pass Class Reference	2013
16.58.1 Detailed Description	2015
16.58.2 Constructor & Destructor Documentation	2015
16.58.2.1 config_pass() [1/4]	2015
16.58.2.2 config_pass() [2/4]	2015
16.58.2.3 config_pass() [3/4]	2016
16.58.2.4 config_pass() [4/4]	2016
16.58.3 Member Function Documentation	2016
16.58.3.1 GetRawValue()	2016
16.58.3.2 GetTextValue()	2016
16.58.3.3 operator NBString()	2017
16.58.3.4 operator=() [1/4]	2017
16.58.3.5 operator=() [2/4]	2017
16.58.3.6 operator=() [3/4]	2017
16.58.3.7 operator=() [4/4]	2017
16.59 config_pass_limit Class Reference	2018
16.59.1 Detailed Description	2019
16.59.2 Constructor & Destructor Documentation	2019
16.59.2.1 config_pass_limit() [1/2]	2019
16.59.2.2 config_pass_limit() [2/2]	2020
16.59.3 Member Function Documentation	2020
16.59.3.1 operator=() [1/2]	2020
16.59.3.2 operator=() [2/2]	2020
16.60 config_string Class Reference	2020
16.60.1 Detailed Description	2021
16.60.2 Constructor & Destructor Documentation	2022
16.60.2.1 config_string() [1/4]	2022
16.60.2.2 config_string() [2/4]	2022
16.60.2.3 config_string() [3/4]	2022
16.60.2.4 config_string() [4/4]	2023
16.60.3 Member Function Documentation	2023
16.60.3.1 c_str()	2023

16.60.3.2	GetTextValue()	2023
16.60.3.3	GetTypeValue()	2023
16.60.3.4	length()	2023
16.60.3.5	operator NBString()	2024
16.60.3.6	operator=() [1/3]	2024
16.60.3.7	operator=() [2/3]	2024
16.60.3.8	operator=() [3/3]	2024
16.60.3.9	operator[]()	2024
16.60.3.10	SetEnumList()	2025
16.61	config_string_limit Class Reference	2025
16.61.1	Detailed Description	2026
16.61.2	Constructor & Destructor Documentation	2026
16.61.2.1	config_string_limit() [1/2]	2026
16.61.2.2	config_string_limit() [2/2]	2026
16.61.3	Member Function Documentation	2028
16.61.3.1	operator=() [1/2]	2028
16.61.3.2	operator=() [2/2]	2028
16.62	config_time_t Class Reference	2028
16.62.1	Detailed Description	2030
16.62.2	Constructor & Destructor Documentation	2030
16.62.2.1	config_time_t() [1/4]	2030
16.62.2.2	config_time_t() [2/4]	2030
16.62.2.3	config_time_t() [3/4]	2031
16.62.2.4	config_time_t() [4/4]	2031
16.62.3	Member Function Documentation	2031
16.62.3.1	operator time_t()	2031
16.62.3.2	operator=() [1/2]	2031
16.62.3.3	operator=() [2/2]	2032
16.63	config_uint Class Reference	2032
16.63.1	Detailed Description	2033
16.63.2	Constructor & Destructor Documentation	2033
16.63.2.1	config_uint() [1/2]	2033
16.63.2.2	config_uint() [2/2]	2033
16.63.3	Member Function Documentation	2033
16.63.3.1	GetTextValue()	2033
16.63.3.2	GetTypeValue()	2034
16.63.3.3	operator uint32_t()	2034
16.63.3.4	operator=() [1/2]	2034
16.63.3.5	operator=() [2/2]	2034
16.64	config_uint_limit Class Reference	2034
16.64.1	Detailed Description	2035
16.64.2	Constructor & Destructor Documentation	2035

16.64.2.1 config_uint_limit() [1/2]	2035
16.64.2.2 config_uint_limit() [2/2]	2036
16.64.3 Member Function Documentation	2036
16.64.3.1 operator=() [1/2]	2036
16.64.3.2 operator=() [2/2]	2036
16.65 config_value Class Reference	2037
16.65.1 Detailed Description	2037
16.65.2 Constructor & Destructor Documentation	2037
16.65.2.1 config_value() [1/2]	2037
16.65.2.2 config_value() [2/2]	2037
16.66 MQTT::ConfigExposer Class Reference	2038
16.66.1 Detailed Description	2040
16.66.2 Member Enumeration Documentation	2040
16.66.2.1 eRenderFlag_t	2040
16.66.3 Constructor & Destructor Documentation	2041
16.66.3.1 ConfigExposer() [1/2]	2041
16.66.3.2 ConfigExposer() [2/2]	2041
16.66.4 Member Function Documentation	2041
16.66.4.1 GetPubTopicName()	2041
16.66.4.2 GetRenderLength()	2042
16.66.4.3 GetSubTopicFilter()	2042
16.66.4.4 ParseFd()	2042
16.66.4.5 RenderToBuffer()	2042
16.66.4.6 RenderToFd()	2043
16.66.4.7 SubscribeEvent()	2043
16.66.4.8 TestOrSetValue()	2043
16.67 MQTT::ConfiguredClient Class Reference	2044
16.67.1 Detailed Description	2046
16.67.2 Constructor & Destructor Documentation	2046
16.67.2.1 ConfiguredClient()	2046
16.68 MQTT::ConnectRequest Struct Reference	2047
16.68.1 Detailed Description	2047
16.69 CUR_IPADDR4 Class Reference	2047
16.69.1 Detailed Description	2047
16.69.2 Constructor & Destructor Documentation	2047
16.69.2.1 CUR_IPADDR4()	2047
16.69.3 Member Function Documentation	2047
16.69.3.1 CommitTestedValue()	2047
16.69.3.2 ExtendedSchema()	2048
16.69.3.3 GetTextValue()	2048
16.69.3.4 GetTypeValue()	2048
16.69.3.5 IsAutoIP()	2048

16.69.3.6 IsGlobalBroadCast()	2048
16.69.3.7 IsLoopBack()	2048
16.69.3.8 IsMultiCast()	2048
16.69.3.9 IsNull()	2048
16.69.3.10 NotNull()	2048
16.69.3.11 operator IPADDR4()	2048
16.69.3.12 SetNull()	2048
16.69.3.13 TestNewValue()	2048
16.70 DelayObject Class Reference	2049
16.70.1 Detailed Description	2049
16.70.2 Constructor & Destructor Documentation	2050
16.70.2.1 DelayObject()	2050
16.70.2.2 ~DelayObject()	2050
16.70.3 Member Function Documentation	2051
16.70.3.1 _wake()	2051
16.70.3.2 DelayUsec()	2051
16.70.3.3 valid()	2052
16.71 DHCP::DhcpInfo Struct Reference	2053
16.71.1 Detailed Description	2053
16.72 DHCP::DhcpLeaseData Struct Reference	2053
16.72.1 Detailed Description	2054
16.73 DHCP::DhcpLeaseRequest Struct Reference	2054
16.73.1 Detailed Description	2054
16.74 DhcpObject Class Reference	2054
16.74.1 Detailed Description	2055
16.74.2 Member Function Documentation	2055
16.74.2.1 bDoFallBack()	2055
16.74.2.2 GetDhcpExpirationTime()	2055
16.74.2.3 GetDhcpRebindTime()	2055
16.74.2.4 GetDhcpRenewTime()	2056
16.74.2.5 GetDHCPState()	2056
16.74.2.6 GetRemainingDhcpLeaseTime()	2056
16.74.2.7 RebindDHCP()	2056
16.74.2.8 RenewDHCP()	2056
16.74.2.9 RestartDHCP()	2057
16.74.2.10 StartDHCP()	2057
16.74.2.11 StopDHCP()	2057
16.74.2.12 ValidDhcpLease()	2057
16.75 DiagCallBack Class Reference	2057
16.75.1 Detailed Description	2058
16.75.2 Constructor & Destructor Documentation	2058
16.75.2.1 DiagCallBack()	2058

16.76 DiagItemClass Class Reference	2058
16.76.1 Detailed Description	2058
16.77 DiagPtrMon Class Reference	2058
16.77.1 Detailed Description	2058
16.77.2 Constructor & Destructor Documentation	2059
16.77.2.1 DiagPtrMon()	2059
16.78 DiagStrMon Class Reference	2060
16.78.1 Detailed Description	2060
16.78.2 Constructor & Destructor Documentation	2060
16.78.2.1 DiagStrMon() [1/2]	2060
16.78.2.2 DiagStrMon() [2/2]	2060
16.79 DiagVarMon Class Reference	2061
16.79.1 Detailed Description	2061
16.79.2 Constructor & Destructor Documentation	2061
16.79.2.1 DiagVarMon()	2061
16.80 NB::Wifi::driverStatusStruct Struct Reference	2061
16.80.1 Detailed Description	2062
16.81 dspIDMAStruct Struct Reference	2062
16.81.1 Detailed Description	2063
16.82 dspIDriverStruct Struct Reference	2063
16.82.1 Detailed Description	2064
16.83 DSPIModule Class Reference	2064
16.83.1 Detailed Description	2065
16.83.2 Constructor & Destructor Documentation	2065
16.83.2.1 DSPIModule() [1/2]	2065
16.83.2.2 DSPIModule() [2/2]	2065
16.83.3 Member Function Documentation	2066
16.83.3.1 ClrSem()	2066
16.83.3.2 DisableDMA()	2066
16.83.3.3 Done() [1/2]	2066
16.83.3.4 Done() [2/2]	2066
16.83.3.5 EnableDMA()	2067
16.83.3.6 GetActualBaudrate()	2068
16.83.3.7 GetSem()	2068
16.83.3.8 Init()	2068
16.83.3.9 RegisterSem()	2069
16.83.3.10 Rx()	2069
16.83.3.11 SetCS()	2069
16.83.3.12 Start()	2070
16.83.3.13 Tx()	2070
16.83.4 Friends And Related Symbol Documentation	2071
16.83.4.1 DSPIInit	2071

16.84 Dynamic_IPADDR Class Reference	2072
16.84.1 Detailed Description	2073
16.84.2 Constructor & Destructor Documentation	2073
16.84.2.1 Dynamic_IPADDR()	2073
16.84.3 Member Function Documentation	2073
16.84.3.1 AddToJsonOutObject()	2073
16.84.3.2 CommitTestedValue()	2073
16.84.3.3 ExtendedSchema()	2073
16.84.3.4 GetTextValue()	2073
16.84.3.5 GetTypeValue()	2073
16.84.3.6 TestNewValue()	2073
16.85 EBI_CS_cfg_t Struct Reference	2074
16.85.1 Detailed Description	2074
16.85.1.1 Timing Diagram Reference	2074
16.86 Effs_Cfg Struct Reference	2078
16.86.1 Detailed Description	2078
16.87 fifo_buffer_storage Class Reference	2078
16.87.1 Detailed Description	2079
16.87.2 Constructor & Destructor Documentation	2079
16.87.2.1 fifo_buffer_storage()	2079
16.87.3 Member Function Documentation	2079
16.87.3.1 Empty()	2079
16.87.3.2 Full()	2080
16.87.3.3 Peek_Begin()	2080
16.87.3.4 Peek_End()	2080
16.87.3.5 PeekData()	2080
16.87.3.6 ReadData()	2080
16.87.3.7 Reset()	2080
16.87.3.8 SpaceAvail()	2081
16.87.3.9 SpaceUsed()	2081
16.87.3.10 WriteData()	2081
16.88 HtmlPageHandler Class Reference	2081
16.88.1 Detailed Description	2082
16.88.2 Constructor & Destructor Documentation	2083
16.88.2.1 HtmlPageHandler()	2083
16.88.3 Member Function Documentation	2083
16.88.3.1 FindHandler()	2083
16.88.3.2 GetGroup()	2083
16.88.3.3 MoveHandlerToLast()	2084
16.88.3.4 ProcessRaw()	2084
16.89 HtmlPostVariableListCallback Class Reference	2084
16.89.1 Detailed Description	2084

16.89.2 Constructor & Destructor Documentation	2084
16.89.2.1 HtmlPostVariableListCallback() [1/2]	2084
16.89.2.2 HtmlPostVariableListCallback() [2/2]	2085
16.90 HTTP_Request Struct Reference	2085
16.90.1 Detailed Description	2086
16.90.2 Member Function Documentation	2086
16.90.2.1 GetBoundaryMarker()	2086
16.90.2.2 ReadSimpleBody()	2087
16.91 I2C Class Reference	2087
16.91.1 Detailed Description	2088
16.91.2 Member Enumeration Documentation	2088
16.91.2.1 Result_t	2088
16.91.3 Constructor & Destructor Documentation	2089
16.91.3.1 I2C()	2089
16.91.4 Member Function Documentation	2089
16.91.4.1 DoTransaction()	2089
16.91.4.2 readReg8()	2089
16.91.4.3 readRegN()	2089
16.91.4.4 setNumAddressBytes()	2090
16.91.4.5 setup()	2090
16.91.4.6 writeReg8()	2090
16.91.4.7 writeRegN()	2091
16.92 I2CDevice Class Reference	2091
16.92.1 Detailed Description	2091
16.92.2 Constructor & Destructor Documentation	2092
16.92.2.1 I2CDevice()	2092
16.92.3 Member Function Documentation	2092
16.92.3.1 getI2CAddress()	2092
16.92.3.2 readReg8()	2092
16.92.3.3 readRegN()	2092
16.92.3.4 setup()	2093
16.92.3.5 writeReg8()	2093
16.92.3.6 writeRegN()	2093
16.93 I4Record Class Reference	2094
16.93.1 Detailed Description	2094
16.93.2 Constructor & Destructor Documentation	2094
16.93.2.1 I4Record()	2094
16.93.3 Member Function Documentation	2094
16.93.3.1 Attach()	2094
16.93.3.2 GetReportItem()	2094
16.94 I6Record Class Reference	2094
16.94.1 Detailed Description	2095

16.94.2 Constructor & Destructor Documentation	2095
16.94.2.1 I6Record()	2095
16.94.3 Member Function Documentation	2095
16.94.3.1 Attach()	2095
16.94.3.2 GetnIf()	2095
16.94.3.3 ReportGetTextValue()	2095
16.95 IEEE1588Timer Class Reference	2095
16.95.1 Detailed Description	2096
16.95.2 Member Function Documentation	2096
16.95.2.1 enablePPSCapture()	2096
16.95.2.2 getConsecutiveValidSeconds()	2096
16.95.2.3 getCurrentTime()	2096
16.95.2.4 getPPSCapture()	2096
16.95.2.5 getPPSLoop()	2096
16.95.2.6 init()	2097
16.95.2.7 markValidTime()	2097
16.95.2.8 PPS_1588_TimeCapIsr()	2097
16.95.2.9 resetConsecutiveSeconds()	2097
16.95.2.10 ring_diff_forward()	2097
16.95.2.11 updateConsecutiveSeconds()	2097
16.96 InterfaceBlock Class Reference	2097
16.96.1 Detailed Description	2099
16.96.2 Constructor & Destructor Documentation	2100
16.96.2.1 InterfaceBlock() [1/2]	2100
16.96.2.2 InterfaceBlock() [2/2]	2100
16.96.3 Member Function Documentation	2100
16.96.3.1 EnableMulticast()	2100
16.96.3.2 GetInterfaceName()	2100
16.96.3.3 GetInterfaceNumber()	2101
16.96.3.4 InterfaceLinkChange()	2101
16.96.3.5 IsRootInterface()	2101
16.96.3.6 LinkActive()	2101
16.96.3.7 LinkDuplex()	2101
16.96.3.8 LinkSpeed()	2102
16.96.3.9 RegisterInterface()	2102
16.97 IPADDR4 Class Reference	2102
16.97.1 Detailed Description	2103
16.97.2 Constructor & Destructor Documentation	2103
16.97.2.1 IPADDR4()	2103
16.97.3 Member Function Documentation	2103
16.97.3.1 fdprint()	2103
16.97.3.2 IsAutoIP()	2103

16.97.3.3 IsGlobalBroadCast()	2104
16.97.3.4 IsLoopBack()	2104
16.97.3.5 IsmDns()	2104
16.97.3.6 IsMultiCast()	2104
16.97.3.7 IsNull()	2104
16.97.3.8 NotNull()	2105
16.97.3.9 SetFromAscii()	2105
16.97.3.10 SetNull()	2105
16.97.3.11 sprintf()	2105
16.98 IPADDR6 Class Reference	2106
16.98.1 Detailed Description	2106
16.98.2 Member Function Documentation	2107
16.98.2.1 AsciiToIp6()	2107
16.98.2.2 Extract4()	2107
16.98.2.3 fdprint()	2107
16.98.2.4 GetEmbeddedIP4UInt()	2108
16.98.2.5 IsEmbeddedIPV4()	2108
16.98.2.6 IsLinkLocal()	2108
16.98.2.7 IsLoopBack()	2108
16.98.2.8 IsMultiCast()	2109
16.98.2.9 IsNull()	2109
16.98.2.10 McastMac()	2109
16.98.2.11 NotNull()	2109
16.98.2.12 NullIP()	2109
16.98.2.13 print()	2110
16.98.2.14 SetFromAscii()	2110
16.98.2.15 SetFromIP4()	2110
16.98.2.16 SetFromUInt32Shortcut()	2111
16.98.2.17 SetNull()	2111
16.98.2.18 sprintf()	2111
16.99 IPPKT Struct Reference	2112
16.99.1 Detailed Description	2112
16.99.2 Member Data Documentation	2112
16.99.2.1 hCSum	2112
16.100 IPV6_DEST Struct Reference	2113
16.100.1 Detailed Description	2113
16.100.2 Member Function Documentation	2114
16.100.2.1 AgeStillValidTest()	2114
16.100.2.2 GetNext()	2114
16.101 IPV6_DHCPD Struct Reference	2114
16.101.1 Detailed Description	2115
16.101.2 Member Function Documentation	2115

16.101.2.1 AgeStillValidTest()	2115
16.101.2.2 FindIA_Addr()	2115
16.101.2.3 GetDhcpRebindTime()	2115
16.101.2.4 GetDhcpRenewTime()	2115
16.102 IPV6_DNS Struct Reference	2116
16.102.1 Detailed Description	2116
16.102.2 Member Function Documentation	2117
16.102.2.1 AgeStillValidTest()	2117
16.102.2.2 GetNext()	2117
16.103 IPV6_IA_ADDR Struct Reference	2117
16.103.1 Detailed Description	2118
16.103.2 Member Function Documentation	2118
16.103.2.1 AgeStillValidTest()	2118
16.104 IPV6_NEIGHBOR Struct Reference	2118
16.104.1 Detailed Description	2119
16.104.2 Member Function Documentation	2119
16.104.2.1 AgeStillValidTest()	2119
16.104.2.2 CoreSendPend()	2119
16.104.2.3 GetNext()	2120
16.104.2.4 GetState()	2120
16.104.2.5 ProcessAdvert()	2120
16.104.2.6 Send()	2120
16.104.2.7 Send_ND_Solicit()	2120
16.104.2.8 SendPending()	2121
16.104.2.9 SetState()	2121
16.104.2.10 StillValidToSend()	2121
16.105 IPV6_PREFIX Struct Reference	2121
16.105.1 Detailed Description	2123
16.105.2 Member Function Documentation	2123
16.105.2.1 AgeStillPreferred()	2123
16.105.2.2 AgeStillValidTest()	2123
16.105.2.3 GetNext()	2123
16.105.2.4 GetRemainingDhcpLeasePreferredTime()	2123
16.105.2.5 GetRemainingDhcpLeaseTime()	2123
16.105.2.6 GetState()	2124
16.105.2.7 OnLink()	2124
16.105.2.8 Preferred()	2124
16.105.2.9 RemainingValidTime()	2124
16.105.2.10 Source()	2124
16.106 IPV6_ROOT_EL Struct Reference	2124
16.106.1 Detailed Description	2125
16.106.2 Member Function Documentation	2125

16.106.2.1 AgeStillValidTest()	2125
16.107 IPV6_ROOT_EL_CONTAINER Struct Reference	2125
16.107.1 Detailed Description	2126
16.107.2 Member Function Documentation	2126
16.107.2.1 Attach()	2126
16.107.2.2 Find()	2126
16.107.2.3 First()	2127
16.107.2.4 Insert()	2127
16.107.2.5 InsertBack()	2127
16.107.2.6 MoveToTop()	2127
16.107.2.7 Remove()	2127
16.108 IPV6_ROUTER Struct Reference	2128
16.108.1 Detailed Description	2129
16.108.2 Member Function Documentation	2129
16.108.2.1 AgeStillValidTest()	2129
16.108.2.2 GetNext()	2129
16.108.2.3 GetNextValid()	2129
16.109 IPv6FrameProcessingStruct Struct Reference	2129
16.109.1 Detailed Description	2130
16.109.2 Constructor & Destructor Documentation	2130
16.109.2.1 IPv6FrameProcessingStruct()	2130
16.109.3 Member Function Documentation	2131
16.109.3.1 DestOptHasError()	2131
16.109.3.2 GetDataPointer()	2131
16.109.3.3 GetEframe()	2131
16.109.3.4 HopByHopParseHasError()	2131
16.109.3.5 RouterHeaderParseHasError()	2131
16.109.3.6 WalkNextHeader()	2132
16.110 IPv6Interface Class Reference	2132
16.110.1 Detailed Description	2134
16.110.2 Constructor & Destructor Documentation	2134
16.110.2.1 IPv6Interface()	2134
16.110.3 Member Function Documentation	2134
16.110.3.1 AddDefaultGateway()	2134
16.110.3.2 AddStaticAddress()	2134
16.110.3.3 AddStaticDNS()	2135
16.110.3.4 GetDestinationMTU()	2135
16.110.3.5 GetFirst_IP6_Interface()	2135
16.110.3.6 GetInterfaceForDestination()	2135
16.110.3.7 GetInterfaceForSource()	2136
16.110.3.8 GetInterfaceN()	2136
16.110.3.9 GetInterfaceNumber()	2136

16.110.3.10 GetMyFirstAddress()	2137
16.110.3.11 GetMyNextAddress()	2137
16.110.3.12 GetNext_IP6_Interface()	2137
16.110.3.13 HadLink()	2137
16.110.3.14 HasRoute()	2137
16.110.3.15 IsMyAddress()	2138
16.110.3.16 MyDNSAddress()	2138
16.110.3.17 MySourceAddress()	2138
16.110.3.18 NotifyReachable()	2138
16.110.3.19 NotifyUnreachable()	2138
16.110.3.20 OnLink()	2139
16.110.3.21 ping()	2139
16.110.3.22 RemoveDefaultGateway()	2139
16.110.3.23 RemoveStaticAddress()	2140
16.110.3.24 RemoveStaticDNS()	2140
16.110.3.25 SendMLDRegistration()	2140
16.110.3.26 SetStaticDNS()	2141
16.110.3.27 ShowAllInfo()	2141
16.110.3.28 ShowInfo()	2141
16.110.4 Friends And Related Symbol Documentation	2141
16.110.4.1 InitIPv6	2141
16.110.4.2 Ping6	2141
16.111 JsonAllocString Struct Reference	2142
16.111.1 Detailed Description	2142
16.112 JsonLexerFDAdapter Class Reference	2142
16.112.1 Detailed Description	2147
16.113 JsonRef Class Reference	2147
16.113.1 Detailed Description	2149
16.113.2 Member Function Documentation	2149
16.113.2.1 GetChildPrintSize()	2149
16.113.2.2 IsArray()	2150
16.113.2.3 IsBool()	2150
16.113.2.4 IsNull()	2150
16.113.2.5 IsNumber()	2150
16.113.2.6 IsObject()	2150
16.113.2.7 IsString()	2150
16.113.2.8 MakeInvalid()	2150
16.113.2.9 name()	2151
16.113.2.10 object()	2151
16.113.2.11 operator bool()	2151
16.113.2.12 operator const char *()	2151
16.113.2.13 operator double()	2152

16.113.2.14 operator float()	2152
16.113.2.15 operator int()	2152
16.113.2.16 operator int16_t()	2152
16.113.2.17 operator int32_t()	2152
16.113.2.18 operator int8_t()	2152
16.113.2.19 operator NBString()	2152
16.113.2.20 operator time_t()	2153
16.113.2.21 operator uint16_t()	2153
16.113.2.22 operator uint32_t()	2153
16.113.2.23 operator uint8_t()	2153
16.113.2.24 operator>()()	2153
16.113.2.25 operator[]()	2154
16.113.2.26 Valid()	2154
16.114 DHCP::LeaseAllocator Class Reference	2154
16.114.1 Detailed Description	2154
16.115 LetsEncryptAcmeServletObject Class Reference	2154
16.115.1 Detailed Description	2155
16.115.2 Constructor & Destructor Documentation	2155
16.115.2.1 LetsEncryptAcmeServletObject()	2155
16.116 listen_list_item Struct Reference	2155
16.116.1 Detailed Description	2155
16.117 MACADR Class Reference	2155
16.117.1 Detailed Description	2156
16.117.2 Member Function Documentation	2156
16.117.2.1 operator+()	2156
16.118 DHCP::MacPrefixAllocator Class Reference	2156
16.118.1 Detailed Description	2156
16.119 mcanMODM7AE70::mcan_config Class Reference	2156
16.119.1 Detailed Description	2157
16.119.2 Member Function Documentation	2157
16.119.2.1 set_config_defaults()	2157
16.119.3 Member Data Documentation	2158
16.119.3.1 automatic_retransmission	2158
16.119.3.2 clock_stop_acknowledge	2158
16.119.3.3 clock_stop_request	2158
16.119.3.4 delay_compensation_filter_window_length	2158
16.119.3.5 delay_compensation_offset	2158
16.119.3.6 edge_filtering	2158
16.119.3.7 extended_id_mask	2159
16.119.3.8 nonmatching_frames_action_extended	2159
16.119.3.9 nonmatching_frames_action_standard	2159
16.119.3.10 protocol_exception_handling	2159

16.119.3.11 remote_frames_extended_reject	2159
16.119.3.12 remote_frames_standard_reject	2159
16.119.3.13 run_in_standby	2159
16.119.3.14 rx_fifo_0_overwrite	2159
16.119.3.15 rx_fifo_0_watermark	2159
16.119.3.16 rx_fifo_1_overwrite	2159
16.119.3.17 rx_fifo_1_watermark	2159
16.119.3.18 tdc_enable	2159
16.119.3.19 timeout_enable	2160
16.119.3.20 timeout_mode	2160
16.119.3.21 timeout_period	2160
16.119.3.22 timestamp_prescaler	2160
16.119.3.23 transmit_pause	2160
16.119.3.24 tx_event_fifo_watermark	2160
16.119.3.25 tx_queue_mode	2160
16.119.3.26 watchdog_configuration	2160
16.120 mcan_extended_message_filter_element Class Reference	2160
16.120.1 Detailed Description	2160
16.121 mcanMODM7AE70::mcan_module Class Reference	2160
16.121.1 Detailed Description	2161
16.121.2 Constructor & Destructor Documentation	2161
16.121.2.1 mcan_module()	2161
16.121.3 Member Function Documentation	2161
16.121.3.1 blocking_send_message()	2161
16.121.3.2 init()	2163
16.121.3.3 MultiCanReplaceRTRMessage()	2163
16.121.3.4 MultiCanSetRTRMessage()	2163
16.121.3.5 MultiCanStopRTRMessage()	2164
16.121.3.6 RegisterRxFifo()	2164
16.121.3.7 RegisterRxFifoMask()	2164
16.121.3.8 RegisterRxFifoRange()	2165
16.121.3.9 send_message()	2165
16.121.3.10 UnRegisterFifo()	2165
16.122 mcan_rx_element_buffer Struct Reference	2166
16.122.1 Detailed Description	2166
16.123 mcan_tx_element Struct Reference	2166
16.123.1 Detailed Description	2166
16.124 mcan_tx_event_element Struct Reference	2166
16.124.1 Detailed Description	2166
16.125 MQTT::mqtt_bool Class Reference	2166
16.125.1 Detailed Description	2169
16.125.2 Constructor & Destructor Documentation	2169

16.125.2.1 mqtt_bool() [1/4]	2169
16.125.2.2 mqtt_bool() [2/4]	2169
16.125.2.3 mqtt_bool() [3/4]	2170
16.125.2.4 mqtt_bool() [4/4]	2170
16.125.3 Member Function Documentation	2170
16.125.3.1 operator bool()	2170
16.125.3.2 operator"!()	2170
16.125.3.3 operator=()	2171
16.125.3.4 ParseFd()	2171
16.125.3.5 RenderToBuffer()	2171
16.125.3.6 RenderToFd()	2171
16.125.3.7 SubscribeEvent()	2172
16.125.3.8 TestOrSetValue()	2172
16.126 MQTT::mqtt_float Class Reference	2172
16.126.1 Detailed Description	2175
16.126.2 Constructor & Destructor Documentation	2175
16.126.2.1 mqtt_float() [1/4]	2175
16.126.2.2 mqtt_float() [2/4]	2175
16.126.2.3 mqtt_float() [3/4]	2176
16.126.2.4 mqtt_float() [4/4]	2176
16.126.3 Member Function Documentation	2176
16.126.3.1 operator double()	2176
16.126.3.2 operator=()	2176
16.126.3.3 ParseFd()	2177
16.126.3.4 RenderToBuffer()	2177
16.126.3.5 RenderToFd()	2177
16.126.3.6 SetStringFormat()	2178
16.126.3.7 SubscribeEvent()	2178
16.126.3.8 TestOrSetValue()	2178
16.127 MQTT::mqtt_int Class Reference	2178
16.127.1 Detailed Description	2181
16.127.2 Constructor & Destructor Documentation	2181
16.127.2.1 mqtt_int() [1/4]	2181
16.127.2.2 mqtt_int() [2/4]	2181
16.127.2.3 mqtt_int() [3/4]	2182
16.127.2.4 mqtt_int() [4/4]	2182
16.127.3 Member Function Documentation	2182
16.127.3.1 operator int()	2182
16.127.3.2 operator++() [1/2]	2183
16.127.3.3 operator++() [2/2]	2183
16.127.3.4 operator--() [1/2]	2183
16.127.3.5 operator--() [2/2]	2183

16.127.3.6 operator=()	2183
16.127.3.7 ParseFd()	2183
16.127.3.8 RenderToBuffer()	2184
16.127.3.9 RenderToFd()	2184
16.127.3.10 SubscribeEvent()	2184
16.127.3.11 TestOrSetValue()	2185
16.128 MQTT::mqtt_IO Class Reference	2185
16.128.1 Detailed Description	2188
16.128.2 Constructor & Destructor Documentation	2188
16.128.2.1 mqtt_IO() [1/2]	2188
16.128.2.2 mqtt_IO() [2/2]	2188
16.128.3 Member Function Documentation	2189
16.128.3.1 ClearFlags()	2189
16.128.3.2 operator bool()	2189
16.128.3.3 operator"!()	2189
16.128.3.4 operator=()	2189
16.128.3.5 ParseFd()	2189
16.128.3.6 PinNotify()	2190
16.128.3.7 RenderToBuffer()	2190
16.128.3.8 RenderToFd()	2190
16.128.3.9 SetFixedDirection()	2190
16.128.3.10 SetFlags()	2191
16.128.3.11 SubscribeEvent()	2191
16.128.3.12 TestOrSetValue()	2191
16.129 MQTT::mqtt_IO_Input Class Reference	2192
16.129.1 Detailed Description	2194
16.129.2 Constructor & Destructor Documentation	2194
16.129.2.1 mqtt_IO_Input() [1/2]	2194
16.129.2.2 mqtt_IO_Input() [2/2]	2195
16.129.3 Member Function Documentation	2195
16.129.3.1 operator=()	2195
16.130 MQTT::mqtt_IO_Output Class Reference	2195
16.130.1 Detailed Description	2198
16.130.2 Constructor & Destructor Documentation	2198
16.130.2.1 mqtt_IO_Output() [1/2]	2198
16.130.2.2 mqtt_IO_Output() [2/2]	2199
16.130.3 Member Function Documentation	2199
16.130.3.1 operator=()	2199
16.131 MQTT::mqtt_leaf Class Reference	2199
16.131.1 Detailed Description	2201
16.131.2 Constructor & Destructor Documentation	2202
16.131.2.1 mqtt_leaf() [1/2]	2202

16.131.2.2 mqtt_leaf() [2/2]	2202
16.131.3 Member Function Documentation	2202
16.131.3.1 AddChild()	2202
16.131.3.2 Assign()	2203
16.131.3.3 ClearFlags()	2203
16.131.3.4 ForEachLeaf()	2203
16.131.3.5 GetChildren()	2203
16.131.3.6 GetFlags()	2203
16.131.3.7 getPriority()	2203
16.131.3.8 GetPubTopicName()	2204
16.131.3.9 GetRenderLength()	2204
16.131.3.10 GetSerializationForm()	2204
16.131.3.11 GetSubTopicFilter()	2204
16.131.3.12 HandlePubAck()	2204
16.131.3.13 HandleSubAck()	2205
16.131.3.14 HandleUnsubAck()	2205
16.131.3.15 MatchId()	2205
16.131.3.16 MqttRxBublish()	2206
16.131.3.17 NameMatch()	2206
16.131.3.18 ParseFd()	2206
16.131.3.19 Publish()	2206
16.131.3.20 PubState()	2207
16.131.3.21 QueuePublish()	2207
16.131.3.22 QueueSubscribe()	2207
16.131.3.23 QueueUnsubscribe()	2207
16.131.3.24 RemoveFirstChild()	2208
16.131.3.25 RenderToBuffer()	2208
16.131.3.26 RenderToFd()	2208
16.131.3.27 RootParse()	2208
16.131.3.28 SetFlags()	2209
16.131.3.29 SetPublishInterval()	2209
16.131.3.30 SetPubState()	2209
16.131.3.31 SetSerializationForm()	2209
16.131.3.32 SetSubState()	2209
16.131.3.33 SubscribeEvent()	2210
16.131.3.34 SubState()	2210
16.131.3.35 TestOrSetValue()	2210
16.132 MQTT::mqtt_obj Class Reference	2211
16.132.1 Detailed Description	2213
16.132.2 Constructor & Destructor Documentation	2213
16.132.2.1 mqtt_obj() [1/2]	2213
16.132.2.2 mqtt_obj() [2/2]	2213

16.132.3 Member Function Documentation	2214
16.132.3.1 AddChild()	2214
16.132.3.2 FindChild()	2214
16.132.3.3 GetChildren()	2214
16.132.3.4 GetRenderLength()	2214
16.132.3.5 GetSubTopicFilter()	2215
16.132.3.6 ParseFd()	2215
16.132.3.7 RemoveFirstChild()	2215
16.132.3.8 RenderToBuffer()	2215
16.132.3.9 RenderToFd()	2216
16.132.3.10 SetPubState()	2216
16.132.3.11 SetSubState()	2216
16.132.3.12 SubscribeEvent()	2216
16.132.3.13 TestOrSetValue()	2217
16.133 MQTT::mqtt_string Class Reference	2217
16.133.1 Detailed Description	2220
16.133.2 Constructor & Destructor Documentation	2220
16.133.2.1 mqtt_string() [1/6]	2220
16.133.2.2 mqtt_string() [2/6]	2220
16.133.2.3 mqtt_string() [3/6]	2221
16.133.2.4 mqtt_string() [4/6]	2221
16.133.2.5 mqtt_string() [5/6]	2221
16.133.2.6 mqtt_string() [6/6]	2222
16.133.3 Member Function Documentation	2222
16.133.3.1 operator NBString()	2222
16.133.3.2 operator=() [1/2]	2222
16.133.3.3 operator=() [2/2]	2222
16.133.3.4 operator[]() [1/2]	2223
16.133.3.5 operator[]() [2/2]	2223
16.133.3.6 ParseFd()	2223
16.133.3.7 RenderToBuffer()	2223
16.133.3.8 RenderToFd()	2223
16.133.3.9 SubscribeEvent()	2224
16.133.3.10 TestOrSetValue()	2224
16.134 MQTT::mqtt_uint Class Reference	2224
16.134.1 Detailed Description	2227
16.134.2 Constructor & Destructor Documentation	2227
16.134.2.1 mqtt_uint() [1/4]	2227
16.134.2.2 mqtt_uint() [2/4]	2227
16.134.2.3 mqtt_uint() [3/4]	2228
16.134.2.4 mqtt_uint() [4/4]	2228
16.134.3 Member Function Documentation	2228

16.134.3.1 operator unsigned()	2228
16.134.3.2 operator++() [1/2]	2229
16.134.3.3 operator++() [2/2]	2229
16.134.3.4 operator--() [1/2]	2229
16.134.3.5 operator--() [2/2]	2229
16.134.3.6 operator=()	2229
16.134.3.7 ParseFd()	2229
16.134.3.8 RenderToBuffer()	2230
16.134.3.9 RenderToFd()	2230
16.134.3.10 SubscribeEvent()	2230
16.134.3.11 TestOrSetValue()	2231
16.135 MQTT::PacketInfo::Msg Struct Reference	2231
16.135.1 Detailed Description	2231
16.135.2 Constructor & Destructor Documentation	2231
16.135.2.1 Msg()	2231
16.135.3 Member Function Documentation	2231
16.135.3.1 GetFlags()	2231
16.135.3.2 GetType()	2231
16.136 NBInterpolatedString Class Reference	2232
16.136.1 Detailed Description	2234
16.136.2 Constructor & Destructor Documentation	2234
16.136.2.1 NBInterpolatedString() [1/4]	2234
16.136.2.2 NBInterpolatedString() [2/4]	2235
16.136.2.3 NBInterpolatedString() [3/4]	2235
16.136.2.4 NBInterpolatedString() [4/4]	2235
16.136.3 Member Function Documentation	2235
16.136.3.1 Append()	2235
16.136.3.2 FdAppend()	2236
16.136.3.3 Interpolate()	2236
16.137 NBString Class Reference	2236
16.137.1 Detailed Description	2238
16.137.2 Constructor & Destructor Documentation	2238
16.137.2.1 NBString() [1/5]	2238
16.137.2.2 NBString() [2/5]	2238
16.137.2.3 NBString() [3/5]	2239
16.137.2.4 NBString() [4/5]	2239
16.137.2.5 NBString() [5/5]	2239
16.137.3 Member Function Documentation	2239
16.137.3.1 Append()	2239
16.137.3.2 c_str()	2240
16.137.3.3 clear()	2240
16.137.3.4 compare() [1/2]	2240

16.137.3.5 compare() [2/2]	2240
16.137.3.6 copy()	2240
16.137.3.7 empty()	2241
16.137.3.8 FdAppend()	2241
16.137.3.9 find() [1/4]	2241
16.137.3.10 find() [2/4]	2242
16.137.3.11 find() [3/4]	2242
16.137.3.12 find() [4/4]	2242
16.137.3.13 Interpolate() [1/2]	2243
16.137.3.14 Interpolate() [2/2]	2243
16.137.3.15 length()	2243
16.137.3.16 replace() [1/2]	2243
16.137.3.17 replace() [2/2]	2244
16.137.3.18 Reserve()	2244
16.137.3.19 siprintf() [1/2]	2244
16.137.3.20 siprintf() [2/2]	2245
16.137.3.21 size()	2245
16.137.3.22 sprintf() [1/2]	2245
16.137.3.23 sprintf() [2/2]	2245
16.137.3.24 stod()	2245
16.137.3.25 stoi()	2245
16.137.3.26 stol()	2246
16.137.3.27 stoui()	2246
16.137.3.28 stoul()	2246
16.137.3.29 strcpy()	2246
16.137.3.30 substr()	2246
16.137.3.31 to_ipaddr()	2247
16.137.3.32 vsiprintf()	2247
16.137.4 Friends And Related Symbol Documentation	2247
16.137.4.1 swap	2247
16.138 NTPServer Class Reference	2247
16.138.1 Detailed Description	2247
16.138.2 Constructor & Destructor Documentation	2248
16.138.2.1 NTPServer()	2248
16.138.3 Member Function Documentation	2248
16.138.3.1 getPort()	2248
16.138.3.2 isRunning()	2248
16.138.3.3 serverTask()	2248
16.138.3.4 start()	2248
16.138.3.5 stop()	2249
16.139 NTPServer::NtpTimeStamp Class Reference	2249
16.139.1 Detailed Description	2249

16.139.2 Constructor & Destructor Documentation	2249
16.139.2.1 NtpTimeStamp()	2249
16.139.3 Member Function Documentation	2249
16.139.3.1 Now()	2249
16.139.3.2 operator=() [1/2]	2249
16.139.3.3 operator=() [2/2]	2250
16.139.3.4 SetFrom()	2250
16.140 ParsedJsonDataSet Class Reference	2250
16.140.1 Detailed Description	2256
16.140.2 Member Function Documentation	2256
16.140.2.1 Add() [1/12]	2256
16.140.2.2 Add() [2/12]	2256
16.140.2.3 Add() [3/12]	2256
16.140.2.4 Add() [4/12]	2256
16.140.2.5 Add() [5/12]	2257
16.140.2.6 Add() [6/12]	2257
16.140.2.7 Add() [7/12]	2257
16.140.2.8 Add() [8/12]	2257
16.140.2.9 Add() [9/12]	2257
16.140.2.10 Add() [10/12]	2258
16.140.2.11 Add() [11/12]	2258
16.140.2.12 Add() [12/12]	2258
16.140.2.13 AddArrayElement() [1/10]	2258
16.140.2.14 AddArrayElement() [2/10]	2259
16.140.2.15 AddArrayElement() [3/10]	2259
16.140.2.16 AddArrayElement() [4/10]	2259
16.140.2.17 AddArrayElement() [5/10]	2259
16.140.2.18 AddArrayElement() [6/10]	2259
16.140.2.19 AddArrayElement() [7/10]	2259
16.140.2.20 AddArrayElement() [8/10]	2260
16.140.2.21 AddArrayElement() [9/10]	2260
16.140.2.22 AddArrayElement() [10/10]	2260
16.140.2.23 AddArrayStart()	2260
16.140.2.24 AddMyMac()	2260
16.140.2.25 AddNull()	2261
16.140.2.26 AddObjectStart()	2261
16.140.2.27 CopyCurrentString() [1/2]	2261
16.140.2.28 CopyCurrentString() [2/2]	2261
16.140.2.29 CopyObject()	2261
16.140.2.30 CurrentBool()	2261
16.140.2.31 CurrentName()	2262
16.140.2.32 CurrentNumber()	2262

16.140.2.33 CurrentString()	2262
16.140.2.34 FindAndCopyFullNameString() [1/2]	2262
16.140.2.35 FindAndCopyFullNameString() [2/2]	2263
16.140.2.36 FindAndCopyGlobalString() [1/2]	2264
16.140.2.37 FindAndCopyGlobalString() [2/2]	2264
16.140.2.38 FindAndCopyString() [1/2]	2265
16.140.2.39 FindAndCopyString() [2/2]	2265
16.140.2.40 FindBoolean()	2265
16.140.2.41 FindBooleanInCurentObject()	2266
16.140.2.42 FindElementAfterName()	2266
16.140.2.43 FindElementAfterNameInCurrentArray()	2266
16.140.2.44 FindElementAfterNameInCurrentObject()	2266
16.140.2.45 FindFullAtName()	2267
16.140.2.46 FindFullName()	2267
16.140.2.47 FindFullNameBoolean()	2267
16.140.2.48 FindFullNameNumber()	2268
16.140.2.49 FindFullNamePermissiveBoolean()	2268
16.140.2.50 FindFullNameString()	2268
16.140.2.51 FindGlobalBoolean()	2269
16.140.2.52 FindGlobalElementAfterName()	2269
16.140.2.53 FindGlobalNumber()	2269
16.140.2.54 FindGlobalObject()	2270
16.140.2.55 FindGlobalPermissiveBoolean()	2270
16.140.2.56 FindGlobalString()	2270
16.140.2.57 FindNumber()	2270
16.140.2.58 FindNumberInCurentObject()	2271
16.140.2.59 FindObject()	2271
16.140.2.60 FindObjectInCurentObject()	2271
16.140.2.61 FindPermissiveBoolean()	2272
16.140.2.62 FindPermissiveBooleanInCurentObject()	2272
16.140.2.63 FindString()	2272
16.140.2.64 FindStringInCurentObject()	2272
16.140.2.65 GetCurrent()	2273
16.140.2.66 GetFirst()	2273
16.140.2.67 GetNext()	2274
16.140.2.68 GetNextArray()	2274
16.140.2.69 GetNextBoolInCurrentArray()	2274
16.140.2.70 GetNextName()	2275
16.140.2.71 GetNextNameInCurrentArray()	2275
16.140.2.72 GetNextNameInCurrentObject()	2275
16.140.2.73 GetNextNumberInCurrentArray()	2275
16.140.2.74 GetNextObject()	2276

16.140.2.75	GetNextObjectInCurrentArray()	2276
16.140.2.76	GetNextStringInCurrentArray()	2276
16.140.2.77	GetParsePosition()	2277
16.140.2.78	GetPrintSize()	2277
16.140.2.79	GetRawCurrent()	2277
16.140.2.80	name()	2278
16.140.2.81	object()	2278
16.140.2.82	operator>()()	2278
16.140.2.83	operator[]()	2279
16.140.2.84	PermissiveCurrentBool()	2279
16.140.2.85	PrintObject()	2279
16.140.2.86	PrintObjectToBuffer()	2279
16.140.2.87	PrintObjectToFd()	2279
16.140.2.88	PrintObjectToString()	2280
16.140.2.89	ReadFrom()	2280
16.140.2.90	ResetPosition()	2280
16.140.2.91	SetParsePosition()	2280
16.140.2.92	SkipCurrentValue()	2281
16.140.2.93	WriteData()	2281
16.141	ParsedURI Class Reference	2281
16.141.1	Detailed Description	2282
16.141.2	Constructor & Destructor Documentation	2282
16.141.2.1	ParsedURI()	2282
16.141.3	Member Function Documentation	2282
16.141.3.1	GetAddr()	2282
16.141.3.2	GetHost()	2283
16.141.3.3	GetPath()	2283
16.141.3.4	GetPort()	2283
16.141.3.5	IsSecure()	2283
16.141.3.6	NewUri()	2283
16.141.3.7	valid()	2283
16.142	PartitionEntry Struct Reference	2284
16.142.1	Detailed Description	2284
16.143	PartitionTable Struct Reference	2284
16.143.1	Detailed Description	2285
16.144	fifo_buffer_storage::PeekIterator Class Reference	2285
16.144.1	Detailed Description	2285
16.145	PinIO Class Reference	2285
16.145.1	Detailed Description	2288
16.145.2	Member Enumeration Documentation	2288
16.145.2.1	pin_fn_t [1/3]	2288
16.145.2.2	pin_fn_t [2/3]	2289

16.145.2.3 pin_fn_t [3/3]	2290
16.145.2.4 pull_t	2291
16.145.3 Constructor & Destructor Documentation	2291
16.145.3.1 PinIO() [1/2]	2291
16.145.3.2 PinIO() [2/2]	2291
16.145.4 Member Function Documentation	2292
16.145.4.1 function()	2292
16.145.4.2 getFn()	2292
16.145.4.3 multdrv()	2292
16.145.4.4 operator bool()	2292
16.145.4.5 operator"!()	2292
16.145.4.6 operator=() [1/2]	2292
16.145.4.7 operator=() [2/2]	2293
16.145.4.8 PullDown()	2293
16.145.4.9 PullUp()	2293
16.145.4.10 read()	2293
16.145.4.11 readBack()	2294
16.145.4.12 setFn()	2294
16.145.4.13 setHighStrength()	2294
16.145.4.14 setPull()	2294
16.145.4.15 tgl()	2294
16.145.4.16 toggle()	2294
16.146 PinIOArray2 Class Reference	2295
16.146.1 Detailed Description	2295
16.147 PinIOArrayJ1 Class Reference	2295
16.147.1 Detailed Description	2295
16.148 PinIOJ1Array Class Reference	2295
16.148.1 Detailed Description	2295
16.149 PinVector< n > Class Template Reference	2296
16.149.1 Detailed Description	2296
16.149.2 Constructor & Destructor Documentation	2296
16.149.2.1 PinVector()	2296
16.149.3 Member Function Documentation	2297
16.149.3.1 operator=()	2297
16.150 pool_buffer Struct Reference	2297
16.150.1 Detailed Description	2297
16.151 PsudeoHeader Struct Reference	2298
16.151.1 Detailed Description	2298
16.152 MQTT::PublishRequest Struct Reference	2298
16.152.1 Detailed Description	2298
16.153 QSPI_Record Struct Reference	2298
16.153.1 Detailed Description	2299

16.154 fifo_buffer_storage::ReadBufPtr Class Reference	2299
16.154.1 Detailed Description	2299
16.155 SAI_cfg_t Struct Reference	2299
16.155.1 Detailed Description	2299
16.155.2 Member Data Documentation	2300
16.155.2.1 rx	2300
16.155.2.2 rxClkFreq	2300
16.155.2.3 rxTxSync	2300
16.155.2.4 sai	2300
16.155.2.5 tx	2300
16.155.2.6 txClkFreq	2300
16.156 SAI_rtx_cfg_t Struct Reference	2300
16.156.1 Detailed Description	2301
16.156.2 Member Data Documentation	2301
16.156.2.1 bitOrder	2301
16.156.2.2 bitsPerWord	2301
16.156.2.3 clkCh	2302
16.156.2.4 clkGate	2302
16.156.2.5 clkOut	2302
16.156.2.6 clkPol	2302
16.156.2.7 clkSrc	2302
16.156.2.8 dataPacking	2302
16.156.2.9 depletionBehavior	2302
16.156.2.10 enable	2302
16.156.2.11 lineIdleState	2302
16.156.2.12 startCond	2302
16.156.2.13 startDly	2302
16.156.2.14 syncCh	2303
16.156.2.15 syncEdge	2303
16.156.2.16 syncLen	2303
16.156.2.17 syncOut	2303
16.156.2.18 unifyDataStream	2303
16.156.2.19 wordMask	2303
16.156.2.20 wordsPerFrame	2303
16.157 SAICtx_t Class Reference	2303
16.157.1 Detailed Description	2304
16.157.2 Member Enumeration Documentation	2305
16.157.2.1 ctxState_t	2305
16.157.3 Member Function Documentation	2305
16.157.3.1 clrStickyRxError()	2305
16.157.3.2 clrStickyTxError()	2306
16.157.3.3 dump()	2306

16.157.3.4	getCurrentConfig()	2306
16.157.3.5	GetMaxRxBuffers()	2306
16.157.3.6	GetMaxTxBuffers()	2306
16.157.3.7	getState()	2307
16.157.3.8	getStickyRxError()	2307
16.157.3.9	getStickyTxError()	2307
16.157.3.10	Init()	2307
16.157.3.11	ReadyReceiveBuffer()	2308
16.157.3.12	RegisterRxBufferCB()	2308
16.157.3.13	RegisterTxBufferCB()	2309
16.157.3.14	Shutdown()	2309
16.157.3.15	TransmitBuffer()	2309
16.158	SEMC_cfg_t Struct Reference	2310
16.158.1	Detailed Description	2311
16.158.2	Member Data Documentation	2312
16.158.2.1	[union]	2312
16.158.2.2	[union]	2312
16.158.2.3	[union]	2312
16.158.2.4	addr_hold	2313
16.158.2.5	addr_setup	2313
16.158.2.6	addr_wr_hold	2313
16.158.2.7	advHoldLvl	2313
16.158.2.8	advPol	2313
16.158.2.9	burstLen	2313
16.158.2.10	busWidth	2313
16.158.2.11	clkMode	2313
16.158.2.12	clkPin	2313
16.158.2.13	colAddrWidth	2313
16.158.2.14	CR0	2313
16.158.2.15	CR1	2313
16.158.2.16	CR2	2314
16.158.2.17	cs_hold_min	2314
16.158.2.18	cs_min_intv	2314
16.158.2.19	cs_setup	2314
16.158.2.20	csPin	2314
16.158.2.21	latency	2314
16.158.2.22	muxMode	2314
16.158.2.23	rd_cycle	2314
16.158.2.24	rd_en_hi	2314
16.158.2.25	rd_en_lo	2314
16.158.2.26	rd_hold	2314
16.158.2.27	RESERVED0	2314

16.158.2.28 RESERVED1	2315
16.158.2.29 turnaround	2315
16.158.2.30 wr_dat_hold	2315
16.158.2.31 wr_dat_setup	2315
16.158.2.32 wr_en_hi	2315
16.158.2.33 wr_en_lo	2315
16.159 SerialConfigServlet Class Reference	2315
16.159.1 Detailed Description	2315
16.160 SerialRecord Struct Reference	2315
16.160.1 Detailed Description	2316
16.160.2 Member Function Documentation	2316
16.160.2.1 AssignUartNumber()	2316
16.160.2.2 CloseListenPort()	2316
16.160.2.3 GetCurrentChannelStatus()	2316
16.160.2.4 MakeTcpConnection()	2316
16.160.2.5 MakeUdpConnection()	2316
16.160.2.6 OkToListen()	2316
16.160.2.7 OpenListenPort()	2317
16.160.2.8 OpenSerialPort()	2317
16.160.2.9 ProcessAccept()	2317
16.160.2.10 ProcessListenError()	2317
16.160.2.11 ProcessNetworkError()	2317
16.160.2.12 ProcessReadNetworkData()	2317
16.160.2.13 ProcessSerialError()	2317
16.160.2.14 ProcessSpecialFrameTCPReadSerialData()	2317
16.160.2.15 ProcessSpecialFrameWriteNetworkData()	2317
16.160.2.16 ProcessSpecialFrameWriteTimeout()	2317
16.160.2.17 ProcessTCPReadSerialData()	2318
16.160.2.18 ProcessTimeouts()	2318
16.160.2.19 ProcessWriteNetworkData()	2318
16.160.2.20 ProcessWriteSerialData()	2318
16.161 DHCP::Server Class Reference	2318
16.161.1 Detailed Description	2318
16.162 SimpleBufferObject Class Reference	2318
16.162.1 Detailed Description	2318
16.163 DHCP::SingleAllocator Class Reference	2319
16.163.1 Detailed Description	2319
16.164 SMPoolPtr Class Reference	2319
16.164.1 Detailed Description	2319
16.164.2 Constructor & Destructor Documentation	2319
16.164.2.1 SMPoolPtr()	2319
16.165 SPI_QSPI Class Reference	2319

16.165.1 Detailed Description	2321
16.165.2 Constructor & Destructor Documentation	2321
16.165.2.1 SPI_QSPI() [1/2]	2321
16.165.2.2 SPI_QSPI() [2/2]	2321
16.165.3 Member Function Documentation	2322
16.165.3.1 Init()	2322
16.165.3.2 Rx()	2323
16.165.3.3 SetBusSpeed()	2323
16.165.3.4 SetCS()	2323
16.165.3.5 Start()	2324
16.165.3.6 Tx()	2324
16.166 SPI_USART Class Reference	2325
16.166.1 Detailed Description	2326
16.166.2 Constructor & Destructor Documentation	2326
16.166.2.1 SPI_USART() [1/2]	2326
16.166.2.2 SPI_USART() [2/2]	2327
16.166.3 Member Function Documentation	2327
16.166.3.1 Init()	2327
16.166.3.2 Rx()	2328
16.166.3.3 SetBusSpeed()	2328
16.166.3.4 SetCS()	2328
16.166.3.5 Start()	2329
16.166.3.6 Tx()	2329
16.167 spiDriverStruct Struct Reference	2330
16.167.1 Detailed Description	2330
16.168 SPIModule Class Reference	2330
16.168.1 Detailed Description	2332
16.168.2 Constructor & Destructor Documentation	2332
16.168.2.1 SPIModule() [1/5]	2332
16.168.2.2 SPIModule() [2/5]	2333
16.168.2.3 SPIModule() [3/5]	2333
16.168.2.4 SPIModule() [4/5]	2333
16.168.2.5 SPIModule() [5/5]	2334
16.168.3 Member Function Documentation	2335
16.168.3.1 ClrSem() [1/2]	2335
16.168.3.2 ClrSem() [2/2]	2335
16.168.3.3 Done() [1/4]	2335
16.168.3.4 Done() [2/4]	2335
16.168.3.5 Done() [3/4]	2335
16.168.3.6 Done() [4/4]	2335
16.168.3.7 GetActualBaudrate() [1/2]	2336
16.168.3.8 GetActualBaudrate() [2/2]	2336

16.168.3.9 GetSem() [1/2]	2336
16.168.3.10 GetSem() [2/2]	2336
16.168.3.11 Init() [1/3]	2336
16.168.3.12 Init() [2/3]	2337
16.168.3.13 Init() [3/3]	2338
16.168.3.14 RegisterSem() [1/2]	2338
16.168.3.15 RegisterSem() [2/2]	2338
16.168.3.16 Rx() [1/2]	2339
16.168.3.17 Rx() [2/2]	2339
16.168.3.18 SetBusSpeed() [1/2]	2339
16.168.3.19 SetBusSpeed() [2/2]	2340
16.168.3.20 SetCS() [1/2]	2340
16.168.3.21 SetCS() [2/2]	2340
16.168.3.22 Start() [1/2]	2341
16.168.3.23 Start() [2/2]	2341
16.168.3.24 Tx() [1/2]	2342
16.168.3.25 Tx() [2/2]	2342
16.169 SRAM_Timing_Spec_t Struct Reference	2342
16.169.1 Detailed Description	2343
16.170 SSC_cfg_t Struct Reference	2343
16.170.1 Detailed Description	2343
16.170.2 Member Data Documentation	2343
16.170.2.1 clkDiv	2343
16.170.2.2 rx	2344
16.170.2.3 tx	2344
16.171 SSC_rtx_cfg_t Struct Reference	2344
16.171.1 Detailed Description	2344
16.171.2 Member Data Documentation	2344
16.171.2.1 bitOrder	2344
16.171.2.2 bitsPerWord	2344
16.171.2.3 clkGate	2345
16.171.2.4 clkOut	2345
16.171.2.5 clkSrc	2345
16.171.2.6 dataValid	2345
16.171.2.7 depletionBehavior	2345
16.171.2.8 enable	2345
16.171.2.9 linIdleState	2345
16.171.2.10 period	2345
16.171.2.11 startCond	2345
16.171.2.12 startDly	2345
16.171.2.13 syncDataEnabled	2345
16.171.2.14 syncEdge	2345

16.171.2.15 syncLen	2346
16.171.2.16 syncOut	2346
16.171.2.17 wordsPerFrame	2346
16.172 SSCCtx_t Class Reference	2346
16.172.1 Detailed Description	2346
16.172.2 Member Function Documentation	2346
16.172.2.1 getCurrentConfig()	2346
16.172.2.2 getState()	2347
16.172.2.3 Init()	2347
16.172.2.4 ReadyReceiveBuffer()	2347
16.172.2.5 TransmitBuffer()	2347
16.173 Stopwatch Class Reference	2348
16.173.1 Detailed Description	2348
16.173.1.1 Thread Safety	2348
16.173.1.2 Timer Allocation	2349
16.173.2 Constructor & Destructor Documentation	2349
16.173.2.1 Stopwatch()	2349
16.173.3 Member Function Documentation	2349
16.173.3.1 Clear()	2349
16.173.3.2 Convert()	2350
16.173.3.3 CountResolution()	2351
16.173.3.4 GetTime()	2352
16.173.3.5 Start()	2353
16.173.3.6 Stop()	2354
16.174 StringBufferObject Class Reference	2355
16.174.1 Detailed Description	2355
16.175 MQTT::SubHandler Class Reference	2355
16.175.1 Detailed Description	2357
16.175.2 Constructor & Destructor Documentation	2357
16.175.2.1 SubHandler()	2357
16.175.3 Member Function Documentation	2358
16.175.3.1 ParseFd()	2358
16.175.3.2 RenderToBuffer()	2358
16.175.3.3 RenderToFd()	2358
16.175.3.4 SubscribeEvent()	2359
16.175.3.5 TestOrSetValue()	2360
16.176 MQTT::SubscribeFilter Struct Reference	2360
16.176.1 Detailed Description	2360
16.177 MQTT::SubscribeFilter_NbStr Struct Reference	2360
16.177.1 Detailed Description	2360
16.178 MQTT::SubscribeRequest Struct Reference	2361
16.178.1 Detailed Description	2361

16.179 TFTP_Server Class Reference	2361
16.179.1 Detailed Description	2361
16.179.2 Constructor & Destructor Documentation	2361
16.179.2.1 TFTP_Server()	2361
16.179.3 Member Function Documentation	2361
16.179.3.1 Add()	2361
16.180 TFTP_Server_source_File Class Reference	2362
16.180.1 Detailed Description	2362
16.181 TimeZoneRecord Struct Reference	2362
16.181.1 Detailed Description	2362
16.181.2 Member Data Documentation	2363
16.181.2.1 bUsCanada	2363
16.181.2.2 Description	2363
16.181.2.3 Name	2364
16.181.2.4 Posix	2364
16.181.2.5 Short_name	2364
16.182 MQTT::TopicHandler Struct Reference	2364
16.182.1 Detailed Description	2365
16.183 TwoWire Class Reference	2365
16.183.1 Detailed Description	2366
16.183.2 Constructor & Destructor Documentation	2366
16.183.2.1 TwoWire() [1/2]	2366
16.183.2.2 TwoWire() [2/2]	2366
16.183.3 Member Function Documentation	2367
16.183.3.1 available()	2367
16.183.3.2 begin()	2367
16.183.3.3 beginTransmission()	2367
16.183.3.4 endTransmission()	2367
16.183.3.5 onReceive()	2368
16.183.3.6 onRequest()	2368
16.183.3.7 ping()	2368
16.183.3.8 read() [1/2]	2368
16.183.3.9 read() [2/2]	2368
16.183.3.10 readReg8()	2369
16.183.3.11 readRegN()	2369
16.183.3.12 requestFrom()	2369
16.183.3.13 setClock()	2370
16.183.3.14 setWireTimeout()	2370
16.183.3.15 write() [1/4]	2370
16.183.3.16 write() [2/4]	2370
16.183.3.17 write() [3/4]	2371
16.183.3.18 write() [4/4]	2371

16.183.3.19 writeReg8()	2371
16.183.3.20 writeRegN()	2372
16.184 UDPPacket Class Reference	2372
16.184.1 Detailed Description	2374
16.184.1.1 Transmission Patterns	2374
16.184.1.2 Reception Pattern	2374
16.184.2 Constructor & Destructor Documentation	2374
16.184.2.1 UDPPacket() [1/5]	2374
16.184.2.2 UDPPacket() [2/5]	2377
16.184.2.3 UDPPacket() [3/5]	2380
16.184.2.4 UDPPacket() [4/5]	2382
16.184.2.5 UDPPacket() [5/5]	2384
16.184.2.6 ~UDPPacket()	2386
16.184.3 Member Function Documentation	2389
16.184.3.1 AddData() [1/2]	2389
16.184.3.2 AddData() [2/2]	2390
16.184.3.3 AddDataByte()	2390
16.184.3.4 AddDataWord()	2391
16.184.3.5 blsIPv6()	2391
16.184.3.6 GetDataBuffer()	2392
16.184.3.7 GetDataSize()	2392
16.184.3.8 GetDestinationAddress()	2393
16.184.3.9 GetDestinationPort()	2394
16.184.3.10 GetDSCP()	2394
16.184.3.11 GetMacSource()	2395
16.184.3.12 GetPacketId()	2397
16.184.3.13 GetSourceAddress()	2398
16.184.3.14 GetSourcePort()	2398
16.184.3.15 Receive()	2400
16.184.3.16 ResetData()	2402
16.184.3.17 Send()	2403
16.184.3.18 SendAndKeep()	2406
16.184.3.19 SendAndKeepVialfAddr()	2408
16.184.3.20 SendAndKeepVialInterfaceNum()	2409
16.184.3.21 SendVialfAddr()	2412
16.184.3.22 SendVialInterfaceNum()	2412
16.184.3.23 SetDataSize()	2415
16.184.3.24 SetDestinationPort()	2415
16.184.3.25 SetDSCP()	2416
16.184.3.26 SetSourcePort()	2417
16.184.3.27 Validate()	2418
16.185 UDPPKT Struct Reference	2419

16.185.1 Detailed Description	2419
16.185.2 Member Data Documentation	2419
16.185.2.1 UdpCSum	2419
16.186 UniqueIdentifier Class Reference	2420
16.186.1 Detailed Description	2420
16.186.2 Member Function Documentation	2420
16.186.2.1 GetBuffer() [1/2]	2420
16.186.2.2 GetBuffer() [2/2]	2420
16.186.2.3 GetUniqueIdentifier() [1/2]	2421
16.186.2.4 GetUniqueIdentifier() [2/2]	2421
16.187 UserAuthManager Class Reference	2421
16.187.1 Detailed Description	2422
16.187.2 Member Function Documentation	2422
16.187.2.1 AddUserAuth()	2422
16.187.2.2 CheckUserAuth() [1/2]	2422
16.187.2.3 CheckUserAuth() [2/2]	2423
16.187.2.4 CheckUserAuthLevel()	2423
16.187.2.5 ClrUserAuthLevel()	2424
16.187.2.6 Init()	2424
16.187.2.7 RemoveUserAuth()	2424
16.187.2.8 SetUserAuthLevel()	2425
16.187.2.9 UpdateUserAuth()	2425
16.187.2.10 UserExists()	2425
16.188 UserAuthRecord Struct Reference	2426
16.188.1 Detailed Description	2426
16.188.2 Member Data Documentation	2426
16.188.2.1 m_authLevel	2426
16.189 WebConfigServlet Class Reference	2426
16.189.1 Detailed Description	2426
16.190 NB::WebSocket Class Reference	2426
16.190.1 Detailed Description	2428
16.190.2 Member Function Documentation	2428
16.190.2.1 cloption()	2428
16.190.2.2 Connect() [1/4]	2430
16.190.3 Connection Process	2431
16.190.4 Parameters	2431
16.190.5 Return Value	2431
16.190.6 Connection Failures	2431
16.190.7 Usage Notes	2432
16.190.7.1 Connect() [2/4]	2435
16.190.8 Parameters	2435
16.190.9 Return Value	2435

16.190.9.1 Connect() [3 / 4]	2436
16.190.10 Connection Process	2436
16.190.11 Parameters	2436
16.190.12 Return Value	2436
16.190.13 Advantages of IP Address Connection	2436
16.190.13.1 Connect() [4 / 4]	2439
16.190.14 Parameters	2439
16.190.15 Return Value	2439
16.190.15.1 DumpSock()	2439
16.190.16 Displayed Information	2439
16.190.17 Usage Scenarios	2440
16.190.17.1 DumpSockets()	2442
16.190.18 Displayed Information	2442
16.190.19 Usage Scenarios	2442
16.190.19.1 Flush()	2445
16.190.19.2 GetFlushTime()	2448
16.190.20 Description	2448
16.190.21 Usage Notes	2448
16.190.22 Flush Timing Strategies	2448
16.190.23 Thread Safety	2448
16.190.24 See Also	2448
16.190.24.1 GetFreeWebSocketCount()	2450
16.190.25 Return Value	2450
16.190.26 Usage Guidelines	2450
16.190.26.1 getoption()	2453
16.190.26.2 GetPingReplyTick()	2457
16.190.26.3 GetState()	2460
16.190.26.4 GetWebSocketRecord()	2463
16.190.26.5 GetWriteSpace()	2464
16.190.26.6 Ping()	2467
16.190.26.7 Pong()	2470
16.190.26.8 promote_tcp_ws()	2473
16.190.26.9 ReadData()	2475
16.190.26.10 RunSkippedCallBack()	2479
16.190.27 Description	2479
16.190.28 Usage Notes	2479
16.190.29 Thread Safety	2480
16.190.30 See Also	2480
16.190.30.1 setoption()	2481
16.190.30.2 WaitForPingReply()	2483
16.190.30.3 WriteData()	2486
16.190.30.4 ws_clroption()	2491

16.190.30.5 ws_externalclose()	2492
16.190.30.6 ws_flush()	2494
16.190.30.7 ws_getoption()	2496
16.190.30.8 ws_read()	2498
16.190.30.9 ws_read_notify()	2501
16.190.31 Description	2501
16.190.32 Parameters	2501
16.190.33 Usage Notes	2501
16.190.34 WebSocket Frame Handling	2501
16.190.35 Thread Safety	2501
16.190.36 See Also	2502
16.190.36.1 ws_readwto()	2503
16.190.36.2 ws_setoption()	2506
16.190.36.3 ws_write()	2507
16.190.36.4 ws_write_notify()	2511
16.190.37 Description	2511
16.190.38 Parameters	2511
16.190.39 Usage Notes	2511
16.190.40 Flow Control	2512
16.190.41 Thread Safety	2512
16.190.42 See Also	2512
16.190.43 Member Data Documentation	2513
16.190.43.1 m_fd_tcp	2513
16.190.43.2 m_fd_ws	2515
16.190.43.3 s_openSocketCount	2517
16.190.43.4 socket_crit	2519
16.190.43.5 WSSockStructs	2522
16.191 wifi_init Struct Reference	2524
16.191.1 Detailed Description	2524
16.192 WifiInterface Class Reference	2524
16.192.1 Detailed Description	2524
16.193 WireIntf Class Reference	2525
16.193.1 Detailed Description	2525
16.193.2 Member Function Documentation	2525
16.193.2.1 available()	2525
16.193.2.2 beginTransmission()	2526
16.193.2.3 endTransmission()	2526
16.193.2.4 flush()	2526
16.193.2.5 read()	2526
16.193.2.6 requestFrom()	2527
16.193.2.7 write() [1/3]	2527
16.193.2.8 write() [2/3]	2527

16.193.2.9 write() [3/3]	2528
16.194 WM8904 Class Reference	2528
16.194.1 Detailed Description	2529
16.194.2 Constructor & Destructor Documentation	2529
16.194.2.1 WM8904()	2529
16.194.3 Member Function Documentation	2529
16.194.3.1 GetMicGain()	2529
16.194.3.2 GetVolume()	2529
16.194.3.3 Init()	2530
16.194.3.4 Mute()	2530
16.194.3.5 MuteMic()	2530
16.194.3.6 ReadReg()	2530
16.194.3.7 ReadyReceiveBuffer()	2531
16.194.3.8 SendCmd()	2531
16.194.3.9 SendCmdList()	2531
16.194.3.10 SetMicGain()	2531
16.194.3.11 SetVolume()	2532
16.194.3.12 TransmitBuffer()	2532
16.194.3.13 UpdateCmd()	2532
16.194.3.14 WriteReg()	2532
16.195 fifo_buffer_storage::WriteBufPtr Class Reference	2533
16.195.1 Detailed Description	2533
Index	2535

Chapter 1

NetBurner 3.5.6

1.1 Welcome to NetBurner Version 3!

Last updated at October 31 2025 06:49:49

This Developers Guide details the capabilities of your NetBurner Development Software and Tools. If you are viewing this document from your development tools installation, the latest version is available at www.netburner.com. There is also a pdf version of this document available at the [PDF Version](#) link above.

Note

Version 3 is used as a development platform on all ARM based devices, as well as MOD5441x, NANO54415 and SB800EX devices. If you have an earlier ColdFire based device please refer to the version 2 development tools.

1.2 Who Is This Guide For?

NetBurner devices come in two forms:

- Devices pre-programmed with factory applications that can be used without any custom programming on the device itself. Serial to Ethernet devices are a good example of this.
- If you wish to customize any device, we provide development tools that enable you to create custom applications.

This guide is intended for software developers to create custom applications to run on NetBurner devices. If you are using a device that does not require custom programming, please refer to that device's product page for documentation and any appropriate software utilities.

1.3 Recommended Starting Point for New Developers

Your NetBurner development system is pre-programmed with a factory application. If connected to a network with Internet access and a DHCP server, you can open a web browser, go to <http://discover.netburner.com>, and access your device immediately: both the application web page and configuration interface are linked from the Discover page.

If you are connected directly, such as a laptop with a direct Ethernet connection, or on a network without a DHCP server and Internet access, please see the procedure below:

- Configure the development kit device/hardware (if necessary) as described in the [Quick Start Guide](#).
- Connect both your development computer and your NetBurner device to a network with Internet access. Internet access is not required for development, but will make the initial configuration of the device easier.
- If using a development kit with a USB interface, connect it from the development board to your computer. It will provide both power and a console/debug serial port. To view status and debug information, run a serial terminal program such as the included MTTY, although any serial terminal application should work.
- Your NetBurner device will be running a pre-programmed factory application. Open a web browser and type "discover.netburner.com" to identify your device. Links on the web page are available to go directly to the device's web page and configuration page.

Note

If you do not have an Internet connection, you can use a utility such as localdiscover. Please refer to the Device Discovery and Configuration section of this guide.

- Refer to the [NBEclipse](#) Getting Started Guide section of this document to learn how to create, build and download applications. If you prefer to use command line tools, they are available as well.
- Select one of the examples from the "nburn\examples" folder. Create a project and download per the NBEclipse Guide.
- Refer to the Programmers Guide and NetBurner API documents for additional information.

Note

NBEclipse provides auto-complete functionality by pressing cntl-space. This makes accessing the API functions much easier.

1.4 Documentation Overview

1.4.1 The NetBurner Application Programming Interface (API)

Description of the functions and capabilities in the NetBurner Library API. These functions include:

- Real-Time Operating System
- TCP/IP Stack
- Peripherals, such as SPI, [I2C](#), 1-Wire, UARTs, ADC, DAC, Timers, etc
- Description of objects, classes and structures
- Function error codes

1.4.2 Device Discovery and Configuration

How to locate your NetBurner device as well as configure system parameters.

1.4.3 The NBEclipse Getting Started Guide

Describes how to create projects and build applications.

1.4.4 Migration Guides

Information for those moving from tools revision 2.x to 3.x.

1.4.5 Example Programs

Well over 100 example programs on everything from web servers, TCP, UDP, Configuration to hardware peripherals. The examples are described in this guide, and located in the "nburn\examples" folder of your installation.

Note

NBEclipse provides an easy example import feature when you create a new project.

Interesting Starting Examples:

- ShowInterfaces: Displays all available network interface information on a web page and serial port
- SimpleHtml: Minimal web server
- IPADDR-IP Address: IPADDR object operations for assignment and display
- PlatformSpecific folder: Unique peripherals for each hardware platform, such as timers, ADC, DAC and unique I2C and SPI peripherals
- RTOS folder: How to use the various system RTOS capabilities

1.4.6 Programmers Guide

Textbook style guide on how to use the various features of your device and networking.

- Device configuration and the Configuration Server
- HTML Processing and Web Server
- NetBurner Real-Time Operating System (NBRTOS)
- Network Protocols such as TCP and UDP
- Security using SSL/TLS
- Embedded Flash File System (EFFS)

1.4.7 Platform References

The Platform References of this guide provide hardware information such as memory maps and device recovery should your application crash in a way that is not easily reset. Schematics for your NetBurner devices are located in the `\nburn\docs\NetBurner\platform` folder.

1.4.8 System Diagnostics

Information on how to enable and use built-in device diagnostics as well as customizing to add your own.

1.4.9 Utilities

Tools and Utilities that are run on a client computer, such as discover, configuration, and application updates.

1.4.10 Production and Deployment

Creating a great application and end product is just part of what needs to be done for a successful product. NetBurner provides deployment information and utilities that you can use in your production environment to program and configure modules in volume, as well as offer to your end customers.

1.4.11 Location of Original Factory Program

If at any time you wish to restore the original factory program that was running on your device, the pre-built application image is located in the `\nburn\platform\<your platform>\original` folder. For example, `\nburn\platform\MODM7AE70\original`. This path assumes a default install to a folder named `nburn`.

1.4.12 Experienced NetBurner Customers

A significant change you will notice from NBRTOS version 2 is how the device configuration is accomplished using a web interface. Please refer to the Migration Guide for a detailed description of the differences between release version 3 and previous releases.

Chapter 2

Build System

2.1 Introduction

The Build System section contains information on subjects such as the GCC Compiler and Build Environment.

2.2 GCC Compiler Flags

Compiler flags can be added or modified in both NBEclipse and command line makefiles.

2.2.1 Generic

Language	Mode	Flag	Description
C/C++	R, D	-gdwarf-2	Adds dwarf-2 debug symbols to the binary. This makes it easier to resolve symbols in gdb.
C/C++	R, D	-Wall	Enables all warnings.
C/C++	R, D	-Werror=return-type	Treats warnings as errors for functions that are missing a return statement.
C/C++	R, D	-Wno-unused	Suppresses warnings for unused variables.
C/C++	R, D	-falign-functions=4	Aligns functions to 4-byte boundaries for improved performance.
C/C++	R	-O2	Enables level 2 optimization for improved performance.
C/C++	D	-O0	Disables optimization for debugging.
C/C++	R, D	-D PLATFORM_↵ NAME=\$(PLATFORM_NAME)	Defines the PLATFORM_NAME macro with the value of the PLATFORM_NAME environment variable.
C/C++	R, D	\$(NBINCLUDE)	Includes the specified directories in the search path for header files.
C	R, D	-std=gnu17	Sets the C standard to GNU C17.

Language	Mode	Flag	Description
C	R, D	<code>-ffunction-sections</code>	Places each function in a separate section to enable linker optimizations.
C	R, D	<code>-fdata-sections</code>	Places each data object in a separate section to enable linker optimizations.
C++	R, D	<code>-std=gnu++17</code>	Sets the C++ standard to GNU C++17.
C++	R, D	<code>-fno-rtti</code>	Disables runtime type information (RTTI) for improved performance.
C++	R, D	<code>-fno-exceptions</code>	Disables C++ exceptions for improved performance.
C++	R, D	<code>-Wno-register</code>	Suppresses warnings for unused register variables.
C++	R, D	<code>-ffunction-sections</code>	Places each function in a separate section to enable linker optimizations.
C++	R, D	<code>-fdata-sections</code>	Places each data object in a separate section to enable linker optimizations.
Assembler	R, D	<code>\$(CPUFLAG)</code>	Specifies the CPU architecture flag.

2.2.2 Architecture-Specific Compiler Flags

2.2.2.1 ARM Cortex-M7

Language	Mode	Flag	Description
C/C++	R, D	<code>-fasynchronous-unwind-tables</code>	Enables asynchronous unwind tables
C/C++	R, D	<code>-D_C11_SOURCE</code>	Defines the <code>_C11_SOURCE</code> macro.
C/C++	R, D	<code>-DCORTEX_M7</code>	Defines the <code>CORTEX_M7</code> macro.

2.2.2.2 ARM Cortex-M0+

Language	Mode	Flag	Description
C/C++	R, D	<code>-fasynchronous-unwind-tables</code>	Enables asynchronous unwind tables
C/C++	D	<code>-fomit-frame-pointer</code>	Omit frame pointer for improved performance.
C/C++	R, D	<code>-D_C11_SOURCE</code>	Defines the <code>_C11_SOURCE</code> macro.
C/C++	R, D	<code>-DCORTEX_M0PLUS</code>	Defines the <code>CORTEX_M0PLUS</code> macro.

2.2.2.3 ColdFire

Language	Mode	Flag	Description
C/C++	R, D	<code>-fno-omit-frame-pointer</code>	Do not omit frame pointer for improved debugging
C/C++	R, D	<code>-fasynchronous-unwind-tables</code>	Enables asynchronous unwind tables
C/C++	R, D	<code>-DCOLDFIRE</code>	Defines the <code>COLDFIRE</code> macro
Assembler	D	<code>--defsym __DEBUG=1</code>	Defines the <code>__DEBUG</code> symbol.

2.2.3 CPU-Specific Compiler Flags

2.2.3.1 SAME70

Language	Mode	Flag	Description
C/C++	R, D	<code>-mcpu=cortex-m7</code>	Specifies the CPU architecture.
C/C++	R, D	<code>-mthumb</code>	Specifies the Thumb instruction set.
C/C++	R, D	<code>-mfpv=fpv5-d16</code>	Specifies the floating-point unit.
C/C++	R, D	<code>-mfloat-abi=\$(ABI_OPT)</code>	Specifies the floating-point ABI. Defaults to softfp
C/C++	R, D	<code>-D\$(CPU)</code>	Defines the <code>\$(CPU)</code> macro.
C++	R, D	<code>-fno-use-cxa-atexit</code>	Disables the use of <code>__cxa_atexit</code>
C++	R, D	<code>-falign-fuinctions=4</code>	Aligns functions to 4-byte boundaries for improved performance.

2.2.3.2 MIMXRT11xx, MIMXRT10xx

Language	Mode	Flag	Description
C/C++	R, D	<code>-mcpu=cortex-m7</code>	Specifies the CPU architecture.
C/C++	R, D	<code>-mthumb</code>	Specifies the Thumb instruction set.
C/C++	R, D	<code>-mfpv=fpv5-d16</code>	Specifies the floating-point unit.
C/C++	R, D	<code>-mfloat-abi=\$(ABI_OPT)</code>	Specifies the floating-point ABI. Defaults to softfp
C/C++	R, D	<code>-D\$(CPU)</code>	Defines the <code>\$(CPU)</code> macro.
C++	R, D	<code>-fno-use-cxa-atexit</code>	Disables the use of <code>__cxa_atexit</code>
C++	R, D	<code>-falign-fuinctions=8</code>	Aligns functions to 8-byte boundaries for improved performance.

2.2.3.3 SAMD20

Language	Mode	Flag	Description
C/C++	R, D	<code>-mcpu=cortex-m0plus</code>	Specifies the CPU architecture.
C/C++	R, D	<code>-mthumb</code>	Specifies the Thumb instruction set.
C++	R	<code>-fno-use-cxa-atexit</code>	Disables the use of <code>__cxa_atexit</code>
C++	R	<code>-falign-fuinctions=4</code>	Aligns functions to 4-byte boundaries for improved performance.

2.2.3.4 MCF5441X

Language	Mode	Flag	Description
C/C++	R, D	<code>-D\$(CPU)</code>	Defines the <code>\$(CPU)</code> macro.
C/C++	R, D	<code>-mcpu=54415</code>	Specifies the CPU architecture.

2.2.4 Platform-Specific Compiler Flags

2.2.4.1 MODM7AE70

Language	Mode	Flag	Description
C/C++	R, D	-D__SAME70Q21↔ __	Defines the __SAME70Q21__ macro.
C/C++	R, D	-DMODM7AE70	Defines the MODM7AE70 macro.

2.2.4.2 SBE70LC

Language	Mode	Flag	Description
C/C++	R, D	-D__SAME70Q21↔ __	Defines the __SAME70Q21__ macro.
C/C++	R, D	-DSBE70LC	Defines the SBE70LC macro.

2.2.4.3 SOMRT1061

Language	Mode	Flag	Description
C/C++	R, D	-D__MIMXRT106xx↔ __	Defines the __MIMXRT106xx__ macro.
C/C++	R, D	-D__MIMXRT1061↔ __	Defines the __MIMXRT1061__ macro.
C/C++	R, D	-DSOMRT1061	Defines the SOMRT1061 macro.

2.2.4.4 MOD5441X

Language	Mode	Flag	Description
C/C++	R, D	-DMOD5441X	Defines the MOD5441X macro.

2.2.4.5 NANO54415

Language	Mode	Flag	Description
C/C++	R, D	-DNANO54415	Defines the NANO54415 macro.

2.2.4.6 SB800EX

Language	Mode	Flag	Description
C/C++	R, D	-DSB800EX	Defines the SB800EX macro.

2.3 Linker Flags

2.3.1 Generic

Flag	Description
\$ (CPUFLAG)	Specifies the CPU architecture flag.
-Wl, -n	Disables linking to dyn
-T\$ (NNDK_ROOT) /platform/\$ (PLATFORM) /linker.ld	Specifies the linker script.
-Wl, -R\$ (NNDK_ROOT) /platform/\$ (PLATFORM) /linker.ld	Specifies the linker script.
-Wl, -Map=\$ (@:.elf=.map)	Generates a map file.
-Wl, --cref	Generates a cross-reference table.
-Wl, --gc-sections	Removes unused sections to reduce the size of the binary.
-Wl, --start-group, \$ (NBLIBS) \$ (XTRALIB) -lstdc++ -lc -Wl, --end-group	Specifies the libraries to link.

2.3.2 Architecture-Specific Linker Flags

None

2.3.3 CPU-Specific Linker Flags

2.3.3.1 SAME70

Flag	Description
-nostartfiles	Disables linking to the default startup files.
-mfloat-abi=\$ (ABI_OPT)	Specifies the floating-point ABI. Defaults to softfp
-Wl--thumb-entry=Reset_Handler	Specifies the entry point.
-Wl--thumb-entry=_start_	Specifies the entry point. (SOMRT1061)

2.3.3.2 MIMXRT11xx, MIMXRT10xx

Flag	Description
-nostartfiles	Disables linking to the default startup files.
-mfloat-abi=\$ (ABI_OPT)	Specifies the floating-point ABI. Defaults to softfp
-Wl--thumb-entry=Reset_Handler	Specifies the entry point.

2.3.3.3 SAMD20

Flag	Description
-nostartfiles	Disables linking to the default startup files.

<code>-Wl--thumb-entry=Reset_Handler</code>	Specifies the entry point.

2.3.3.4 MCF5441X

None

2.3.4 Platform-Specific Linker Flags

None

2.4 Pack Binary

2.4.1 Generic

None

2.4.2 Architecture-Specific

None

Executable	I/O	Flag	Description

2.4.3 CPU-Specific

2.4.3.1 MCF5441X

Executable	I/O	Flag	Description
m68k-elf-objcopy	.elf -> .s19	-O srec	Converts the elf file to s19
m68k-elf-objcopy	.elf -> .s19	-Wl,--gc-sections	Removes unused sections to reduce the size of the binary.
m68k-elf-objcopy	.elf -> .s19	--strip-all	Removes all symbols from the binary.
compcode	.s19 -> _APP.s19	-M \$(^↵ :.s19=.map)	Specifies the map file.
compcode	.s19 -> _APP.s19	-R	Print report.
compcode	.s19 -> _APP.s19	-P\$(PLATFORM)	Specifies the platform.
compcode	.s19 -> .bin	-O binary	Converts the s19 file to binary
compcode	.s19 -> .bin	-M \$(^↵ :.s19=.map)	Specifies the map file.
compcode	.s19 -> .bin	-B	Specifies binary output.
compcode	.s19 -> .bin	-P\$(PLATFORM)	Specifies the platform.
compcode	.s19 -> .bin	-R	Print report.

2.4.3.2 SAMD20

Executable	I/O	Flag	Description
nbfspace	.elf -> .bin	-output-target bin	Converts the elf file to binary

2.4.3.3 SAME70

Executable	I/O	Flag	Description
arm-none-eabi-size	.elf -> .bin		Displays the size of the binary.
nbfspace	.elf -> .bin	-output-target bin	Converts the elf file to binary
nbfspace	.elf -> .bin	-S	Show build statistics.
nbfspace	.elf -> .bin	-P \$(PLATFORM)	Specifies the platform.

2.4.3.4 MIMXRT11xx, MIMXRT10xx

Executable	I/O	Flag	Description
nbfspace	.elf -> .bin	-output-target bin	Converts the elf file to binary

2.4.4 Platform-Specific

2.4.4.1 MODM7AE70

Executable	I/O	Flag	Description
nbfspack	.elf -> .bin	-C	Compress program sections
nbfspack	.elf -> .bin	-cflag S:3	.
nbfspack	.elf -> .bin	-cflag S:0	.
nbfspack	.elf -> .bin	-cflag S:1	.

2.4.4.2 SBE70LC

Executable	I/O	Flag	Description
nbfspack	.elf -> .bin	-C	Compress program sections
nbfspack	.elf -> .bin	-cflag S:3	.
nbfspack	.elf -> .bin	-cflag S:0	.
nbfspack	.elf -> .bin	-cflag S:1	.

2.4.4.3 SOMRT1061

None

2.4.4.4 MOD5441X

Executable	I/O	Flag	Description
comPCODE	.s19 -> .bin	0xC0040000 0xC1FC0000	Application address range.

2.4.4.5 NANO54415

Executable	I/O	Flag	Description
comPCODE	.s19 -> .bin	0x04000 0x800000	Application address range.

2.4.4.6 SB800EX

Executable	I/O	Flag	Description
comPCODE	.s19 -> .bin	0x04000 0x800000	Application address range.

Chapter 3

Command Line Tools

3.1 Introduction

As an alternative to NBEclipse, you can use the command line tools to build your NetBurner projects. In fact, all the example programs are built and tested internally with command line tools. You will find a makefile in each example folder, so any of the examples can be used as a demonstration of how to use command line tools. Windows, Mac and Linux are supported. The command line tools can also be used to invoke the NetBurner tools from the development environment of your choice.

3.2 Command Line Summary

Command	Description
make	Build a project using the makefile in the project directory
make -j	Make using multiple cores to speed up the process
make load	Build project and load into device
make -j load	Build project with multiple cores and load into device
make clean	Delete a project's libraries and object files

Note

The command you will use 99% of the time is: "make -j load". If you have build errors that are difficult to track down, doing a single core "make" will cause the build to stop at the first error.

3.3 Command Line Environment Variables

There are a number of environment variables that can be set to effect the command line build when you do a make. For example, you can specify the target device IP address in which to download the application with a make -j load.

On the command line, typing "set <variable>" will display the current value. Typing "set <variable>=<value>" will set the environment variable. For example, "set DEVIP" will display the current target IP address. Typing "set DEVIP=10.1.1.1" will set DEVIP to the specified IP address.

Variable	Description
DEVIP	Causes a make with the "load" parameter to download to the specified IP address
PLATFORM	Specifies the NetBurner platform to build for
NNDK_ROOT	Specifies the location of your NetBurner development tools installation
PATH	Specifies folders to search for system binaries, like <code>make</code> and other compiler tools

For example, to reference a standard NetBurner tools installation on Windows (also see "`\nburn\setenv.bat`" for an example):

```
set NNDK_ROOT=c:/nburn
set PLATFORM=MODM7AE70
set DEVIP=192.168.1.66
set PATH=C:/nburn/gcc/bin;C:/nburn/pcbins;C:/nburn/gcc/msys/1.0/bin;%path%
```

3.4 Create a New Project From an Example

The easiest way to create a new project is to copy an existing one and modify it:

- Open a command prompt at the location where you keep your projects, such as `\NetBurner\Projects`
- Create a folder for your project, lets use HelloWorld as an example
- Move to the HelloWorld folder
- Copy an example, such as "`\nburn\examples\web\SimpleHtml`"
- At this point your project structure will look like:

```
- \NetBurner\Projects
  * HelloWorld
    · html
      · index.html
    · src
      · main.cpp
    · makefile
```

- Run "`make -j`" to build the project. The "-j" options uses multiple cores for the build
- When the build completes, an obj folder is created with the object files and .bin application image file↵
: `\NetBurner\Projects\HelloWorld\obj\release\SimpleHtml.bin` (and also .s19 for SREC platforms).

```
- \NetBurner\Projects
  * HelloWorld
    · html
      · index.html
    · obj
      · release
        · SimpleHtml.bin
    · src
      · main.cpp
    · makefile
```

- To load the application into your device, use the "`make -j load`" command.

There are many files and folders under the obj folder, including the compiled libraries and object files.

Note

Each project has its own build of the libraries and system files. That way if any changes need to be made (which is rare), it can be done on a per-project basis. This will be discussed in the overload section of this document.

We named our project HelloWorld, but are running SimpleHtml at this point. To customize the project:

- Edit the makefile to change the application image file name to HelloWorld
- Edit main.cpp to change the application name to HelloWorld
- Run "make -j load" to build and load the application into the device

3.4.1 Edit the makefile

The example makefile is shown below:

```
NAME = SimpleHtml
```

```
CPP_SRC = src/main.cpp
```

```
CREATEDTARGS += src/htmldata.cpp
```

```
CPP_SRC += src/htmldata.cpp
```

```
src/htmldata.cpp : $(wildcard html/*.*)
```

```
    comphtml html -osrc/htmldata.cpp
```

```
include $(NNDK_ROOT)/make/boilerplate.mk
```

Changing the NAME from SimpleHtml to HelloWorld will create the image file as HelloWorld.bin (and .s19)

The CPP_SRC is a list of all .cpp files to build. In this case there is only one, but if there were more it would look like:

```
CPP_SRC = src/main.cpp \
        += src/file2.cpp \
        += src/file3.cpp \
        += src/file4.cpp
```

If you have .c source files you can add them with C_SRC in the same manor.

The html related lines auto-generate a cpp file from the code in the project's html directory.

The final line calls the boilerplate makefile which includes the platform specific make instructions. While you can drill down into the details, this type of makefile structure makes creating projects much easier since you only need to list the source files of your project.

3.4.2 Edit main.cpp

A fully functional application with the web server enables is actually very few lines of code:

```
#include <init.h>
#include <nbrtos.h>
#include <system.h>

const char *AppName = "Simple HTML Example";

void UserMain(void *pd)
{
    init();
    StartHttp();
    WaitForActiveNetwork();

    iprintf("Web Application: %s\r\nNNDK Revision: %s\r\n", AppName, GetReleaseTag());
    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

This example just initializes the system, starts the web server, prints out some information, and loops forever with a 1 second delay. Changing the AppName to HelloWorld will change the name that shows up in the find and configuration utilities.

3.5 Modifying System Files with Overload

There can be instances in which you need to modify a NetBurner system file. This is accomplished using the Overload feature. NBEclipse creates an overload folder automatically, but with command line builds it must be created manually.

One file that is frequently overridden to modify how NBRTOS works is `predef.h`. However, we suggest using `predef-overload.h` instead for compatibility with future development tools upgrades. `predef-overload.h` will be identical to `predef.h` for the build in which you overload it. The build system will include `predef-overload.h` last, so that if `predef.h` changes in a future release your application will still only overload what you intended.

Note

The overload example is located in `\nburn\examples\Overload`

To overload system files, we add an `overload` folder to our HelloWorld project. Then once you know which file you want to overload, the filename and path in the overload folder must exactly match the `NNDK_ROOT` system file path. For example, if we want to overload the `timezones.cpp` file located in the `\nburn\nbrtos\source` folder:

- `\NetBurner\Projects`
 - HelloWorld
 - * html
 - index.html
 - * obj
 - release
 - SimpleHtml.bin
 - * overload
 - nbrtos
 - source
 - timezones.cpp
 - * src
 - main.cpp
 - * makefile

Now the project's `timezones.cpp` will be used along with any changes made to it.

3.5.1 Important Overload Rules

- The first time a file is added to the overload folder, a "make clean" is required. From that point on, any modifications to that file will automatically handle building all necessary system files.

If a system include folder is overloaded, this folder should be added to your project include paths. Right click on the project and select project properties. Under C/C++ Build->Settings, select GNU C++ Compiler->Includes and add the overload include folder. If utilizing C code, then GNU C Compiler->Includes should also be added.

3.6 GNU Debugger (GDB)

To load an application and start GDB from the command line:

- Build and load a debug build of the application `make loaddebug -j`.
- Start GDB:
 - For ARM devices such as the SOMRT1061, MODM7AE70 and SBE70LC: `arm-eabi-gdb -se obj/debug/<elf.file>`
 - For MOD5441X, NANO54415, and SB800EX platforms: `m68k-elf-gdb -se obj/debug/<elf.file>`
- Connect to the target device with: `target remote <ipaddress>:2159`

To end the GDB session and leave the target running use `detach`. To end the GDB session use `quit`. The GDB command reference is located here: <http://www.gnu.org/software/gdb/documentation/>.

3.6.1 Example GDB Session

An GDB session running the SimpleHtml example on a MOD54417 is shown below. The example has been modified to add some variables that do simple counting.

```
void UserMain(void *pd)
{
    init();                                // Initialize network stack
    StartHttp();                           // Start web server, default port 80
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    iprintf("Web Application: %s\r\nNNDK Revision: %s\r\n", AppName, GetReleaseTag());

    uint32_t i = 0;
    uint32_t j = 0;
    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND);
        i++;
        j = j + 1;
        iprintf("i: %ld, j: %ld\r\n", i, j);
    }
}
```

Load the application. Note that the DEVIP environment variable is set to the device's IP address 10.1.1.169. The application will begin execution and wait for GDB to connect.

```
C:\NetBurner\projects\SimpleHtml>make loaddebug -j
C:\nburn\pcbin\nbupdate obj/debug/DBSimpleHtml.bin 10.1.1.169
uriReq: http://10.1.1.169:20034/appupdate.htm

urlStr: http://10.1.1.169:20034/appupdate.htm
200

C:\NetBurner\projects\SimpleHtml>
```

Start GDB:

```
C:\NetBurner\projects\SimpleHtml>m68k-elf-gdb -se obj/debug/DBSimpleHtml.elf
GNU gdb (GDB) 8.2
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=m68k-unknown-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from obj/debug/DBSimpleHtml.elf...done.
(gdb)
```

Connect to the MOD54417 target on port 2159. This will pause the application at whatever line of code it happens to be on. For simple examples that will typically be the NBRtos idle task.

```
(gdb) target remote 10.1.1.169:2159
Remote debugging using 10.1.1.169:2159
OSTaskIdle (data=0x0) at C:/nburn/arch/coldfire/source/nbrtosmain.cpp:425
425          asm ( " NOP" );
(gdb)
```

At this point you can do whatever type of GDB commands you wish. To set a break point at the `i++`; at line 29 of `main.cpp` and use the `continue` command to execute until the break point is reached:

```
(gdb) break main.cpp:29
Breakpoint 1 at 0x40009282: file src/main.cpp, line 29.
(gdb) info break
Num   Type             Disp Enb Address          What
1     breakpoint        keep y   0x40009282 in UserMain(void*) at src/main.cpp:29
(gdb) continue
Continuing.
[Switching to Thread 2]

Thread 2 hit Breakpoint 1, UserMain (pd=0x0) at src/main.cpp:29
29          i++;
(gdb)
```

The `list` command can be used to view the source code around the breakpoint:

```

Thread 2 hit Breakpoint 1, UserMain (pd=0x0) at src/main.cpp:29
29             i++;
(gdb) list
24             uint32_t i = 0;
25             uint32_t j = 0;
26             while (1)
27             {
28                 OSTimeDly(TICKS_PER_SECOND);
29                 i++;
30                 j = j + 1;
31                 iprintf("i: %ld, j: %ld\r\n", i, j);
32             }
33         }
(gdb)
\code

\latexonly \end{tcolorbox} \endlatexonly

\n
Now view the variables with print, as well as all local variables:

\latexonly \begin{tcolorbox} \endlatexonly

\code
(gdb) print i
$1 = 2
(gdb) info local
i = 2
j = 2
(gdb)

```

Use `next` to go to the next line without stepping into a function (the `step` command steps into a function). GDB commands can typically use just the first letter, in this case `n`:

```

(gdb) n
30             j = j + 1;
(gdb) info local
i = 3
j = 2
(gdb)

```

To make changes to your code use the `detach` command so that the application can resume and will be ready for your next code download:

```

(gdb) detach
Detaching from program: C:\NetBurner\projects\SimpleHtml\obj\debug\DBSimpleHtml.elf, Remote target
Ending remote debugging.
[Inferior 1 (Remote target) detached]
(gdb)

```

To exit GDB, use the `quit` command:

```

(gdb) quit

C:\NetBurner\projects\SimpleHtml>

```

3.7 Addr2line

Note

On Windows and ColdFire-based modules, you can use [WinAddr2Line](#) instead.

To get line numbers and function names from trap memory addresses, you can use `addr2line` on the command line like this:

```
addr2line -ifCe Release/YourProject.elf 00001111 11112222
```

Replace "addr2line" with the path to the appropriate program below, and replace the numbers with memory addresses you want to find.

3.7.1 ColdFire platforms

For ColdFire based platforms, use `m68k-elf-addr2line`. This is distributed in the `\nburn\gcc\bin\` folder.

3.7.2 ARM platforms

For ARM-based platforms, use `arm-unknown-eabi-addr2line`. This is distributed in the `\nburn\gcc\bin\` folder.

Chapter 4

Device Discovery and Configuration

4.1 Introduction

Device discovery and configuration can be accomplished in a number of ways, including methods that are platform independent, including Windows, macOS, Linux, tablets and mobile phones.

Note

Please refer to the [Production & Deployment](#) section of this document for methods and utilities designed to program and configure devices in a production environment.

4.1.1 Device Discovery Methods:

- If the device and computer have Internet access, open a web browser and go to 'discover.netburner.com'.
- Use the localdiscover utility located in `\nburn\pcbin\localdiscover`. This is a multi-platform utility written in Golang for Windows, macOS and Linux. The compiled version can be distributed to your customers, but not the Golang source code.
- Use the Python find utility, located in `\nburn\pctools\find`.
- Use the debug/console serial port.

4.1.2 Device Configuration methods:

- Interactively using a web browser to access the device's Configuration Server
- Sending a JSON object and a utility such as wget (See [Production & Deployment](#))
- The device's serial interface to the Configuration Server

The primary configuration method is through the device's Configuration Web Server. By utilizing a web server, a device can be configured from any platform or operating system. The underlying configuration mechanism is handled through a JavaScript Object Notation (JSON) object. This will be described in more detail later, but the important point is that it provides a simple interface that can be used by non-embedded developers (e.g. web developers) to interact and configure the device. The implementation is also designed to enable you to add any application specific information to be stored and accessed in the same manor, such as calibration values, set points, etc.

You are also able to create your own configuration web interface that will appear in place of the default interface. In this way you can organize and present data in the appropriate way for your customers, as well as add your own custom logo, images and descriptions. Examples are provided.

4.2 Configuration Procedure

- Connect your NetBurner device to a network with Internet access.
- Open a web browser on your host computer and go to `discover.netburner.com`. You will see a list of devices on your network. (If the device does not have Internet access, use the `localdiscover` utility instead).

All system and application settings can now be viewed and/or modified by navigating the tree structure.

Using `discover.netburner.com`:

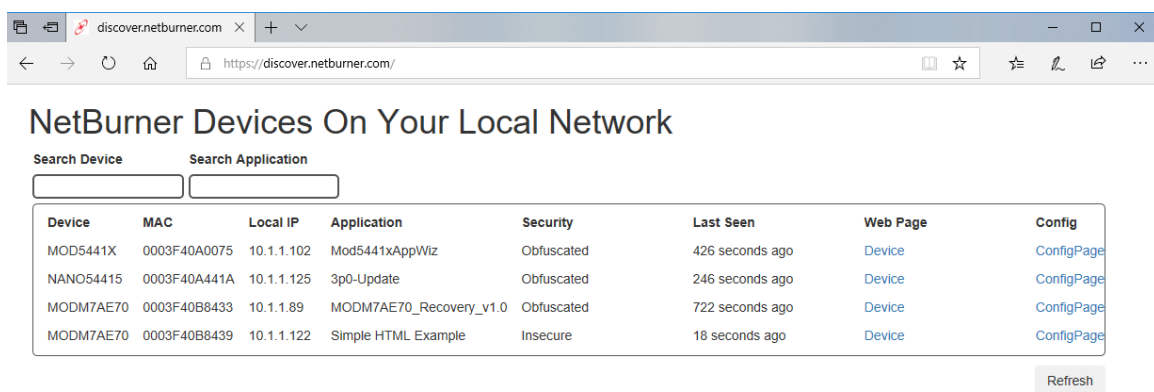


Figure 4.1 Discover using `discover.netburner.com`

Using `localdiscover` utility:

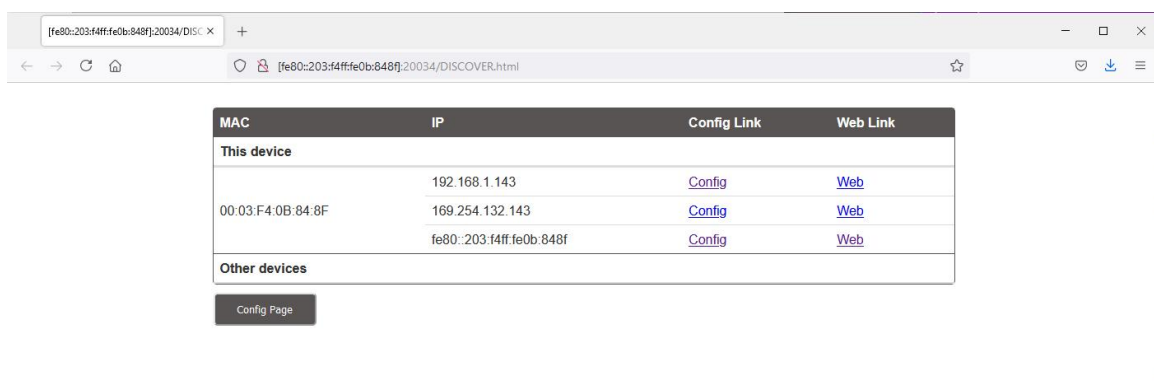


Figure 4.2 LocalDiscover Utility

The "Device" link will take you to the device application web page. The "ConfigPage" link will take you to the device configuration web page, as shown below:



Basic Config System Example x +

← → ↻ ⓘ Not secure | 10.1.1.73:20034/ROOT.html

▼ Config

▶ AppData

▼ Sys

Application Simple HTML Example

▼ Boot

Abort A

BootBaud 115200

BootDelay 2

BootQuiet ☐

BootUart 0

Password

SerialConfig DuringBoot ▼

User

▼ NetIf

▼ Ethernet0

DeviceName

DhcpDiscoverSec 1

DiscoveryObfuscate ☒

DiscoveryReportInterval 900

DiscoveryReportUrl discover.netburner.com/Devi

▶ IPv4

▶ IPv6

MAC 00:03:F4:0B:83:F5

Platform MODM7AE70

Version 8

Reboot ☒

Update Application

Choose File Send File

NetBurner

Expand All

Collapse All

Update Record

Raw Config Page

NB Discover

Figure 4.3 Configuration Web Page

4.2.1 Configuration Settings

4.2.1.1 Config

Top level configuration leaf containing System (Sys) and Application Data (AppData) configuration leafs. AppData will only be populated if an application uses the configuration system for storage/configuration of application specific items, as opposed to common system configuration settings.

Setting	Description
Version	Configuration system write counter. Increments each time new values are saved.
Reboot	Check this option to reboot the system once new values are saved

4.2.1.2 Update Application

A new application can be programmed into the device by choosing an image file and selecting the "Send File" button. Note that the Reboot checkbox should normally be checked so that the device reboots and runs the new application that was sent.

4.2.1.3 Config > Sys

System settings

Setting	Description
Application	Name of running application.
Platform	Name of hardware platform, such as MOD54415

4.2.1.4 Config > Sys > BOOT

Boot settings.

Setting	Description
Abort	Character or string to send to the boot serial port to abort the application boot process and enter the Serial Port Configuration interface. Default is 'A'.
BootBaud	Boot serial port baud rate. Default is 115200, N, 8, 1.
BootDelay	Number of seconds to delay starting the application. This is the time available to send the Abort character or string. Default is 2 seconds. DO NOT SET TO 0.
BootQuiet	At power-up the boot serial port will send the message: "Waiting 2 seconds to boot..." Enabling quiet boot will disable this message. Default is off.
BootUart	Designated boot UART number. Varies by platform, but typically default is 0.
User	Optional user name for password protection of the web page interface.
Password	Optional password to access the web page configuration interface.
SerialConfig	Boot options for the Serial Port Configuration interface: - During Boot (default) - Always Enabled - Pause after boot - Disabled

Note

The password can be reset through the Serial Port Configuration interface.

4.2.1.5 Config > Sys > NetIf

The NetIf section contains settings for all available network interfaces. For example, if the system has a single Ethernet interface, there will be a section named `Ethernet0`. If there are two Ethernet interfaces, the second will be `Ethernet1`. If a Wifi interface is available it will be listed as well.

4.2.1.6 Config > Sys > NetIf > Ethernet0

Settings for each Ethernet interface

Setting	Description
DeviceName	Interface name used for DNS and NetBIOS
DhcpDiscoverSec	Number of seconds after boot to wait before sending a DHCP Discover message. Default is 1.
DiscoveryObfuscate	Send data to Discover Server obfuscated rather than plain text for security. Default is enabled.
DiscoveryReportInterval	How often to report to Discovery Server in seconds. Default is 900.
DiscoveryReportUrl	Name of Discover Server. Default is: <code>discover.netburner.com/Device</code> ↵ <code>Post</code> .
MAC	MAC address
SupressDefault	Disable Ping and UDP Echo ports. Default is not suppressed.
VlanTag	Virtual Network tag number. Default is 0

4.2.1.7 Config > Sys > NetIf > Ethernet0 > IPv4

IPv4 network interface settings.

Setting	Description
ActiveAddr	Current IP address.
ActiveDNS1	Current DNS1 address.
ActiveDNS2	Current DNS2 address.
ActiveGate	Current Gateway address.
ActiveMask	Current IP Mask.
AutoIPAddr	Auto IP address.
AutoIPEn	Enable AutoIP feature. Default is enabled.
EnableMacmDNS	Enable device access by name in the format: <code>nburnxxxxxx.local</code> , where <code>xxxxxx</code> is the last 6 digits of the MAC address. Default is disabled.
LocalName	Specify device access by name in the <code>.local</code> format. For example: <code>mydevice.local</code> .

Setting	Description
Mode	IP Address Mode: - DHCP (default) - DHCP with fallback to static - Static - Disabled
StaticAddr	Static IP address.
StaticDNS1	Static DNS1 address.
StaticDNS2	Static DNS2 address.
StaticGate	Static Gateway address.
StaticMask	Static IP Mask.

4.2.1.7.1 DHCP With Fallback Details

After boot, the first [DHCP](#) Discover message will be sent once the network link is active. Note that if the network link is not active, such as when a network cable is not connected, [DHCP](#) will not start, and the fallback to a static address will not occur.

Once [DHCP](#) starts it will attempt to get the IP settings from a [DHCP](#) server. If not successful, it will retry 4 more times at 8, 16, 23, and 64 seconds. If not successful, then the fallback to static IP address settings will occur.

For custom applications, the retries can be modified in the [DHCP](#) Client header file (may be moved to [constants.h](#) in a future release). The defaults are shown below:

```
#define DEFAULTTIMEOUT (4)
#define DEFAULTRETRY (5)
```

4.2.1.8 Config > Sys > NetIf > Ethernet0 > IPv6

IPv6 network interface settings main leaf. Note: refer to [DHCP](#) With Fallback Details section above.

Setting	Description
Mode	IP Address Mode: - DHCP (default) - DHCP with fallback to static - Static - Disabled
StaticAddr	Static IP address.
StaticDNS1	Static DNS1 address.
StaticDNS2	Static DNS2 address.

4.2.1.8.1 Config > Sys > NetIf > Ethernet0 > IPv6 > ActiveAddr

Displays a list of active addresses. The number will depend on the network architecture. All devies should have at least a link local address with prefix fe80::. Other addresses can be assigned from a DHCPv6 server and routers on the network. The example below shows link local, DHCPv6 and a router.

```
0 fe80::203:f4ff:fe07:386b
1 2600:1700:410:190f:203:f4ff:fe07:386b
2 2600:1700:410:190f::2000
```

4.2.1.8.2 Config > Sys > NetIf > Ethernet0 > IPv6 > ActiveDNS

Displays a list of active DNS serversaddresses. The number will depend on the network architecture. All devies should have at least a link local address with prefix fe80::. Other addresses can be assigned from a DHCPv6 server and routers on the network. The example below shows link local, DHCPv6 and a router.

```
0 fe80::203:f4ff:fe07:386b
1 2600:1700:410:190f:203:f4ff:fe07:386b
2 2600:1700:410:190f::2000
```

4.2.1.8.3 Config > Sys > NetIf > Ethernet0 > IPv6 > ActiveDNS

Displays a list of active DNS servers.

4.2.1.8.4 Config > Sys > NetIf > Ethernet0 > IPv6 > Route

Displays a list of active router addresses.

4.3 The JSON Object

It is not necessary to understand the underlying JSON implementation, but for those that are interested a sample JSON object is shown below. It identifies each configuration field, which can range from a string, boolean, or a selection of values. For developers, the programmers guide and examples provide the operational details.

Warning

If you do decided to use any of the JSON configuration methods, you must first download a JSON object from your specific device. Under no circumstances should you use the example below as you actual device will likely be different in some respects.

The root of the object is Config. The next levels are AppData and Sys. AppData is available for your application to store information. Sys is used for the system configuration such as IP settings.

```
{
  "Config":{
    "AppData":{},
    "Sys":{
      "Application":"Simple HTML Example",
      "Boot":{
        "Abort":"A",
        "BootBaud":115200,
        "BootDelay":2,
        "BootQuiet":false,
        "BootUart":0,
        "Password":"",
        "SerialConfig":{
          "Choices":"DuringBoot, AlwaysEnabled, PauseAfterBoot, Disabled", "Value":"DuringBoot"
        },
        "User":""
      },
      "NetIf":{
        "Ethernet0":{
          "DeviceName":"",
          "DhcpDiscoverSec":1,
          "DiscoveryObfuscate":true,
          "DiscoveryReportInterval":900,
          "DiscoveryReportUrl":"discover.netburner.com/DevicePost",
          "IPv4":{
            "ActiveAddr":"10.1.1.73",
            "ActiveDNS1":"10.1.1.1",
            "ActiveDNS2":"0.0.0.0",
            "ActiveGate":"10.1.1.1",
            "ActiveMask":"255.255.252.0",
            "AutoIPAddr":"169.254.131.245",
            "AutoIPEn":true,
            "Mode":{
              "Choices":"DHCP,DHCP w Fallback,Static,Disabled", "Value":"DHCP"
            },
            "StaticAddr":"0.0.0.0",
            "StaticDNS1":"0.0.0.0",
            "StaticDNS2":"0.0.0.0",
            "StaticGate":"0.0.0.0",
            "StaticMask":"0.0.0.0"
          },
          "IPv6":{
            "ActiveAddr":["2602:306:b8e9:c83f::14c0","2602:306:b8e9:c83f:203:f4ff:fe0b:83f5","fe80::203:f4ff:fe0b:83f5"],
            "ActiveDNS":["2602:306:b8e9:c83f:208:a2ff:fe0c:b081","2602:306:b8e9:c83f:208:a2ff:fe0c:b081"],
            "ActiveRoute":["fe80::1:1"],
            "Mode":{
              "Choices":"DHCP,DHCP w Fallback,Static,Disabled", "Value":"DHCP"
            },
            "StaticAddr":"::",
            "StaticDNS1":"::",
            "StaticDNS2":"::"
          },
          "MAC":"00:03:F4:0B:83:F5"
        },
        "Platform":"MODM7AE70"
      },
      "Version":8,
      "Reboot":false
    }
  }
}
```

4.3.1 Developer Notes

If you were writing a web interface and need only the data under Ethernet in the tree, it can be accessed by: <http://10.1.1.71:20034/UI.html?CONFIG/SYS/NETIF/Ethernet>

To obtain just the IP address: <http://10.1.1.71:20034/UI.html?CONFIG/SYS/NETIF/Ethernet/IPv4/ActiveAddr>

Recommended Examples: The following examples are recommended to begin evaluating the platform. They are located in the \nburn\examples directory.

- ShowInterfaces
- \Configuration\Web\BasicWebConfig

4.4 Configuration Security

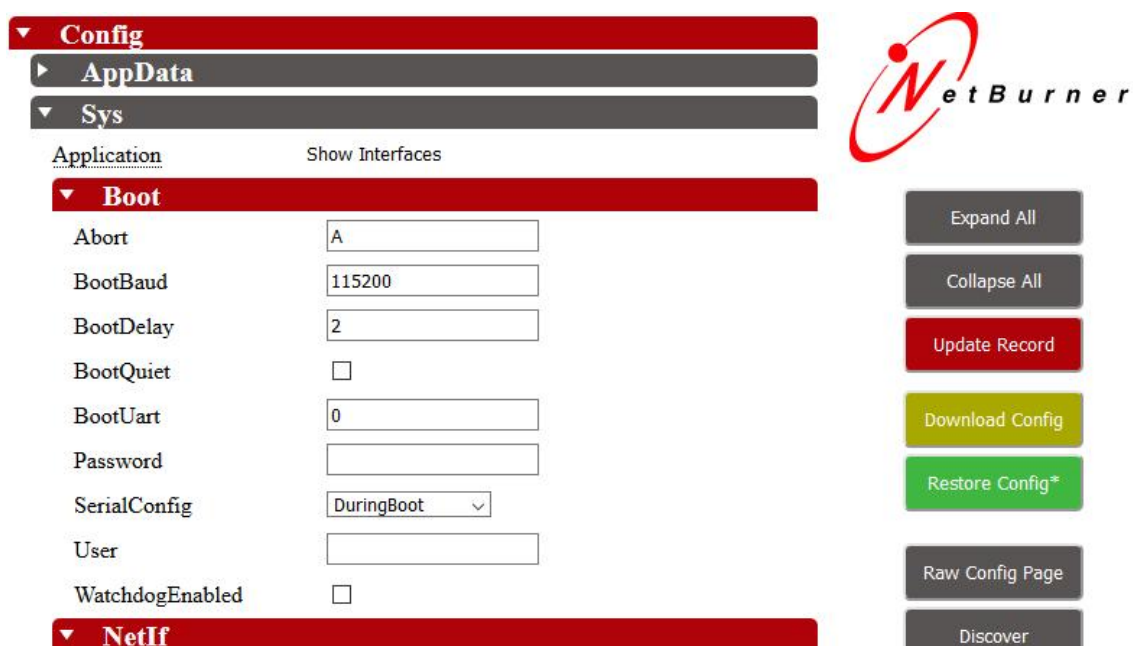
One of the features of NetBurner 3.0 is that each device is configured through its own configuration web interface. Data sent and received through the network interface has three options for security:

- Open: No security, plain text
- Obfuscated: Encrypted, but without TLS and a secure certificate and private key
- Secure: Encrypted with TLS and a certificate and private key

4.5 Serial Configuration Server

Network connectivity is not required to use your NetBurner device. If enabled, the Configuration Server can be accessed through the device's serial port. The configuration tree is exactly the same as what you see on the web interface and in the JSON object. For example, the Ethernet0 settings are located at `Config > Sys > NetIf > Ethernet0`.

The serial port interface is located on UART0 by default. It can be modified and enabled/disabled in the `Config > Sys > Boot` level of the Configuration Server.



Config

- AppData**
- Sys**
 - Boot**
 - Abort: A
 - BootBaud: 115200
 - BootDelay: 2
 - BootQuiet: ☐
 - BootUart: 0
 - Password:
 - SerialConfig: DuringBoot
 - User:
 - WatchdogEnabled: ☐
 - NetIf**

NetBurner

Expand All

Collapse All

Update Record

Download Config

Restore Config*

Raw Config Page

Discover

Figure 4.4 Config Boot Level

4.5.1 Serial Configuration

To access the serial port configuration:

- Connect a cable to the serial port selected as the BootUart
- Run a serial terminal program such as MTTTY
- Reset the device, and type a 'A' when prompted to Abort the boot sequence

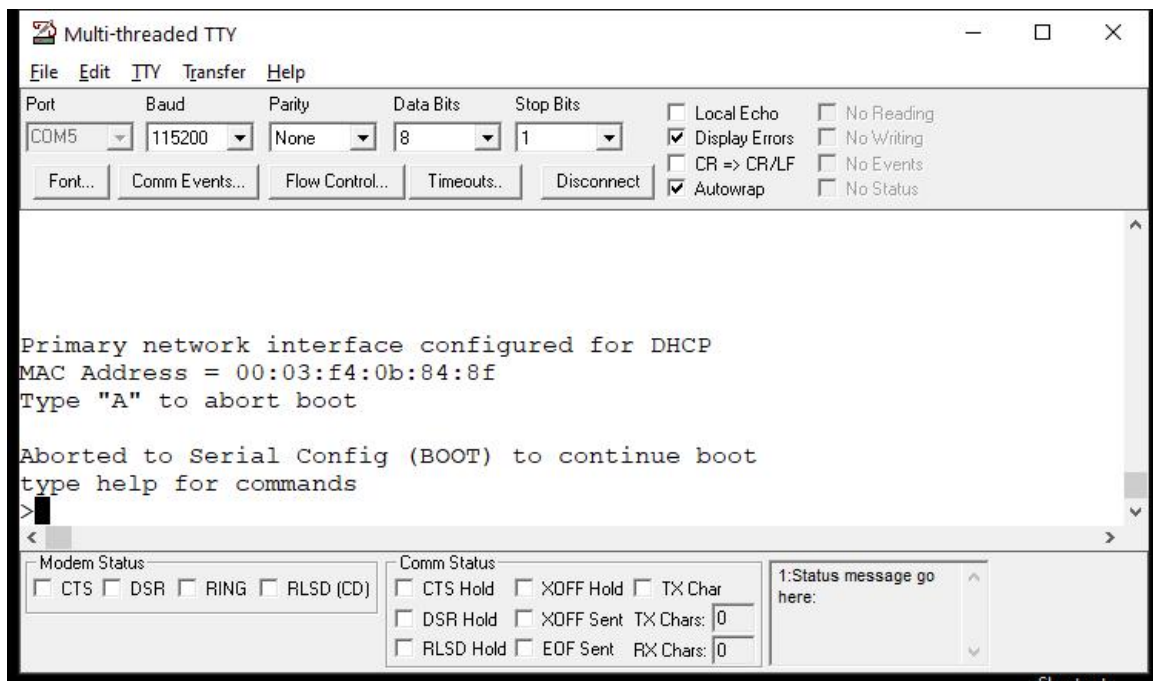


Figure 4.5 Abort Boot Sequence

You are now in the serial configuration mode as indicated by the '>' character. Typing "help" will display the help menu.

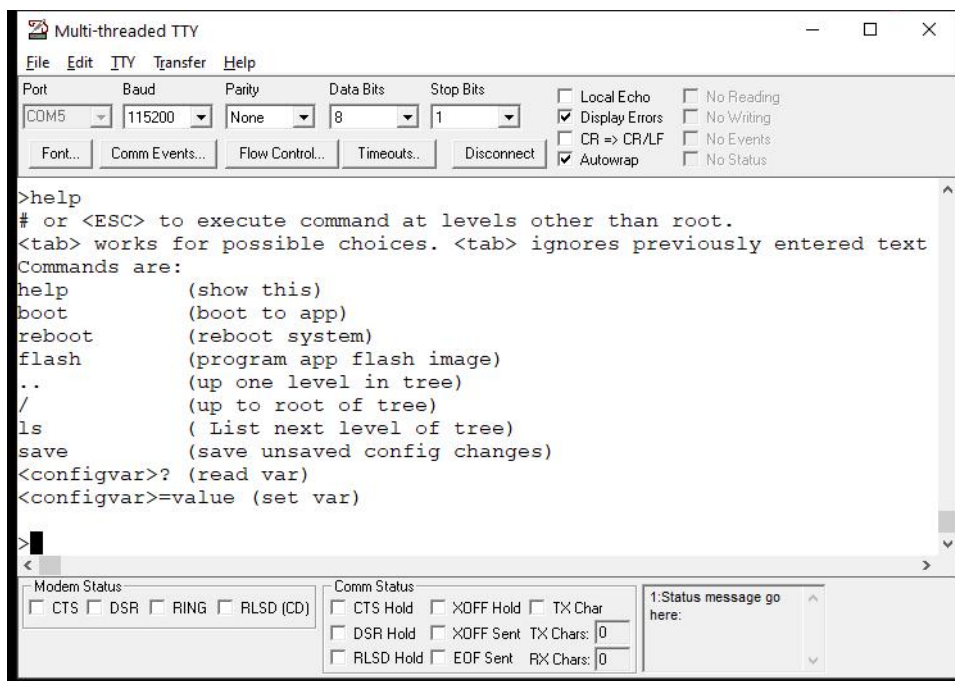


Figure 4.6 Serial Help Menu

Typing the "ls" command displays the next level of the tree, which is `Config` since we are at the root level. In this example we will set the Static IP values. Note that to change from `DHCP` to static, we must also change the `Mode` option from `DHCP` to Static. To navigate to the next level you type in the name of that level. For example, to go to the `Config` level:

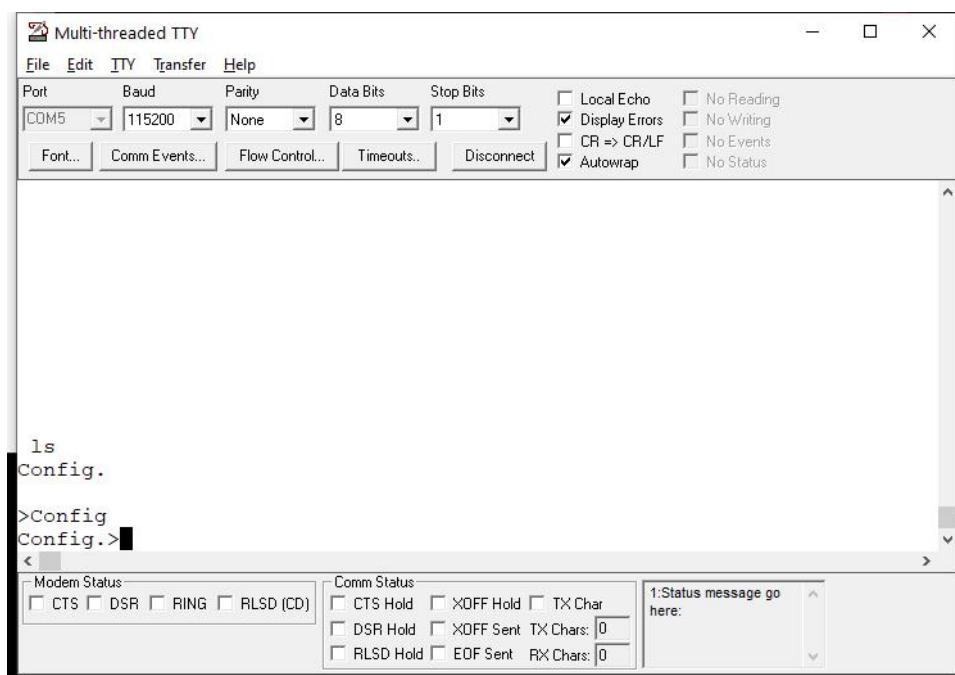


Figure 4.7 Config Level

The prompt will show the current location, in this case `Config>`. In the same manor to navigate to the IPv4 settings the level names are typed into the terminal. Note that tabbed auto-completion is also available.

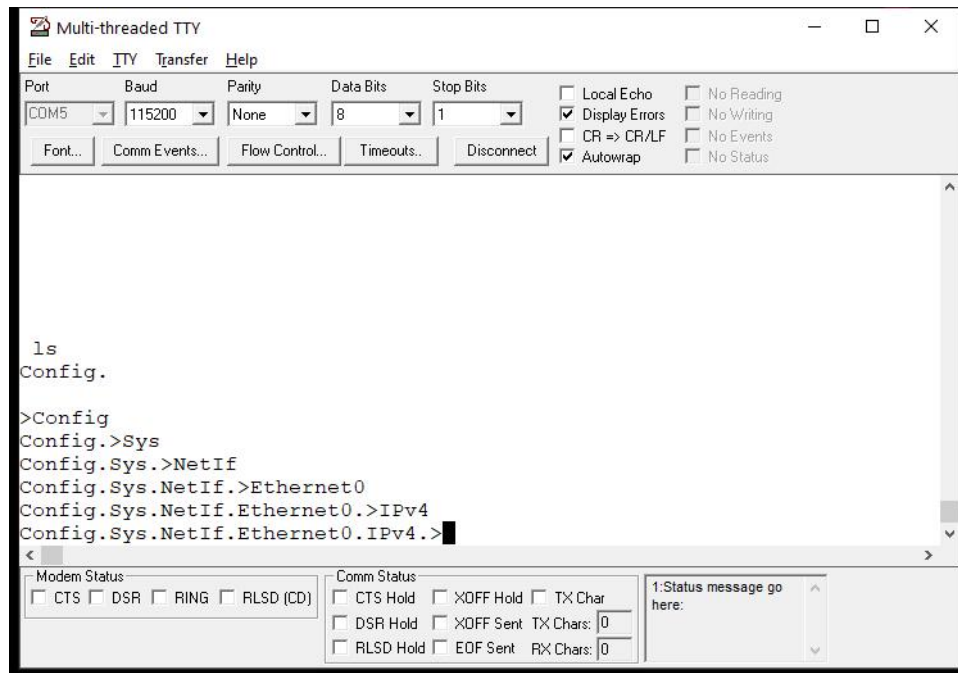


Figure 4.8 IPv4 Level

At any level, typing a "?" will display the current values of that level and all levels beneath in the JSON object format. For example:

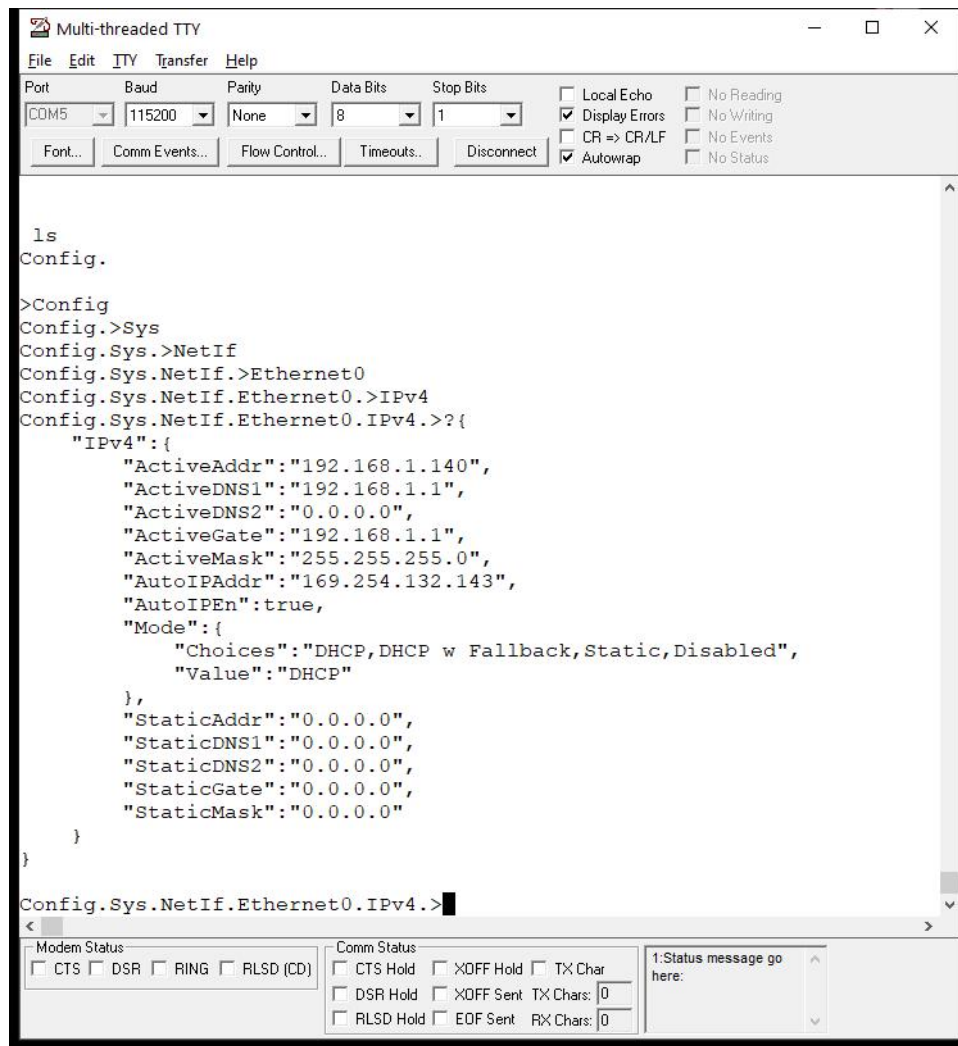


Figure 4.9 Display IPv4 Level

In this example we are interested in the static IP settings. There are two choices to set them: from the IPv4 level by including the name of the next level and the value, or by going down to each level individually and entering just the value. In this case we will stay at the IPv4 level and include the name of the static item we want to set. To set the StaticAddr type: StaticAddr="10.1.1.99". We will also type a "?" to see the changed value:

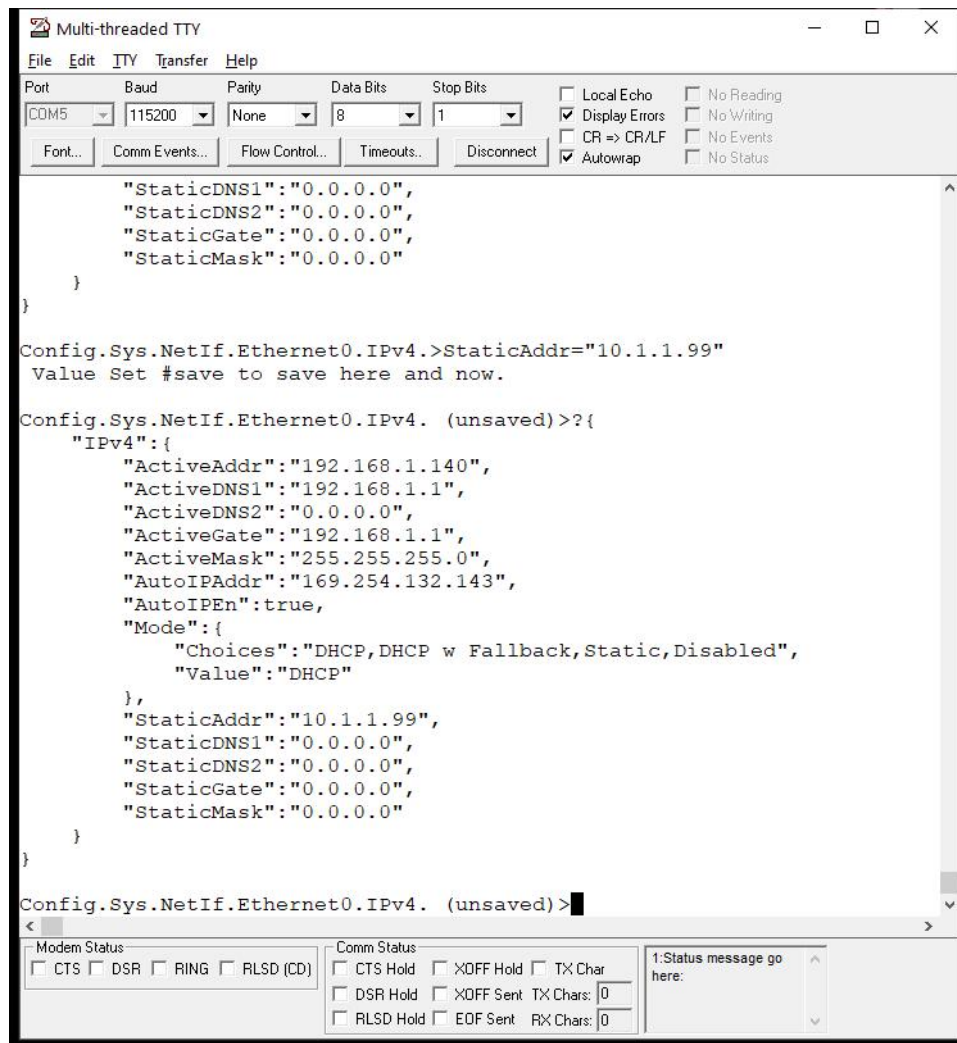


Figure 4.10 Set StaticAddr Value

In the same manor we will set the Mask, Gateway and DNS. To set the Mode we will need to either include the Mode level in the setting string or go to the Mode level, since the setting is a Choices selection and not a simple string like the static IP settings. To accomplish this from the IPv4 level type: `Mode.Choices="Static"`. If you navigate the to Mode level, the command would be: `Choices="Static"`. At this point the unsaved settings are:

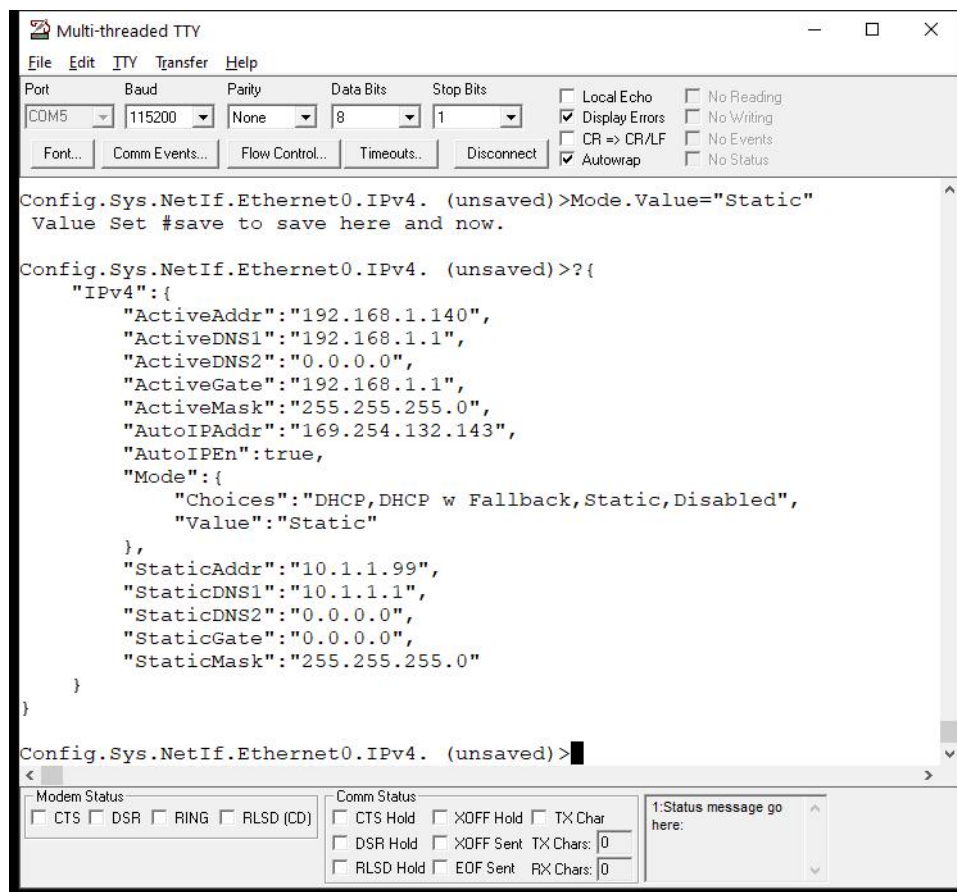


Figure 4.11 Static Summary

Warning

From the settings display you can see that the Mode selector control has two settings: Choices and Value. The Choices are values that appear in the web page selector control and you do not want to modify them. Only modify the Value to one of the Choices values.

To store the modified values in the Configuration Record in Flash memory, type #save. This will store the values, but the IP settings will not take effect until the device is rebooted.

4.5.2 Serial App Downloads

Just as with the select file and upload feature of the web interface, applications can be uploaded to your device using the Serial Configuration Server.

The procedure is:

- Abort the boot process type typing 'A' when prompted to enter the Serial Configuration server mode. Note: the SerialConfig Server options includes an "AlwaysEnabled" setting so that the Serial Configuration Server is always active and a reboot and Abort is not required. This can be especially useful during development if you are not connecting your device to a network.
- At the '>' prompt, type "flash"
- When the download prompt appears, send the .bin file using your serial terminal program

The serial terminal output is shown below:

4.5.2.1 Start of Serial Configuration Server.

```
MAC Address = 00:03:f4:0e:21:ee
Type "A" to abort boot

Aborted to Serial Config (BOOT) to continue boot
type help for commands
>
```

4.5.2.2 Optional, type "help" to see a list of commands.

```
MAC Address = 00:03:f4:0e:21:ee
Type "A" to abort boot

Aborted to Serial Config (BOOT) to continue boot
type help for commands
>help# or <ESC> to execute command at levels other than root.
<tab> works for possible choices. <tab> ignores previously entered text
Commands are:
help          (show this)
boot          (boot to app)
reboot        (reboot system)
flash         (program app flash image)
..            (up one level in tree)
/             (up to root of tree)
ls            ( List next level of tree)
save          (save unsaved config changes)
Config        (enter Config mode)
  <configvar>? (read var)
  <configvar>=value (set var)
  Ex Config Seqs:
  > Config.Sys.Boot?
  > Config
  > Config.Sys.NetIf.Ethernet0.Ipv4.StaticAddr="10.1.0.100"

>
```

4.5.2.3 Type "flash" command

```
MAC Address = 00:03:f4:0e:21:ee
Type "A" to abort boot

Aborted to Serial Config (BOOT) to continue boot
type help for commands
>help# or <ESC> to execute command at levels other than root.
<tab> works for possible choices. <tab> ignores previously entered text
Commands are:
help          (show this)
boot          (boot to app)
reboot        (reboot system)
flash         (program app flash image)
..            (up one level in tree)
/             (up to root of tree)
ls            ( List next level of tree)
save          (save unsaved config changes)
Config        (enter Config mode)
  <configvar>? (read var)
  <configvar>=value (set var)
  Ex Config Seqs:
  > Config.Sys.Boot?
  > Config
  > Config.Sys.NetIf.Ethernet0.Ipv4.StaticAddr="10.1.0.100"

> flash
Begin Download
```

4.5.2.4 Send Application File

The device is now waiting to receive the application image file. Use your serial terminal option to send a binary file, and select the .bin file of the application you wish to send. After the file is uploaded, the device will reboot. In the display below the flash command is entered, the file is sent, and the status messages confirming the programming was complete. The device reboot information of the new application begins with "Primary network interface....".

```
> flash
Unknown command (help for help)
Begin Download

Programming complete

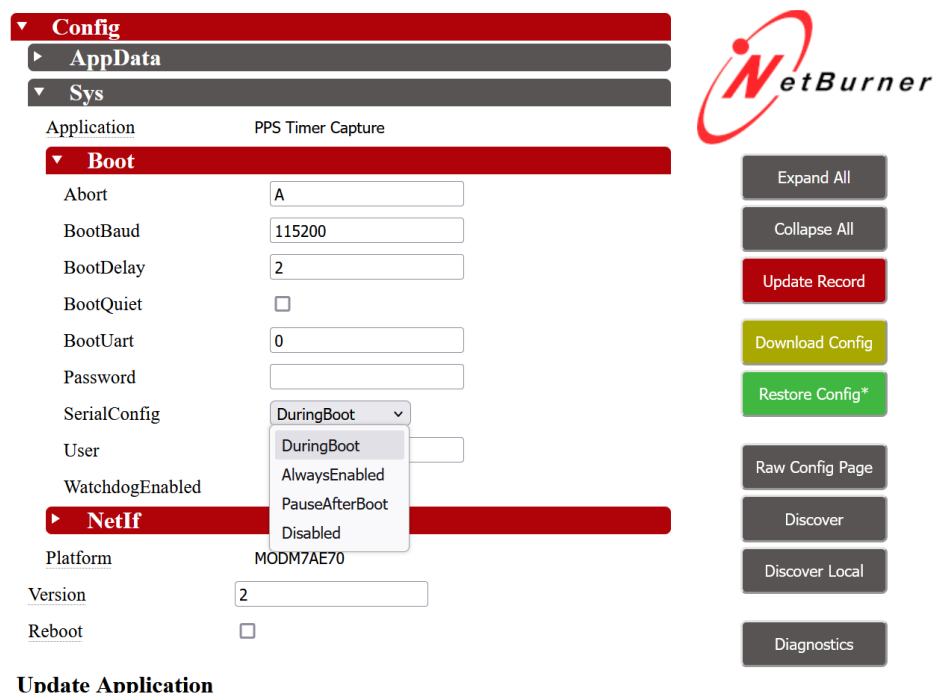
Primary network interface configured for DHCP
MAC Address = 00:03:f4:0e:21:ee
Type "A" to abort boot
```

4.5.3 Serial Config Settings

As mentioned previously, the Serial Configuration Server behavior has four options:

- During Boot. Must abort boot sequence by sending an 'A'.
- Always Enabled. Serial Configuration Server will always be active and process commands.
- Pause After Boot. The device will boot into the Serial Configuration Server instead of the application.
- Disabled.

The SerialConfig field is used to select the desired option. It can be done through the web page, or through the Serial Configuration Server itself. It is located in the Config > Sys > BOOT section of the tree.



Config

- AppData**
- Sys**
 - Application
 - PPS Timer Capture
 - Boot**
 - Abort: A
 - BootBaud: 115200
 - BootDelay: 2
 - BootQuiet: ☐
 - BootUart: 0
 - Password:
 - SerialConfig: DuringBoot (dropdown menu open showing: DuringBoot, AlwaysEnabled, PauseAfterBoot, Disabled)
 - User:
 - WatchdogEnabled: ☐
 - NetIf**
 - Platform: MODM7AE70
 - Version: 2
 - Reboot: ☐

Update Application

NetBurner

Expand All
Collapse All
Update Record
Download Config
Restore Config*
Raw Config Page
Discover
Discover Local
Diagnostics

Figure 4.12 SerialConfig

From a serial terminal using the Serial Configuration Server itself. The example below uses the full path from the root, but you can also traverse the tree by entering each in succession: Config, Sys, Boot, SerialConfig. A '?' returns the JSON object's current setting and options.

Command Examples:

- Config.Sys.Boot.SerialConfig?
- Config.Sys.Boot.SerialConfig.Value="Default"
- Config.Sys.Boot.SerialConfig.Value="AlwaysEnabled"

```
>
> Config.Sys.Boot.SerialConfig?
  "SerialConfig":{
    "Choices":"DuringBoot,AlwaysEnabled,PauseAfterBoot,Disabled",
    "Value":"DuringBoot"
  }
primary network interface configured for DHCP
MAC Address = 00:03:f4:0e:21:ee
Type "A" to abort boot

Aborted to Serial Config (BOOT) to continue boot
type help for commands
>Config.Sys.Boot.SerialConfig.ValueConfig leaf not found, help for help

>

> Config.Sys.Boot.SerialConfig?{
  "SerialConfig":{
    "Choices":"DuringBoot,AlwaysEnabled,PauseAfterBoot,Disabled",
    "Value":"DuringBoot"
  }
}

> Config.Sys.Boot.SerialConfig.Value="AlwaysEnabled" Value Set #save to save here and now.

Config.Sys.Boot.SerialConfig.Value. (unsaved)>

Config.Sys.Boot.SerialConfig.Value. (unsaved)> ?"AlwaysEnabled"
Config.Sys.Boot.SerialConfig.Value. (unsaved)> ..Config.Sys.Boot.SerialConfig. (unsaved)>

Config.Sys.Boot.SerialConfig. (unsaved)> ?{
  "SerialConfig":{
    "Choices":"DuringBoot,AlwaysEnabled,PauseAfterBoot,Disabled",
    "Value":"AlwaysEnabled"
  }
}

Config.Sys.Boot.SerialConfig.Value. (unsaved)>
Config.Sys.Boot.SerialConfig.Value. (unsaved)> #save
Config.Sys.Boot.SerialConfig.Value.>
```


Chapter 5

Migration Guides

There are currently three NetBurner development tool branches:

- 2.7.x, IPv4 only for all products except ARM.
- 2.9.x, IPv4/IPv6 dual stack for all products except ARM. This release will continue to be fully supported for ColdFire devices.
- 3.x, Latest release including IoT enhancements. Supports 5441x (MOD5441x, NANO54415, SB800EX) and ARM (MODM7AE70) platforms.

Note

If you are updating from a 2.7.x release to a 2.9.x release, please see the IPv6 guide in the `\nburn\docs` folder.

The following sections will describe the migration path in more detail.

- [Release 2.x to 3.x Overview](#)
- [Release 2.x to 3.x Porting Guide](#)
- [Upgrade 5441x Platforms to 3.x](#)

5.1 Release 2.x to 3.x Overview

When developing NetBurner 3.x we incorporated customer requests to enable the use of the latest ARM base processors, security protocols, and IoT capabilities. For existing customers we have done our best to balance the additional capabilities with ease of migration.

5.1.1 Hardware

- Support for ARM processors, starting with the 300MHz M7.
- Support for NXP/Freescale 5441x based products: MOD54415, MOD54417, SB800EX, NANO54415. For other NetBurner platforms we will fully maintain the 2.9.x branch.
- Applications easily scale from single chip solutions to large memory platforms.

5.1.2 Configuration

A new configuration and update system has been created:

- Updates, discovery and configuration are now platform independent and do not require any computer OS specific utilities.
- Devices can be discovered by going to "discover.netburner.com" in a web browser.
- Applications can be updated from a web browser, the nbupdate utility, NBEclipse, WGET, or the serial interface.
- Configuration can now be done with a JSON object. These objects can be saved or uploaded, making configuration in a production environment much easier and faster.
- Developers can easily crease their own configuration objects.
- The IPSetup windows utility can be used to locate a device, but not configure it.

5.1.3 RTOS

The low level code of the RTOS has been modified to support new features including:

- Increased the number of tasks from 64 to a user defined value.
- OS functions are now managed as C++ objects.

5.1.4 TCP

- Increased the maximum number of sockets from 32 to a user defined value. A Maximum value over 256 may have an impact on performance.

5.1.5 Network Interfaces

- Many new interfaces supported
- Access to interfaces now requires looking at an Interface Object, rather than global variables such as ETHER-NETIP. This was needed to support additional interfaces, a larger number of interfaces, and IPv6 capability.

5.1.6 SSL/TLS

- Updated to the latest protocols and ciphers (2.9.x also has the SSL/TLS updates).
- The process for generating compiled-in certificates has been made easier, but is slightly different.
- Devices can now generate certificates.

5.1.7 System Initialization

- A new function, `init()`, is now used to initialize stack, all interfaces, and network debugging.

5.1.8 NBEclipse

- Updated to Oxygen
- Can generate makefiles that can be used on command line builds.

5.1.9 GCC

- Updated to 8.1

5.2 Release 2.x to 3.x Porting Guide

5.2.1 Device Configuration

Previously device configuration was accomplished with IPSetup, a windows based program. In 3.x there are a number of ways to configure the device that are independent of the host computer type. The primary method is through the device's own configuration web server. In this way the device can be configured by a web browser on any platform. There is also a serial interface configuration method.

- Can be configured by any web browser, and applications can be programmed into the device. *Note:* Autoupdate and IPSetup are no longer used to download applications or configure the device. However, you can still use IPSetup to discover a device's IP address.
- Devices can be found by typing `discover.netburner.com` in the web browser URL field.
- The underlying configuration mechanism is handled through a JavaScript Object Notation (JSON) object. This will be described in more detail later, but the important point is that it provides a simple interface that can be used by non-embedded developers (e.g. web developers) to interact and configure the device. The implementation is also designed to enable you to add any application specific information to be stored and accessed in the same manor, such as calibration values, set points, etc.
- You are able to create your own configuration web interface that will appear in place of the default interface. In this way you can organize and present data in the appropriate way for your customers, as well as add your own custom logo, images and descriptions. Examples are provided.

5.2.2 Configuration Procedure

- Connect your NetBurner device to a network with Internet access.
- Open a web browser on your host computer and go to `discover.netburner.com`. You will see a list of devices on your network. 
- The "Device" link will take you to the device's web page. The "ConfigPage" link will take you to the device's configuration web page as shown below: 

A sample JSON object is shown below:

```
{
  "Config":{
    "AppData":{},
    "Sys":{
      "Application":"Simple HTML Example",
      "Boot":{
        "Abort":"A",
        "BootBaud":115200,
        "BootDelay":2,
        "BootQuiet":false,
        "BootUart":0,
        "Password":"",
        "SerialConfig":{
          "Choices":"DuringBoot, AlwaysEnabled, PauseAfterBoot, Disabled", "Value":"DuringBoot"
        },
        "User":""
      },
    },
    "NetIf":{
      "Ethernet0":{
```

```

    "DeviceName": "",
    "DhcpDiscoverSec": 1,
    "DiscoveryObfuscate": true,
    "DiscoveryReportInterval": 900,
    "DiscoveryReportUrl": "discover.netburner.com/DevicePost",
    "IPv4": {
        "ActiveAddr": "10.1.1.73",
        "ActiveDNS1": "10.1.1.1",
        "ActiveDNS2": "0.0.0.0",
        "ActiveGate": "10.1.1.1",
        "ActiveMask": "255.255.252.0",
        "AutoIPAddr": "169.254.131.245",
        "AutoIPEn": true,
        "Mode": {
            "Choices": "DHCP,DHCP w Fallback,Static,Disabled", "Value": "DHCP"
        },
        "StaticAddr": "0.0.0.0",
        "StaticDNS1": "0.0.0.0",
        "StaticDNS2": "0.0.0.0",
        "StaticGate": "0.0.0.0",
        "StaticMask": "0.0.0.0"
    },
    "IPv6": {
        "ActiveAddr": ["2602:306:b8e9:c83f::14c0", "2602:306:b8e9:c83f:203:f4ff:fe0b:83f5", "fe80::203:f4ff:fe0b:83f5"],
        "ActiveDNS": ["2602:306:b8e9:c83f:208:a2ff:fe0c:b081", "2602:306:b8e9:c83f:208:a2ff:fe0c:b081"],
        "ActiveRoute": ["fe80::1:1"],
        "Mode": {
            "Choices": "DHCP,DHCP w Fallback,Static,Disabled", "Value": "DHCP"
        },
        "StaticAddr": "::",
        "StaticDNS1": "::",
        "StaticDNS2": "::"
    },
    "MAC": "00:03:F4:0B:83:F5"
  },
  "Platform": "MODM7AE70"
},
"Version": 8,
"Reboot": false
}

```

If you were writing a web interface and need only the data under Ethernet in the tree, it can be accessed by:

<http://10.1.1.71:20034/UI.html?CONFIG/SYS/NETIF/Ethernet>

To obtain just the IP address: <http://10.1.1.71:20034/UI.html?CONFIG/SYS/NETIF/Ethernet/IPv4/ActiveAddr>

5.2.3 Recommended Examples

The following examples are recommended to begin evaluating the platform. They are located in the `\nburn\examples` directory.

- ShowInterfaces
- \Configuration\Web\BasicWebConfig

5.2.4 Configuration Security

One of the features of NetBurner 3.0 is that each device is configured through its own configuration web interface. Data sent and received through the network interface has three options for security:

- Open: No security, plain text
- Obfuscated: Encrypted, but without TLS and a secure certificate and private key
- Secure: Encrypted with TLS and a certificate and private key

Each NetBurner device generates a default certificate and private key, which is stored in its internal flash memory, which can be used to secure the connection as a self-signed certificate. You can also use the device's private key to create your own externally self-signed certificate, or have a certificate issued by a recognized third party. The externally signed certificate can then be installed by overwriting the device's default key.

5.2.4.1 Cautions When Using the Jumper Recover Procedure on a Device

Each device has a location to insert a shorting jumper to reset the module in the event of a catastrophic software problem, such as downloading an application image that corrupts flash memory or crashes. This recovery procedure will erase all of flash memory and download a recovery image. A new private key and certificate will then be generated. In such a case, any external self-signed or third party certificates will no longer be valid since the private key will have changed. A new certificate will have to be created from the new private key.

5.2.5 Device Application Updates

Note

The AutoUpdate utility is no longer used for download application to your device.

You will notice there is an Update Application on the web page. Application are now uploaded using the web interface. Alternatively, there is a windows executable that can be run from the command line in `\nburn\pcbin\nbupdate.exe`. Application images are now `.bin` files, they are no longer `.s19` files. If building in NBEclipse the `.bin` file will be located in your workspace project's release folder. The format using `nbupdate.exe` is as follows: `nbupdate <application .bin file> <device IP>`

5.2.6 Eclipse and Development Tools Updates

Summary of new and improved features:

- Eclipse has been updated to the Oxygen version. It also takes advantage of multiple processor cores on the host computer to greatly speed up compile times. *Note:* if using the command line to build you can use `make -j` to use multiple cores.
- GCC has been updated to version 8.1.
- Each project now has its own NetBurner library. Now any system changes you make will be specific to the project, and each project can have its own modifications.
- The makefile implementation is now identical for NBEclipse and command line builds.

5.2.7 SSL/TLS

Improvements include:

- Moving from NetBurnerSSL to wolfSSL
- Support for a much broader range of ciphers
- Support for DER encoded certificates
- Support for server-side peer verification
- Support for onboard certificate generation
- Substantial improvements to SSL stability
- Support for client side certificate authority lists
- A cleaner interface for using certificate authority lists, as well as the ability to dynamically alter them during runtime
- Faster key generation for onboard certificate generation
- Lays the groundwork for future improvements
 - TLS v1.3
 - FIPS 140 Ready
 - DTLS

5.2.8 Notes on Porting From Previous Revisions

- Big endian vs. little endian. The ColdFire processors have a big endian architecture, while ARM processors are little endian. If you have code that manipulates 16-bit or 32-bit numbers that rely on a big endian format you will need to ensure it works for both types.
- The typical startup sequence of function calls: `InitializeStack()`, `GetDhcpIfNecessary`, `EnableAutoUpdate()`, etc have been replaced with a single call: `init()` which is declared in the header file `init.h`. You will need to remove header files such as:


```
#include <startnet.h>
#include <autoupdate.h>
#include <dhcpcclient.h>
#include <NetworkDebug.h>
```
- The real-time operating system has been modified to increase performance and capabilities. The header file `ucos.h` is now `nbrtos.h`. Any RTOS function calls that had UCOS in related names have been modified to reflect this.
- `UCOS_ENTER_CRITICAL` has been replaced with `NBRTOS_ENTER_CRITICAL`.
- The `StartHTTP()` function is now `StartHttp()` and `StartHttps()`.
- Callbacks have been added to replace the `MyDoGet()` and `MyDoPost()` functions. See [HTML Processing](#) for examples on usage.
- A new function has been added to wait for an active network connection, meaning the device has link. A typical startup sequence will be:


```
init();
WaitForActiveNetwork(TICKS_PER_SECOND * 10);
StartHttp(); // or StartHttps();
```

- Types such as `WORD`, `DWORD`, etc have been replaced with `uint16_t`, `int16_t`, `uint32_t`, `int32_t`, etc.
- Registering a name with a [DHCP](#) server is no longer done by assigning a string to `extern const char pDHCPOfferName`. It is now set by assigning a name to the configuration tree JSON object element: `CONFIG:SYS:NETIF:Ethernet:DeviceName`.
- Runtime calls to the network interface such as `EthernetIP()` have been removed. Please refer to the `ShowInterfaces` example for methods to obtain runtime interface data.

5.2.9 Porting I2C From MCF52xx Based Products to MODM7AE70

Applicable Platforms:

- MOD5234
- MOD5270
- MOD5272
- MOD5282
- PK70EX
- SB70
- SB70LC
- SB72

MCF52xx platforms have a single [I2C](#) peripheral, therefore the [I2C](#) driver API knew which signal pins to use and the set of functions to send/receive data only had to consider a single peripheral. ARM based platforms have many [I2C](#) peripherals, so a different type of driver/API is required. For example, with just a single [I2C](#) peripheral, an `I2CSend()` function is all that is needed. If there are many [I2C](#) peripherals, the driver must be able to send on any of them. This applies to all [I2C](#) functions: read, write, status, error detection, etc.

The most robust and scalable implementation is to use [I2C](#) peripheral C++ objects. You do not need to be a C++ programmer to use these objects in your application, other than to understand the format to call a member function of the object instead of a global C function call. For example, if in C you had a global function named `GetI2CStatus()`, it would now be a member function of a specific [I2C](#) object/peripheral and the call would be `MyI2CObject.GetI2CStatus()`. This method enables your application to handle multiple [I2C](#) peripherals, and the status function knows which peripheral to use because it operates on the object. Note that since the status function is called as a member function of the object, it is common to take that into consideration in the member function name, so it would likely be `MyI2CObject.GetStatus()`.

In this way you can easily handle multiple [I2C](#) peripherals: `MyI2CObject1`, `MyI2CObject2`, `MyI2CObject3`, etc. Your application code does not need to manage and continuously check for which peripheral is being used, such as having a C style array of [I2C](#) peripherals.

The [I2C](#) driver for MCF52xx devices in the NNDK 2.x tools has the interface listed below. Note that there is a bit of C++ even in the old driver in terms of the function prototypes. You are likely calling some of these functions without any parameters, or with less than all the parameters listed. In C++ default parameters are specified in the function prototype with the '=' sign. So you can call `I2CInit()` with no parameters in your application, and the default parameters in the function prototype will be used.


```

void I2CInit( BYTE slave_Addr = 0x08, BYTE freqdiv = 0x3C )

BYTE I2CSend( BYTE val, DWORD ticks_to_wait = I2C_RX_TX_TIMEOUT )
BYTE I2CSendBuf( BYTE addr, PBYTE buf, int num, bool stop = true)

BYTE I2CRead( PBYTE val, DWORD ticks_to_wait = I2C_RX_TX_TIMEOUT )
BYTE I2CReadBuf( BYTE addr, PBYTE buf, int num, bool stop = true )

BYTE I2CStart( BYTE addr, bool Read_Not_Write, DWORD ticks_to_wait = I2C_RX_TX_TIMEOUT )
BYTE I2CStop( DWORD ticks_to_wait = I2C_RX_TX_TIMEOUT )
BYTE I2CRestart( BYTE addr, bool Read_Not_Write, DWORD ticks_to_wait = I2C_RX_TX_TIMEOUT )

void I2CResetPeripheral()

bool I2CRXAvail()
DWORD I2CTXAvail()
BYTE I2CGetByte()

```

The **I2C** implementation in the 3.x tools for ARM platform uses the **I2CDevice** Class. For example, on the MODM7↵AE70, an **I2C** object to select an **I2C** peripheral to interface with an EEPROM can be declared as (code taken from the **I2C** Pic Kit example):

```

#define I2C_MODULE_NUM      0          // use I2C0/TW0
#define I2C_EEPROM_ADDRESS (0xA0 > 1) // Microchip 24LC0B EEPROM, 2Kbit (256 x 8)
I2CDevice EepromDevice(i2c[I2C_MODULE_NUM], I2C_EEPROM_ADDRESS);

```

The declaration of the **I2CDevice** creates an **I2C** object for **I2C** peripheral module 0 at the specified EEPROM **I2C** address. Creating the object also handles the initialization, so the equivalent of **I2CInit()** is not needed. Once we have created the object, all we need to do is call the member functions to control it in the format: **Eeprom↵Device.<member function>**. For example, to send/write a single byte value of 0x60 to address 0x01 of the EEPROM: **EepromDevice.writeReg8(0x01, 0x60);**. Note that in ARM terms, the **I2C** addresses are referred to as registers. **writeReg8()** refers to 8-bits.

You will also note that the return values use the C++ operator ":". Whereas the 2.x driver returned an unsigned 8-bit value (BYTE), the 3.x driver returns a typed value of **Result_t**. Using typed values will result in more robust application code since the compiler can detect when a return value is used incorrectly. the "I2C::" result types are (be sure to view the latest information in the 3.x NetBurner API manual):

```

I2C_RES_ACK      Acknowledged
I2C_RES_NACK     Not acknowledged
I2C_RES_ARB_LST  Arbitration listening
I2C_RES_BUSY    Bus is busy
I2C_RES_ARG     Bad argument

```

The corresponding functions to the MCF52xx **I2CDevice** driver are below. This is the recommended driver.

```

// Constructor to create object, also initializes the I2C peripheral. The pInterface
// parameter will be i2c[0], i2c[1] or i2c[2], representing the 3 I2C peripherals.
// The optional 3rd parameter is normally left as a default. It can be used to set the
// number of register address byte to send, from 0 to 3.
// Example: I2Cdevice EepromDevice(i2c[0], address);
I2CDevice(I2C & pInterface, uint8_t deviceAddress, uint8_t numAddressBytes = 1)

```

```

Result_t writeReg8(uint32_t reg, uint8_t dat)
Result_t writeRegN(uint32_t reg, uint8_t *buf, uint32_t blen)

```

```

Result_t readReg8(uint32_t reg, uint8_t &dat)
Result_t readRegN(uint32_t reg, uint8_t *buf, uint32_t blen)

```

Start, Stop and Restart are taken care of in the **writeRegN** and **readRegN** functions. In situations in which a restart was used for sending more than 1 byte as an address, the **numAddressBytes** parameter in the **I2CDevice** constructor can be used.

```

void resetBus()
void setup(uint32_t busSpeed) // change bus speed

```

If you wish to have more manual control over the **I2C** transactions, you can create an **I2C** object instead of an **I2CDevice** object:

```

// Create an I2C peripheral object. The module parameter for the MODM7AE70 can be 0, 1 or 2.
I2C(int module)

```

```

Result_t writeReg8(uint8_t devAddr, uint32_t reg, uint8_t data)
Result_t writeRegN(uint8_t devAddr, uint32_t reg, uint8_t *buf, uint32_t blen)

```

```

Result_t readReg8(uint8_t devAddr, uint32_t reg, uint8_t &data)
Result_t readRegN(uint8_t devAddr, uint32_t reg, uint8_t *buf, uint32_t blen)

```

```

void setup(uint32_t busSpeed)

```

```

Result_t DoTransaction(I2CTxn_t *pTransaction, bool bRepeatedStart = false);

```

```

void resetBus()

```

In NNDK 3.x there is also a third option. ARM refers to the [I2C](#) as a two-wire interface, or a wire interface in general. A [WireIntf](#) class is available for even more manual control. It was written for those familiar with Arduino drivers, but can be used for more detailed control in general. Please refer to the NNDK 3.x documentation for details.

5.3 Upgrade 5441x Platforms to 3.x

5.3.1 Introduction

The ColdFire 5441x platforms were created before NetBurner 3.x, but they can be easily upgraded. The primary difference is in the device configuration. For reverse compatibility with prior tool sets, all 5441x devices are shipped with a factory application built with the 2.x release so they work with the IPSetup and AutoUpdate utilities. Once you download an application created in NetBurner 3.x, the device will then have the advantages of the 3.x features. The upgrade procedure consists of first loading a conversion application as described below. Thereafter you can load 3.x applications directly.

5.3.2 5441x Product Models Numbers

- MOD54415
- MOD54417
- NANO54415
- SB800EX

5.3.3 3.x Update Procedures

There are 3 procedures that can be used depending on the production date of your 5441x device. For reverse compatibility all 5441x devices ship from the factory running a 2.x application. All are described in more detail in the following sections.

1. For devices manufactured January 2022 or later, the 2.x factory application can process 2.x or 3.x application images, so download your 3.x application using the AutoUpdate utility.
2. For devices manufactured prior to January 2022, use the AutoUpdate utility to download a 2.x to 3.x conversion application.
3. For devices manufactured January 2022 or later, abort to the Alternate Monitor and download either a 2.x or 3.x application. Note that this procedure is also used to revert back to a 2.x compatible device.

Attention

Once a device is running a 3.x application, you will no longer use the 2.x utilities IPSetup and AutoUpdate. Instead, you will use the methods described in the Device Discovery and Configuration section of this manual.

Attention

The NetBurner development tools for 5441x platforms produce both S-Record (.s19) and binary (.bin) application images. The 2.x compatible AutoUpdate utility mentioned in the migration section only uses .s19 files. The preferred file format going forward for all 3.x applications is the .bin format.

5.3.3.1 Procedure For a New 5441x Device Running The NetBurner Factory App

New devices from the factory are programmed with a 2.x factory application to be compatible with older designs. The application can process both 2.x and 3.x applications. The procedure is to identify the device and download a 3.x application.

- Connect the NetBurner device to the network and verify you can see the device's IP address with the Auto↔ Update utility.
- Select your 3.x application and update.

5.3.3.2 Procedure Using the 2.x to 3.x Conversion Application: xxxx-3p0-Update_APP.s19

This procedure can be used for a 5441x device that is currently running any 2.x application. The procedure is to identify the device, download a 2.x to 3.x conversion application, then download any 3.x application.

- Connect the NetBurner device to the network and verify you can see the device's IP address with the Auto↔ Update utility.
- Use the Autoupdate utility to download the appropriate conversion application image for your particular platform, located in the folder: `\nburn\platform\<platform name>\original\`. For example, a MOD5441x application name would be: "MOD5441X-3p0-Update_APP.s19".
- You will now be able to create and download 3.x applications on the device.

5.3.3.3 Procedure Using Alternate Boot Monitor

As of April 2021, the Alternate Boot Monitor has the ability to process a 3.x image file (older revisions must use the previous procedure by loading the conversion application). Alternate Boot Monitor revisions must be the revision shown below or later:

Platform	Revision
MOD5441x	1.11
NANO54415	1.05
SB800EX	1.05

- Connect to the debug/console port of the device with a serial terminal such as MTTTY
- Reset or power cycle the device (or use the recovery jumper)
- Press the 'A' key to abort when prompted (or use the recovery jumper)
- Use the AutoUpdate utility to download your 3.x application image

Note

If you have a device with an earlier Alternate Boot Monitor revision and wish to update it (the Alternate Boot Monitor), please contact NetBurner Support at <https://support.netburner.com>.

5.3.4 Procedure to Revert Back to 2.x

If you wish to revert back to 2.x, there are two methods to abort the boot sequence and enter the Alternate Boot Monitor:

1. Using the serial port interface
2. Using the hardware reset jumper

5.3.4.1 Serial Port Procedure

- Connect to the RS-232 debug/console port of the device with a serial terminal such as MTTTY
- Reset or power cycle the device
- Press the 'A' key to abort when prompted to abort to the Alternate Boot Monitor
- Use the AutoUpdate utility to download your 2.x application image

5.3.4.2 Hardware Reset Jumper Procedure

- Insert the hardware reset jumper
- Reset or power cycle the device to abort to the Alternate Boot Monitor
- Use the AutoUpdate utility to download your 2.x application image

5.3.5 Note on IPSetup utility and 3.x Devices

The windows IPSetup utility will still identify the IP address of a 3.x device. However, it cannot configure the device. The IPSetup device window designates these devices with a prefix of NEWCONFIG.

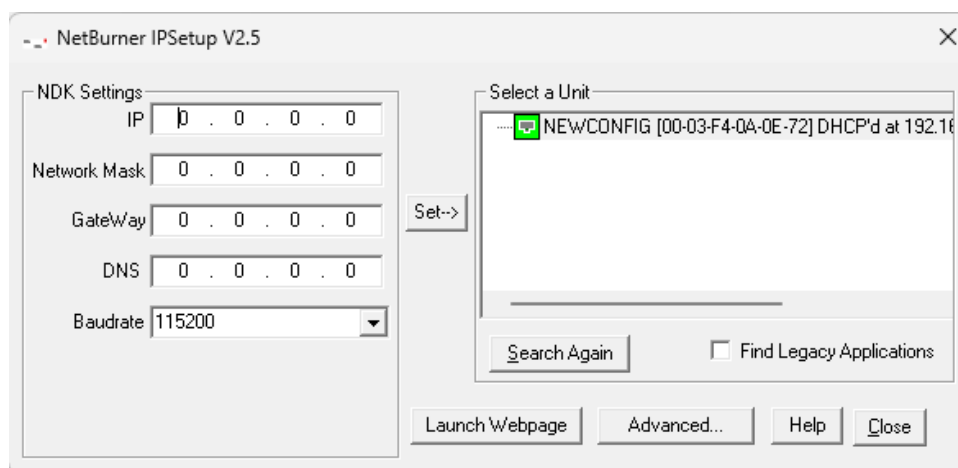


Figure 5.1 IPSetup with 3.x Application

Chapter 6

NBEclipse

There are three different ways to develop with the NetBurner Tools:

1. Install NBEclipse, which provides project management, a make system, and an editing environment.
2. From the command line using the make utility. Each example in the `\nburn\examples` folder provides an example makefile.
3. Use your own editing environment, and have that environment call the makefile for the project.

The documents in this section will describe how to use the NBEclipse environment and command line.

[Getting Started Guide](#)

[Add a Library to a Project](#)

[Change The IP Targeted By A Project](#)

[Change CompCode Settings for EDFS](#)

6.1 Getting Started Guide

6.1.1 Introduction

6.1.1.1 How to Use This Guide

The goal of the NetBurner Eclipse Getting Started Guide is to familiarize new users with the NetBurner Eclipse (NBEclipse) interface and basic operation. NBEclipse is based on the standard Eclipse release with additional features to support development with NetBurner hardware, such as the ability to locate your NetBurner device on the network, and download an application to the device's flash memory. Topics include:

- How to create a new project
- Import, compile, and execute example programs
- Debug an example program
- Use the application wizard to create a new program
- Various tips on useful features of the NBEclipse environment

6.1.1.2 Source Code for Example Programs

Source code for the examples in this document are located in the `\nburn\examples` directory of your NetBurner tools installation.

6.1.1.3 Hardware Setup

This document assumes you have a working hardware platform on which you can run the examples. Your NetBurner development kit hardware is pre-programmed with an example application at the factory. Before executing the examples in this document, it would be a good idea to connect your NetBurner device to your computer and verify you have serial and network communication with the device as described in the NNDK Quick Start Guide.

6.1.1.4 Software Installation

NetBurner software tools run on Microsoft Windows and OSX. When you receive your development kit there will be a bright red card with a keycode. Register this keycode at <https://support.netburner.com> to download the development tools.

6.1.1.5 Upgrading From a Previous Installation

Do not install over an existing version when upgrading from a previous version of the NetBurner tools. The upgrade requires that you either rename or uninstall the existing installation directory. The recommended procedure is as follows:

1. Backup any existing projects you want to save, especially if they are in the NetBurner tools installation directory.
2. Rename the existing tools installation folder. For example, if upgrading from revision 2.8.8, rename `\nburn` to `\nburn2p8p8`, then install the new version to `\nburn`. This way you can always get back to the previous installation and environment by renaming `\nburn2p8p8` to `\nburn`.

6.1.1.6 Java Installation

Newer versions of NBEclipse (3.3.3 and higher) bundle Java internally and so no separate installation is required.

Older versions of NBEclipse (3.3.2 and lower) require the 64-bit Java Runtime Environment (JRE) revision 1.8 or higher. Direct links to recommended versions are provided on the NNDK Download page. Download either the Java SE (Standard Edition) Development Kit (JDK) or Runtime Environment and follow the installation prompts. You may need to restart your computer after installation is complete.

Note

NBEclipse tools version prior to 3.0 require the 32-bit JRE.

6.1.1.7 Debug Port

Throughout this guide, we will refer to the "serial debug port". The serial debug port is one of the RS-232 ports that can be used to interact with your NetBurner device in the example programs. Stdout, stdin, and stderr are mapped to the debug port by default, so when you use functions like printf(), scanf(), gets(), they read and write to the serial debug port. Mapping of the stdio is configurable, as you can disable the serial debug port and use it as a general purpose UART, or reassign the stdio file descriptors to use other serial or network interfaces. Debugging in NBEclipse on network-enabled platforms will occur through the Ethernet connection, while debugging on non-network platforms will use the serial port.

6.1.2 Platform Overview

6.1.2.1 NetBurner Development Platform Choices

Your NetBurner Network Development Kit provides two options for development:

1. NBEclipse Integrated Development Environment (IDE)
2. Command line tools using the GNU make utility.

The choice of what platform you wish to use depends on personal preferences. The command line tools use makefiles, which are included for every example program and can be used as a starting point for your own makefiles. The make utility can also be invoked from within other IDEs and code editors such as Microsoft Visual Studio.

6.1.3 Project Management

The following points are useful to users who have not used a similar type of project manager. There are three primary methods to maintain and build a project:

1. Use NBEclipse to manage the project with the project files located in the NBEclipse workspace. This is the recommended method.
2. Use NBEclipse to manage the project with the project files located in an existing directory outside the NBEclipse workspace.
3. Configure NBEclipse to use an external makefile that you maintain.

6.1.3.1 NBEclipse Key Points

1. NBEclipse uses a "workspace" in which projects are created and source files are copied.
2. All source code files must reside in the project's "src" folder.
3. If using the web server, all HTML and associated files must reside in the project's "html" folder.
4. All recognized files in a project directory will be compiled. Recognized files have extensions of .c, .cpp, and .s (.s is an assembly language file).
5. The Project Explorer tab on the left side of the NBEclipse IDE is a navigation window like a file browser, it is not a file project manager. All recognized files will be built as part of the application. If you have non-project related files in your project directory and wish to see only the project files in the tab, you can select a filter to display only source code files.
6. To download an application to your NetBurner device, select "Run as NetBurner Application" from the project's Build Targets folder.
7. Application build output files such as .o, .elf, .map, .bin and .s19, and .bin will be written to project sub-directories named "Release" and "Debug" that correspond to the type of build executed. The .bin and .s19 files are the application image files.
8. The keyboard shortcut `<ctrl> b` will build the active project.
9. The keyboard combination of `<ctrl> <space>` invokes function auto-complete.

6.1.4 Creating Projects and Importing Files

This is the most common type of project creation. It will be used to build and run example programs, and is a good starting point for your own projects. By "empty project" we are referring to creating a project without running the Application Wizard to auto-generate a minimal application. After we create the project we will use the project import feature to import source code files from an example into the project.

1. Create an empty project.
2. Import source files by right-clicking on the project name and using the Import feature from the drop down menu.
3. Build the application and download to your NetBurner device.

6.1.4.1 Create a New Project

From the main menu, select **File->New->Project**:

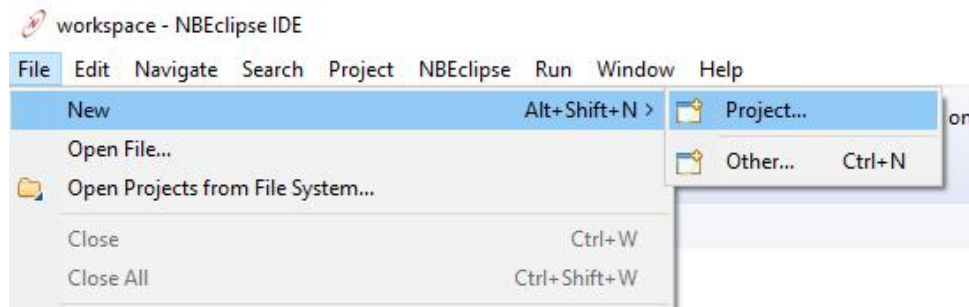


Figure 6.1 Create a New Project

6.1.4.2 Select the NetBurner Project Type

Select the NetBurner project type, then select Next:

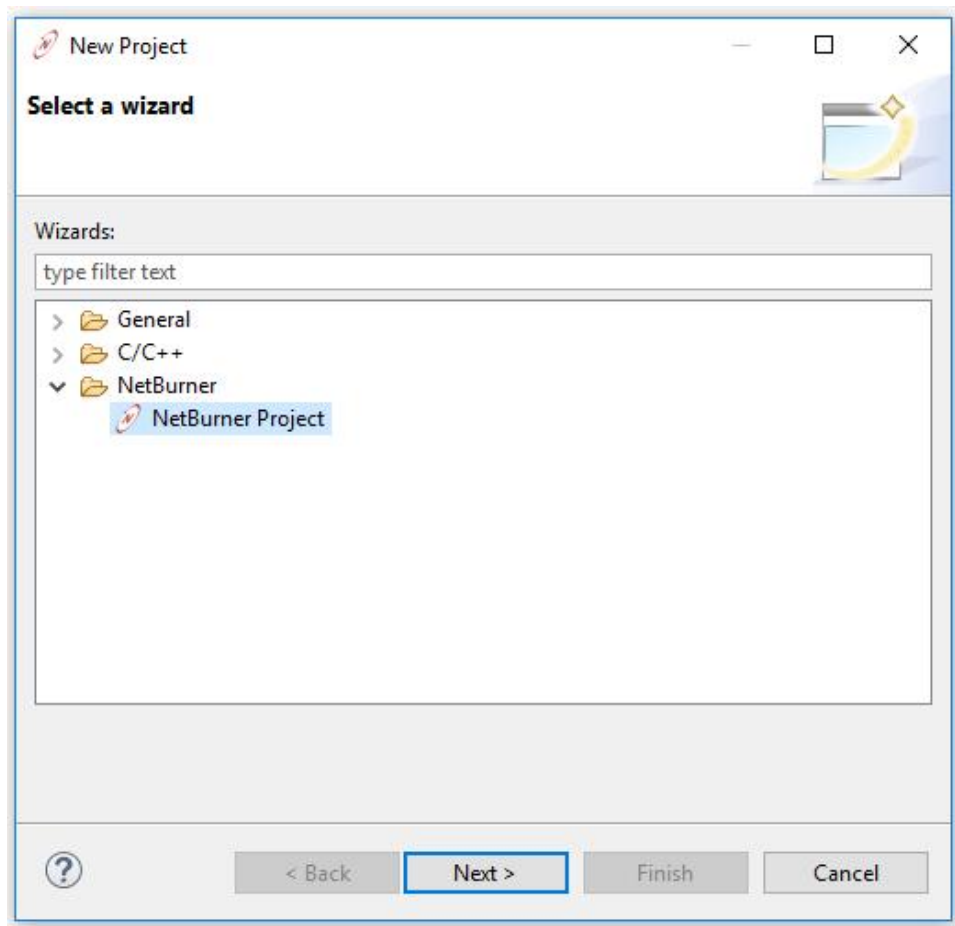


Figure 6.2 Select Product Type

6.1.4.3 Specify Project Name and Executable Option

Enter your project name in the Project Name field (it is best not to use spaces with gcc tool chains). In this example we entered "SimplHtml", since that is the example we will import. Make sure the Executable is set to: "NetBurner C++ Project".

Warning

Be sure to select NetBurner Device Executable, do not select the C/C++ option.

Note

You can create a project in a directory outside of your workspace by unchecking the Use Default Location checkbox, and using the Browse button to select the directory containing the source files.

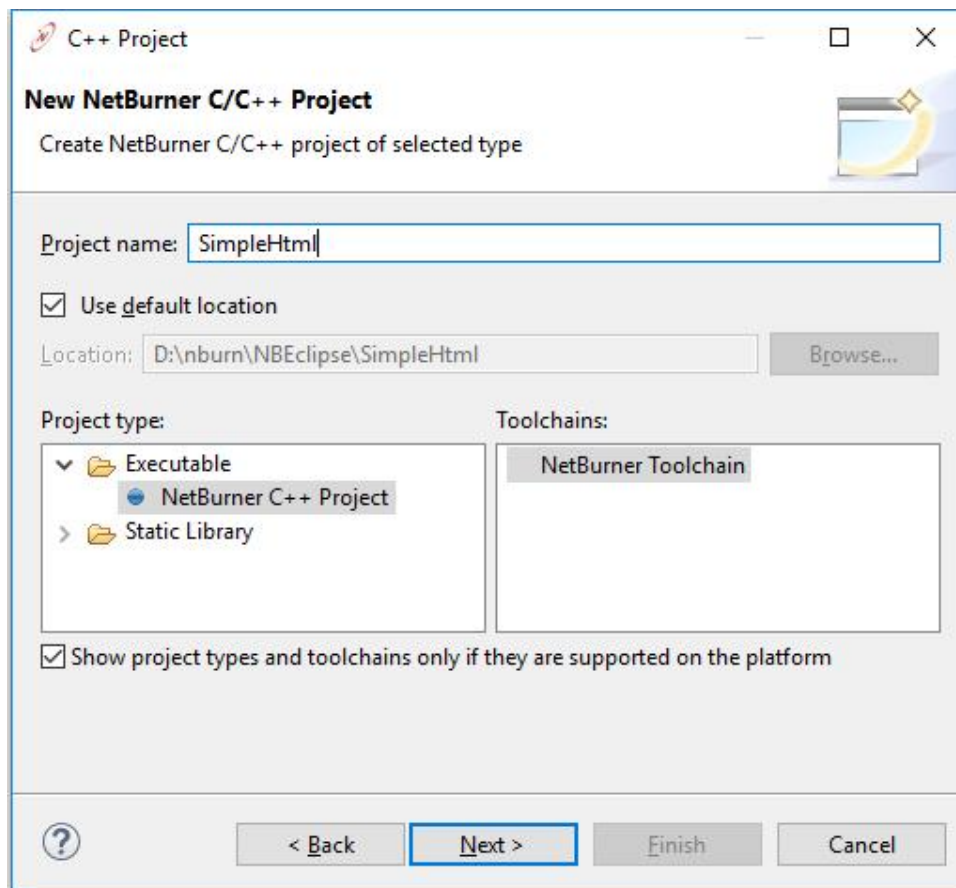


Figure 6.3 Select NetBurner Device Executable

6.1.4.4 Project Configurations

The default options for the project configurations are both Release and Debug. Leave the defaults and select Next.

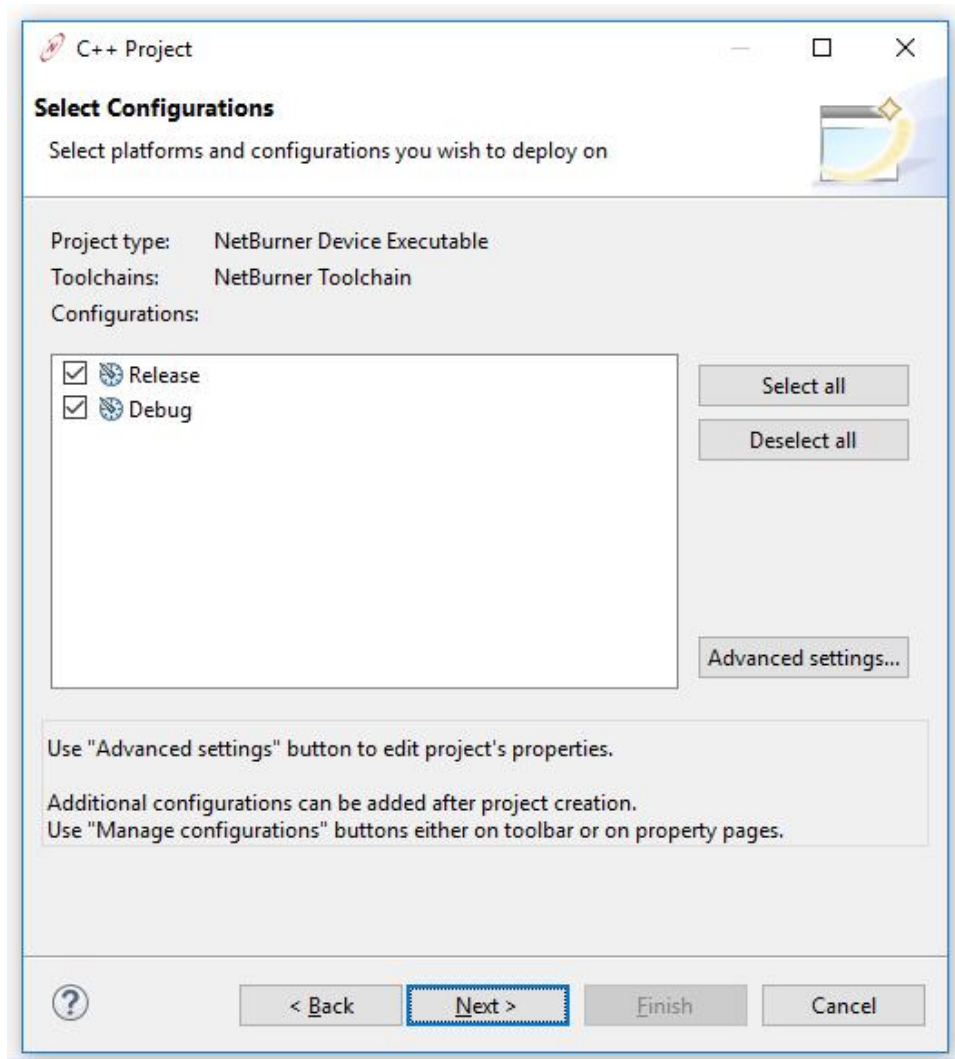


Figure 6.4 Project Configurations

6.1.4.5 Select Your NetBurner Target Device

There are two ways to select your target NetBurner device:

1. Click on the Search button to bring up a list of NetBurner devices on your network (device must be operational).
2. Manually: Click on the Target Platform drop down box to select your platform, then type in the IP address.

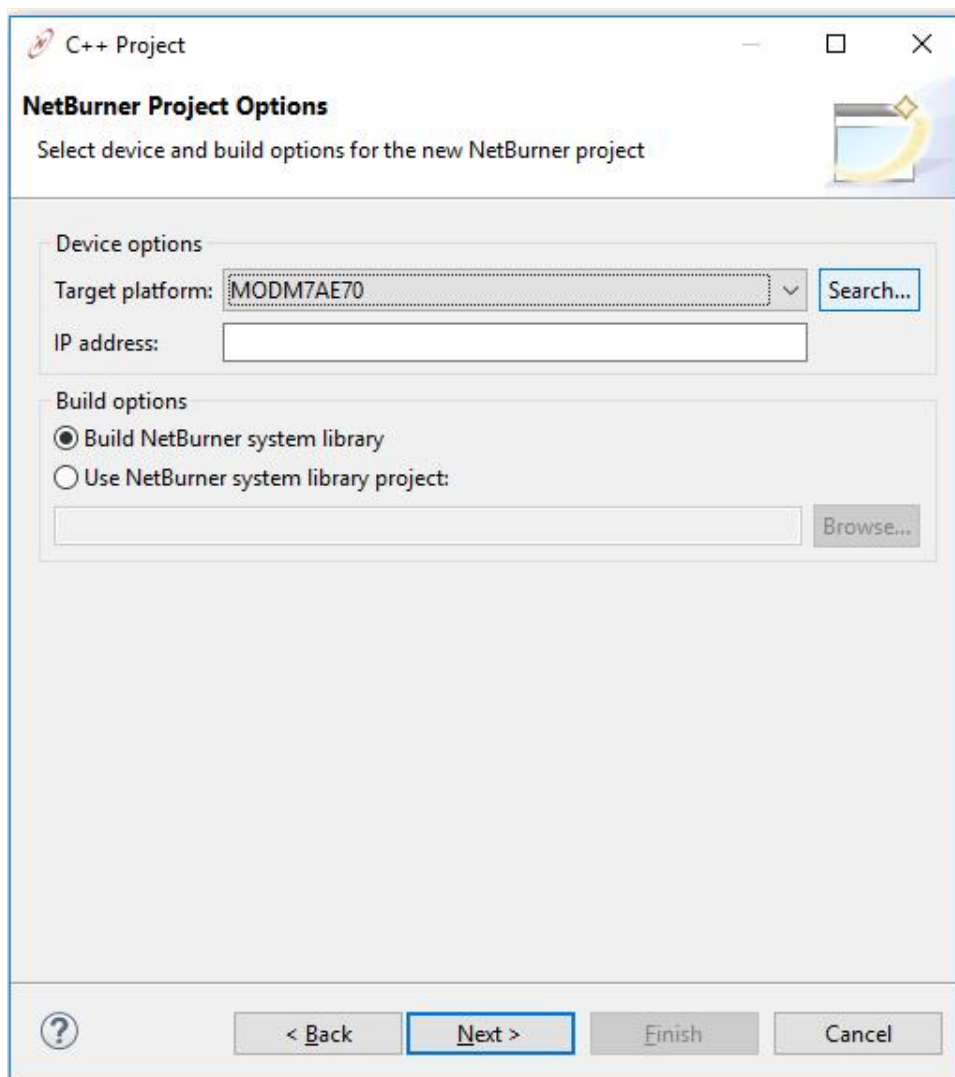


Figure 6.5 Search for devices

When using the Search feature, select the device in the list that corresponds to the MAC address printed on the label of your NetBurner device. In the image below we can see the device type, MODM7AE70, it's MAC address, and it's IP address. It will then fill in the platform and IP address for you. When done select Next.

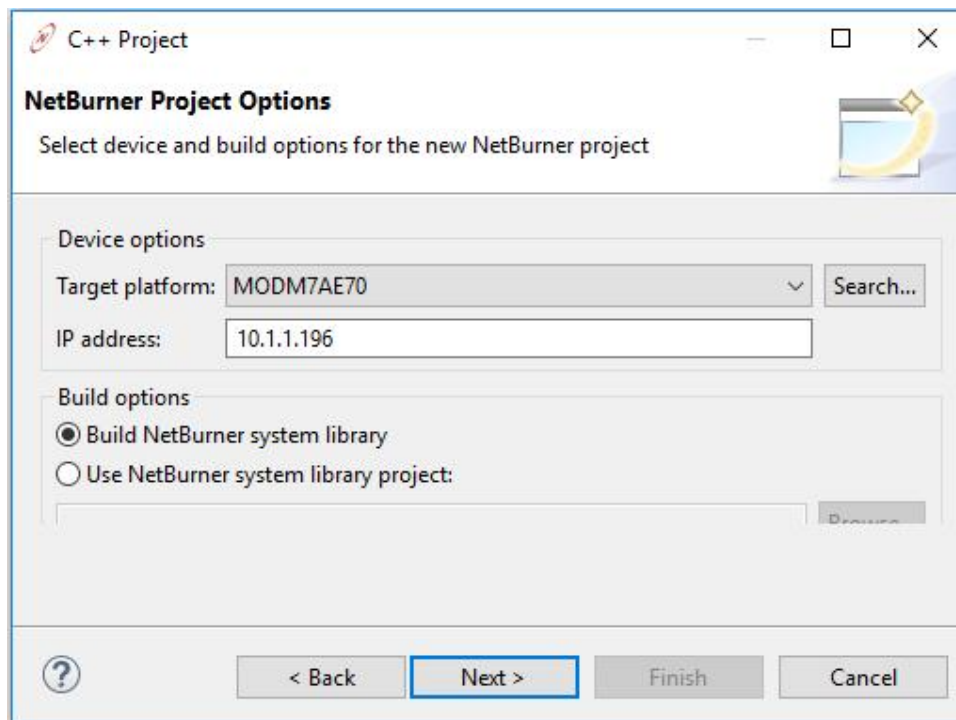


Figure 6.6 Successful search entry

6.1.4.6 Application Wizard

Since we are creating an empty project and will be importing our own source files we do not want to select any items here. However, if you wish to generate a minimal application using the Application Wizard, select both Standard Initialization and Web Server. This will generate a main.cpp source file to use as a starting point for your application.

Select Finish to complete the project creation.

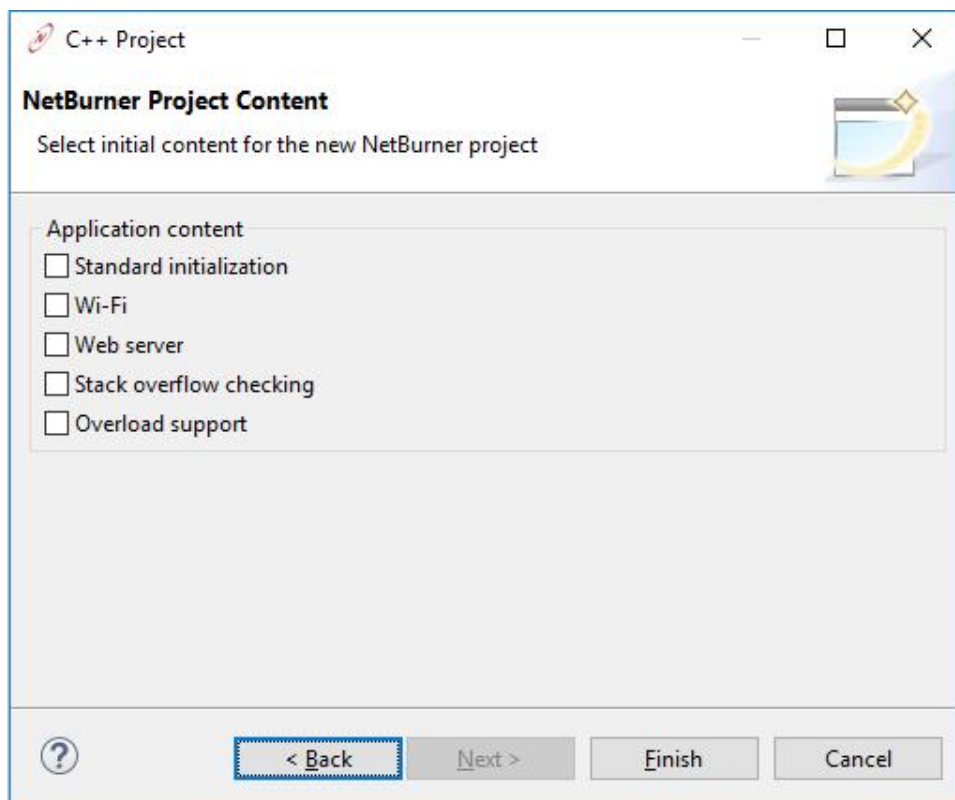


Figure 6.7 Skip Application Wizard code generation

6.1.4.7 New Project Created

You will now see your new project in the Project Explorer pane. At this point we have a project, but still need to import our source files (if you did not use the Application Wizard).

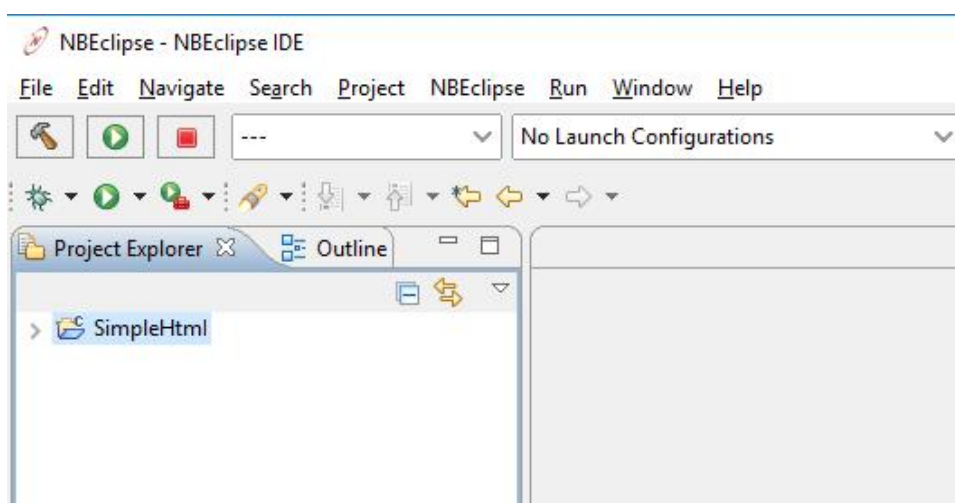


Figure 6.8 A new project has been created

6.1.4.8 Import Example Source Files to the Project SRC Folder

The next step is to import the source files of an example. NBEclipse uses a make utility to manage projects, and will attempt to build all source files in your project directory. Do not import non-source code files, such as .o, .elf, .map, .mk, auto-generated htмлdata.cpp files, etc. NBEclipse will store all output files similar to these in project subdirectories named "Release" and "Debug" corresponding to the build type performed.

For this demonstration we will use "`\\nburn\\examples\\Web\\SimpleHtml`". The first step is to right-click on your project in the NBEclipse Project Explorer pane:

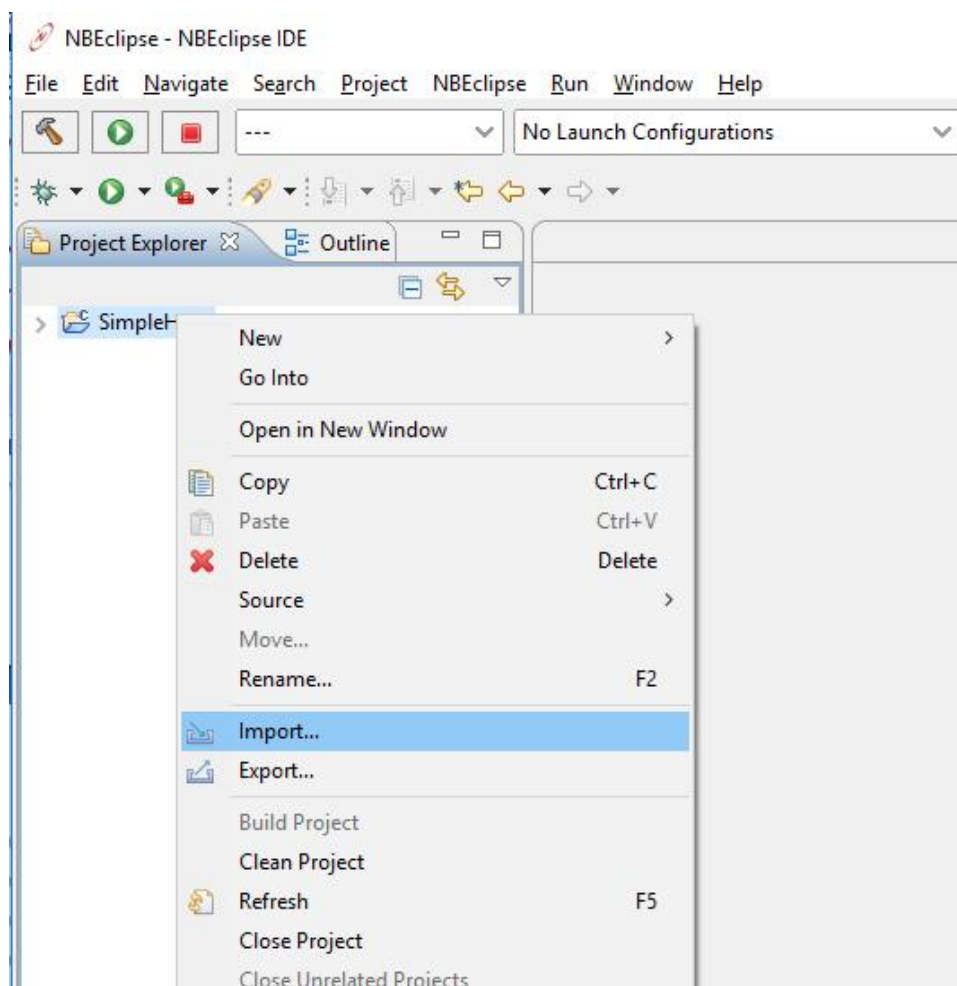


Figure 6.9 Select Import

Next select the General->File System, then select Next:

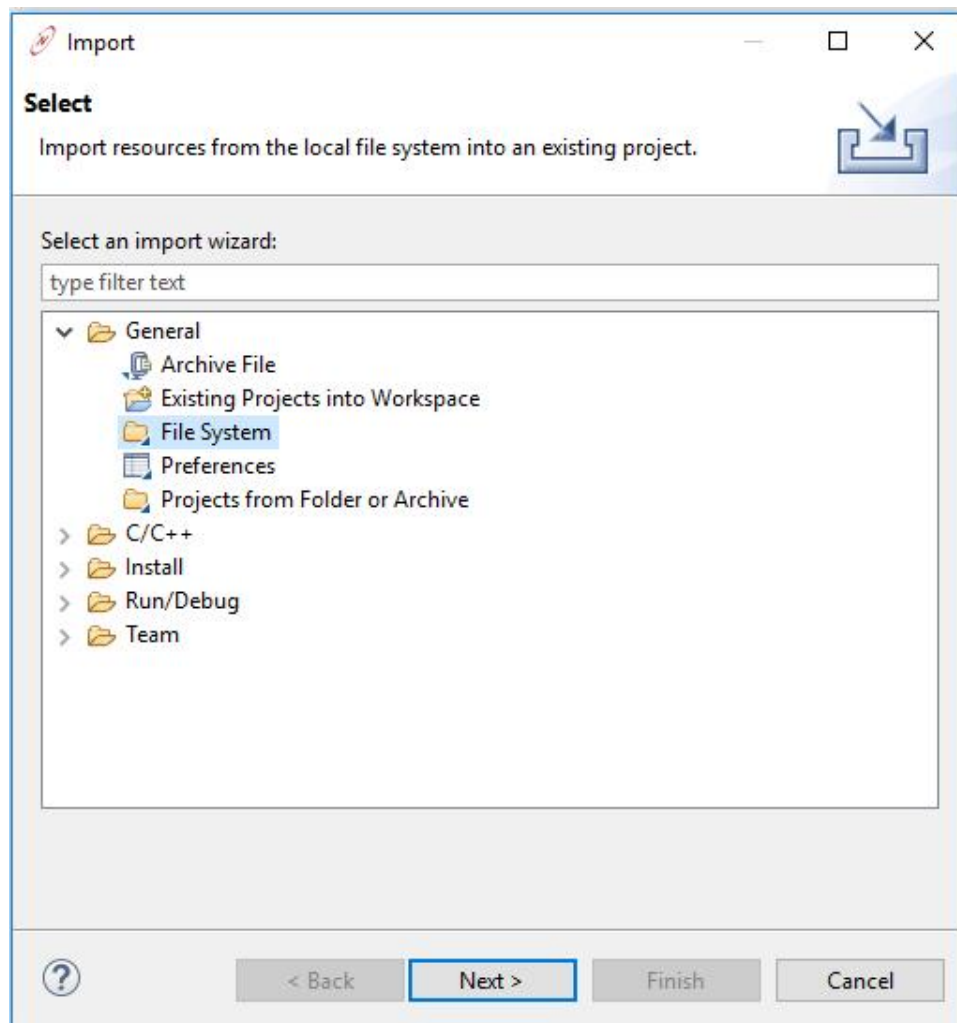


Figure 6.10 Select File System

Select the Browse button to bring up a file selection window, navigate to `\nburn\examples\Web\SimpleHtml`, and select the OK button:

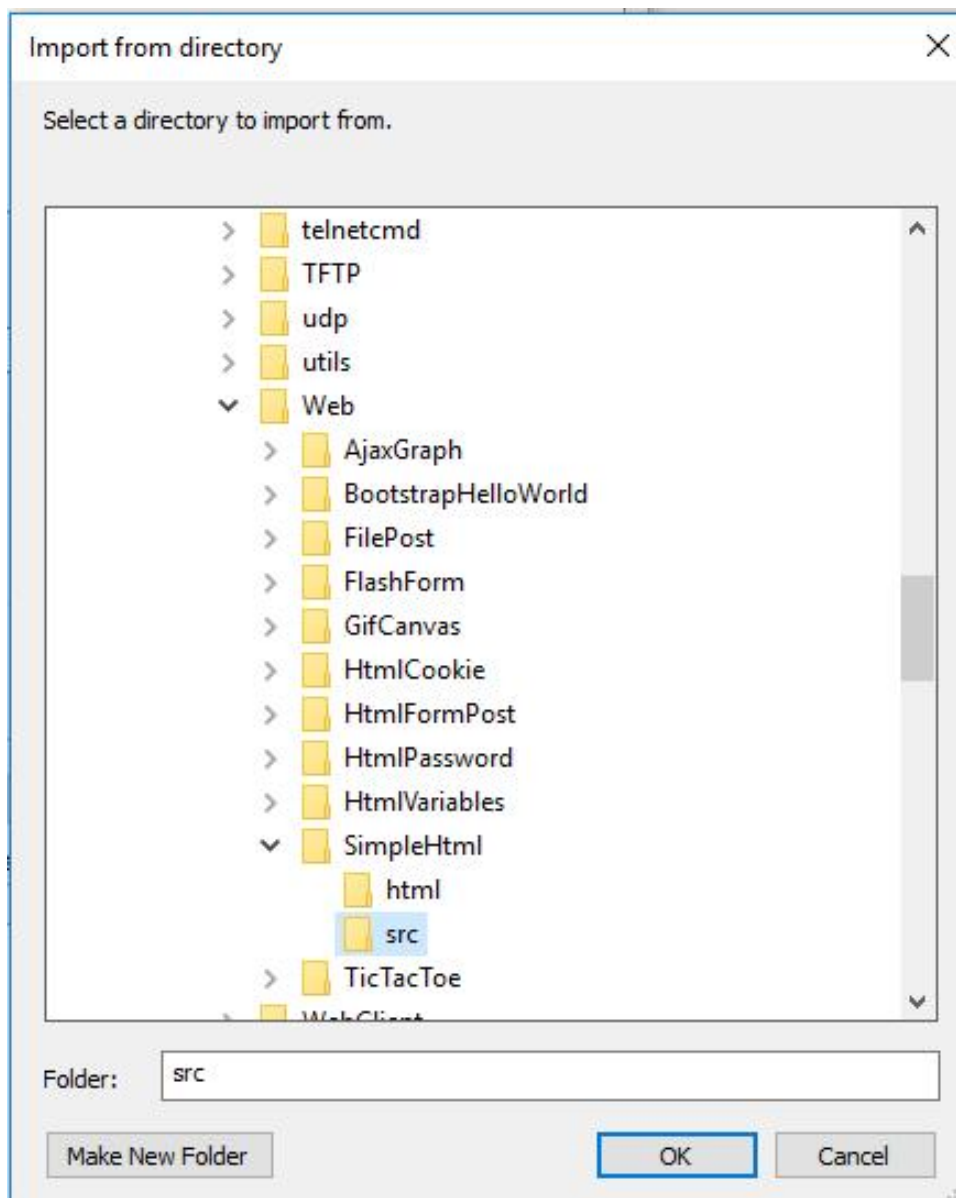


Figure 6.11 Select the Simple HTML Example

Once back at the File System dialog box, click on the ">" symbol next to the SimpleHtml name to open the tree structure. You can see that there is a folder named src and a folder named html. We want the files in both folders, so select the checkbox next to the name "SimpleHtml", which will select all folders underneath it. Note: the makefile you see is only used for command line builds, but it does no harm to import it as well.

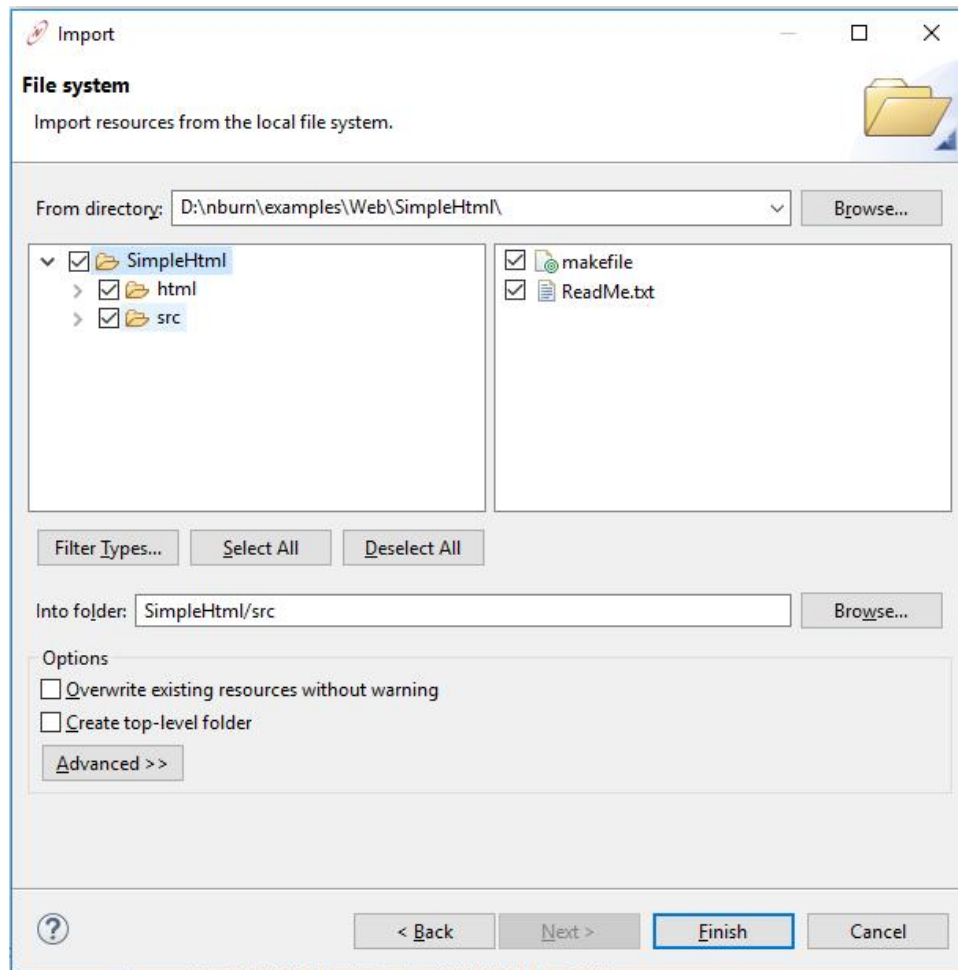


Figure 6.12 Select all SimpleHtml files for import

Select Finish, which will import the files and automatically build the project. You can view the status of the build process in the console window, located in the lower right corner of your screen. In the Project Explorer, click on the ">" symbol next to SimpleHtml. Some important items of note:

- Build Targets has two functions: downloading code to your device, and rebuilding the system libraries if needed.
- The src folder contains all your source code files, such as main.cpp.
- The html folder contains all your web based code.
- The overload directory is used to override any system library files. See the example in \nburn\examples\OverloadDirectory for more details.

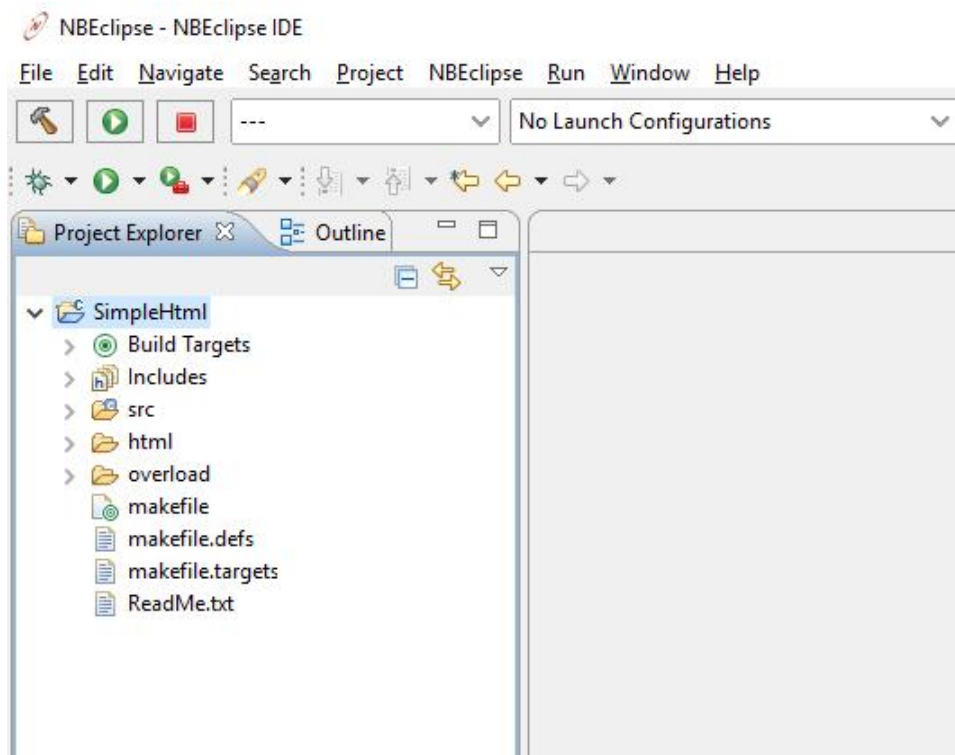


Figure 6.13 Import complete

6.1.4.9 Open your main.cpp source file

Clicking on the ">" symbol next to the src folder, you can open main.cpp in the editor

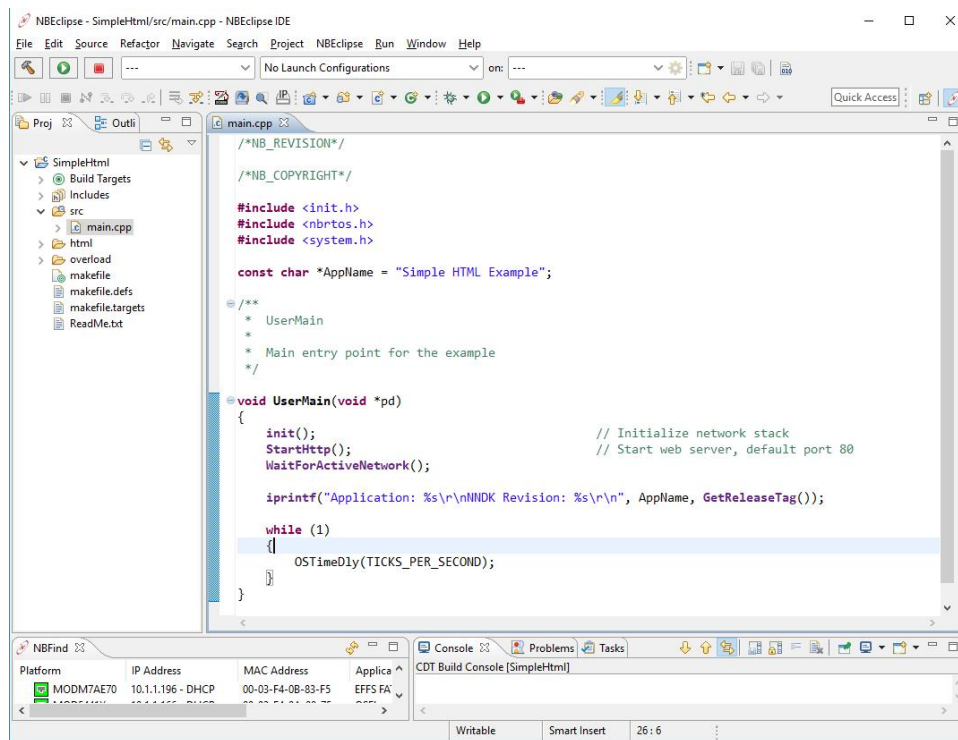


Figure 6.14 Open main.cpp

6.1.4.10 Project Import Complete

At this point you have a project complete with source code files. You can use the editor to modify both src and html files. If you wish to open a html file in text mode to do things like add dynamic content tags such as CPPCALL, VARIABLE, etc, right-click on the html file and use the Open With option to select the text editor:

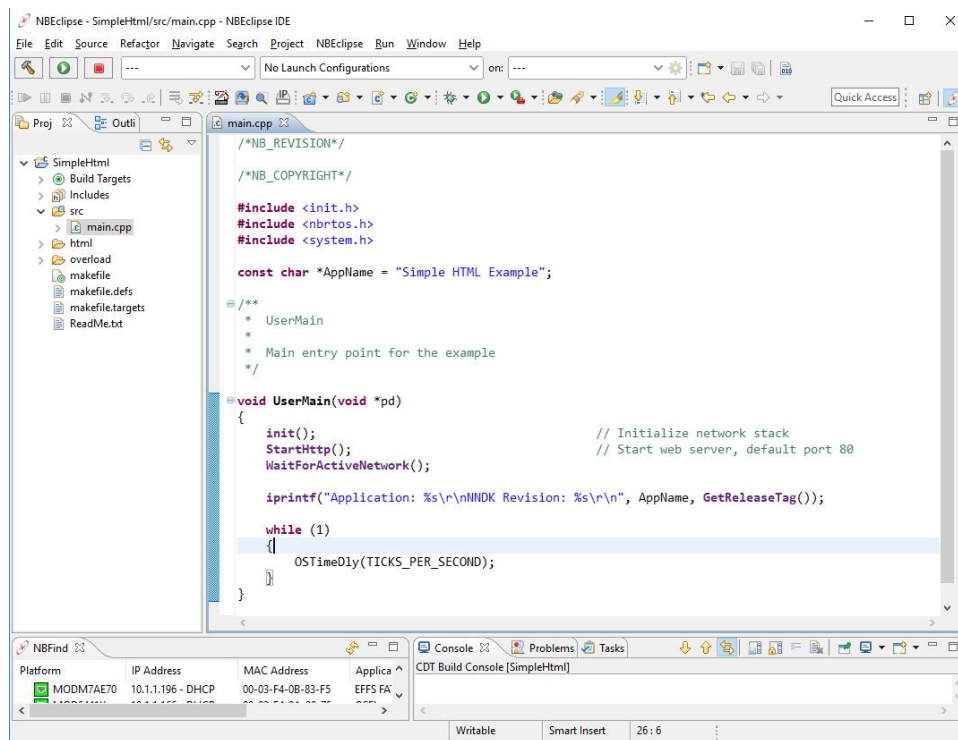


Figure 6.15 Open HTML files in text mode

By default, any time you modify and save a source file, the system will automatically build the application. If you prefer to turn off the auto-build option, to build the project: from the main menu select Project->Build Project, or use the `<ctrl> b` keyboard shortcut.

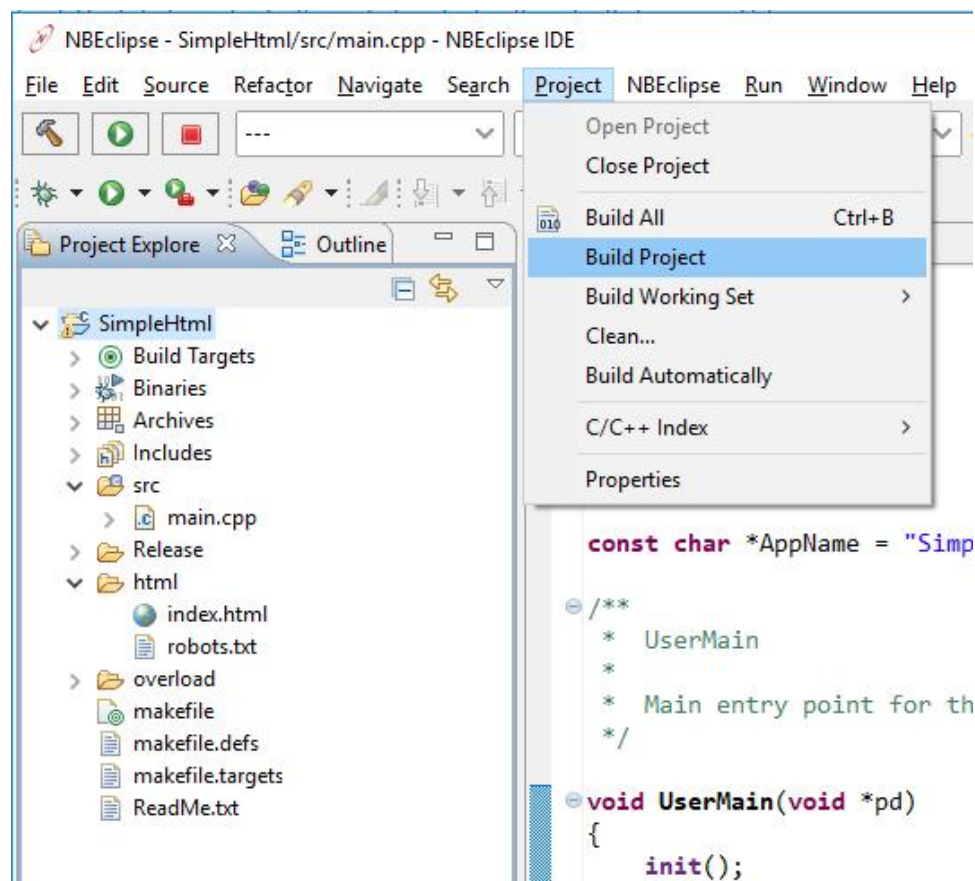


Figure 6.16 Build the Project

6.1.4.11 Download the Application to Your Device

Now that an application image has been built, the next step is to download the binary image to your NetBurner device. NBEclipse can simultaneously handle multiple NetBurner hardware platforms of the same or different types. For example, multiple MODM7AE70 core modules, or a mix with some MOD54415 modules. Targeting a project to a specific hardware device, is accomplished by creating a Run Configuration for release mode project builds, or a Debug Configuration for project debug builds. An important part of such a configuration is associating the device's IP address with the project so NBEclipse knows where to download the project's binary image.

When you create a new project and select the download option for the first time, NBEclipse will create a Run Configuration for you (Run and Debug Configurations can also be created manually). First select your project by left clicking on it in the Project Window:

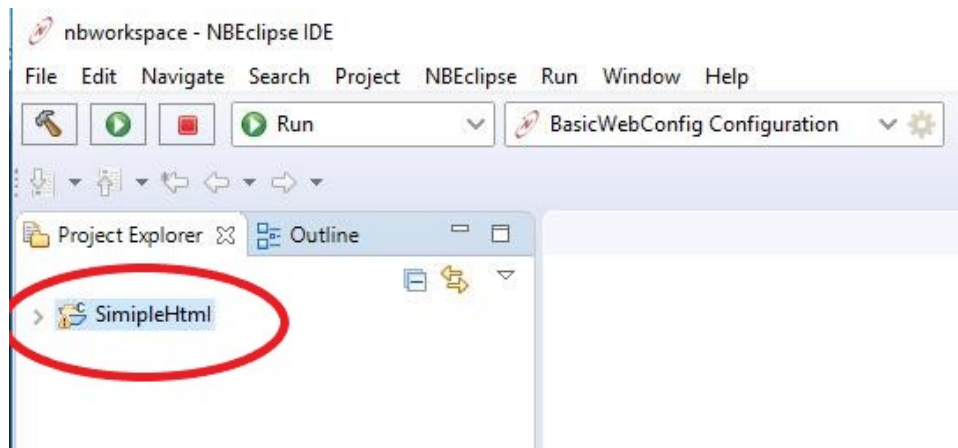


Figure 6.17 Select Project

Then click on the Run button in the upper right corner of NBEclipse:

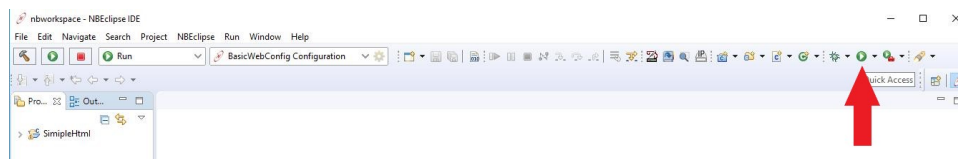
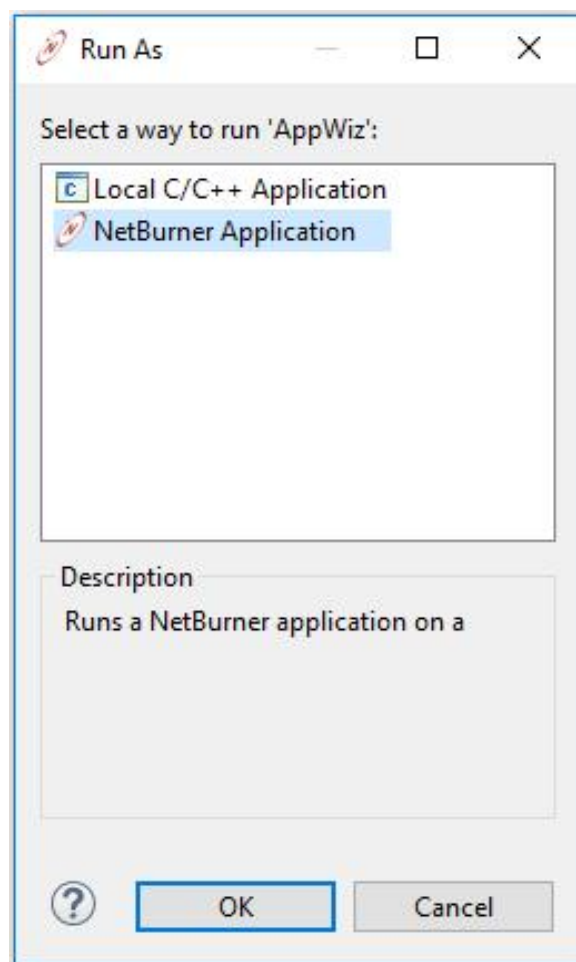
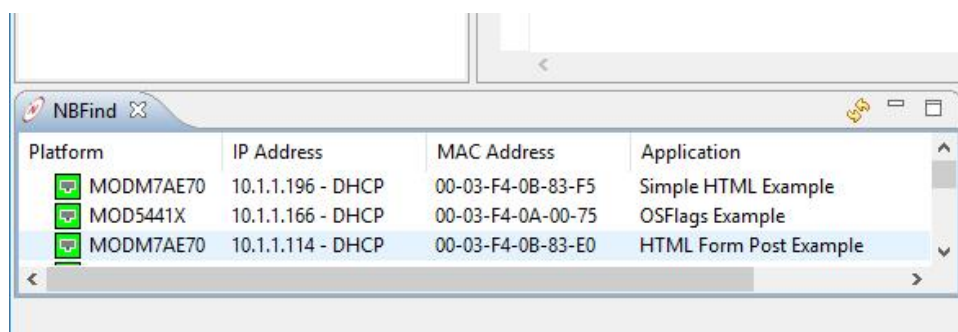


Figure 6.18 Select Run Button

The Run As dialog box will appear. Select NetBurner Application, which will run the application on the remote system: When you click the OK button, NBEclipse will download the binary image to the remote device and reboot it.

**Figure 6.19 Select Run Application**

The application should now be running on the device. If you do not remember the IP address of the device, you can find it using the "Find" window in the lower left of NBEclipse. In the screenshot below there are 3 devices. The device running SimpleHtml has IP address 10.1.1.196.

**Figure 6.20 Find Device IP**

6.1.4.12 View the Device Web Page

Opening a web browser and typing 10.1.1.196 in the URL field:

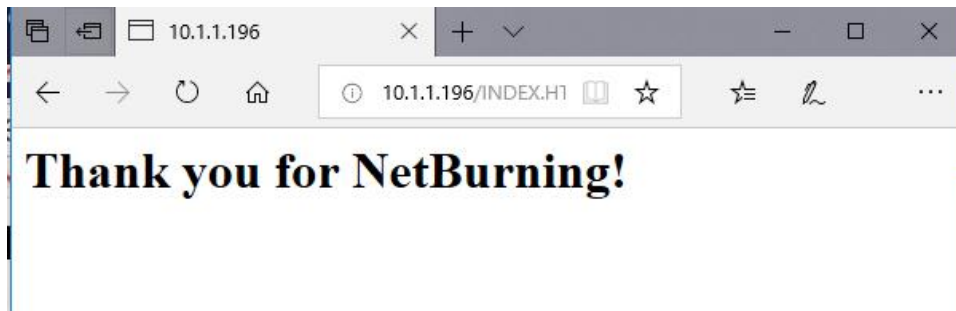


Figure 6.21 HTML Page

If you now look at the upper left side of NBEclipse, you will see the configuration set to Run, and the project set to SimpleHtml for the configuration.

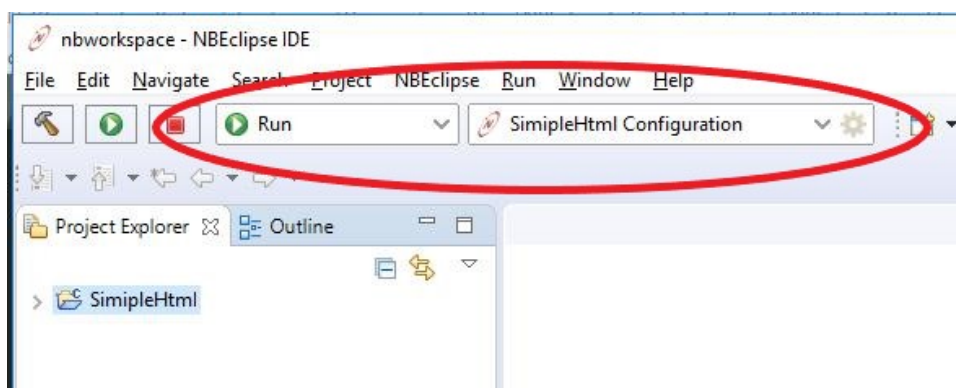


Figure 6.22 Run Configuration

6.1.4.13 View Serial Port Status Messages with MTTTY

All NetBurner examples make use of the debug/console serial port to provide status messages. You will need to have a serial connection to your device either through USB or the DB9 on the development board, depending on your hardware configuration. You can then use the MTTTY serial terminal application to view the serial output. To open MTTTY:

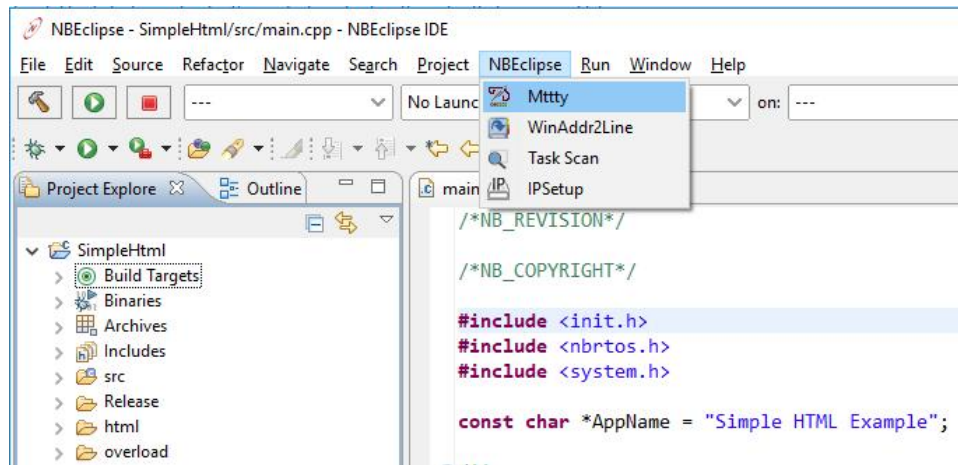


Figure 6.23 Open MTTTY

When the device boots the SimpleHtml application you will see:

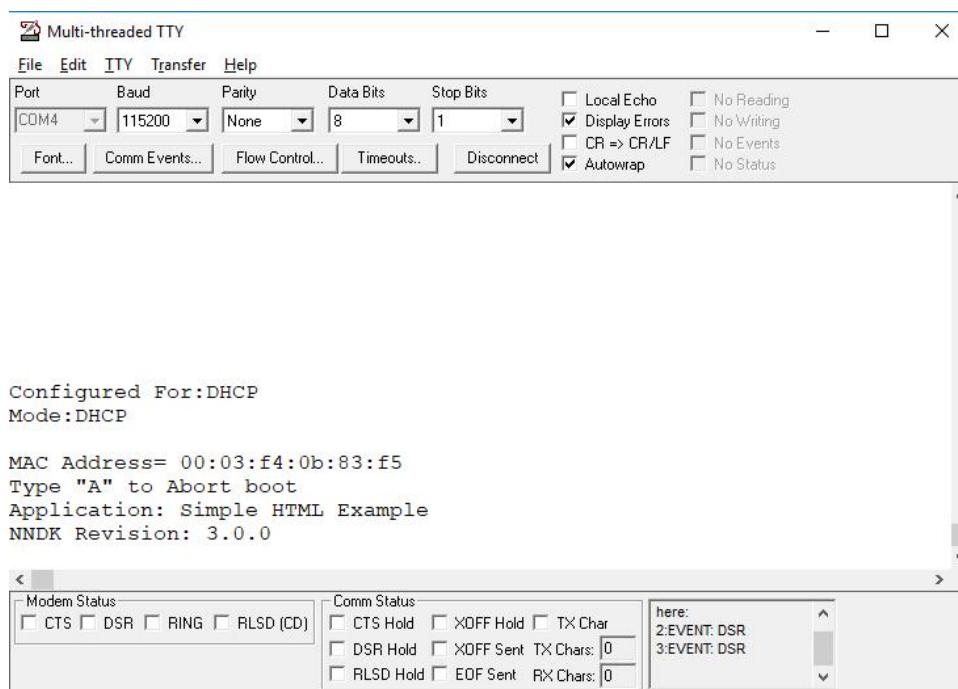


Figure 6.24 MTTTY Output

6.1.5 Debugger

In the previous section we created and ran a project in release mode. To use a debugger, the application must be built in debug mode. All debuggers require the code be built in a different manor than a release build:

- Code optimization is disabled
 - If the platform uses external SDRAM with the internal SRAM acceleration for stacks, buffers and variables, it is disabled
- NBEclipse uses a network debugger. This means that an application must be able to at least boot through network initialization so the NBEclipse agent can connect to the GDB stub on the target device.

6.1.5.1 Starting a Debug Session

We will pick up right where we left off for the SimpleHtml example project in the previous section. First, lets open main.cpp and add a variable. UserMain() is shown below:

```
void UserMain(void *pd)
{
    init();                      // Initialize network stack
    StartHttp();                 // Start web server, default port 80
    WaitForActiveNetwork();

    iprintf("Application: %s\r\nNNDK Revision: %s\r\n", AppName, GetReleaseTag());

    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

Add two lines of code to create an integer and increment it:

int i = 0; and i++;

So UserMain() becomes:

```
void UserMain(void *pd)
{
    init();                      // Initialize network stack
    StartHttp();                 // Start web server, default port 80
    WaitForActiveNetwork();

    iprintf("Application: %s\r\nNNDK Revision: %s\r\n", AppName, GetReleaseTag());

    int i = 0;                    // <--- Declare variable
    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND);
        i++;                     // <--- Increment variable
    }
}
```

Previously for the Run Configuration and download we clicked on the green play button in the upper right corner of NBEclipse. This time we will click on the debug button, which is next to it on the left:



Figure 6.25 Select Debug Configuration

It will take a few seconds for the debugger to connect, and the NBEclipse Perspective will change from the NetBurner perspective to the Debug perspective:

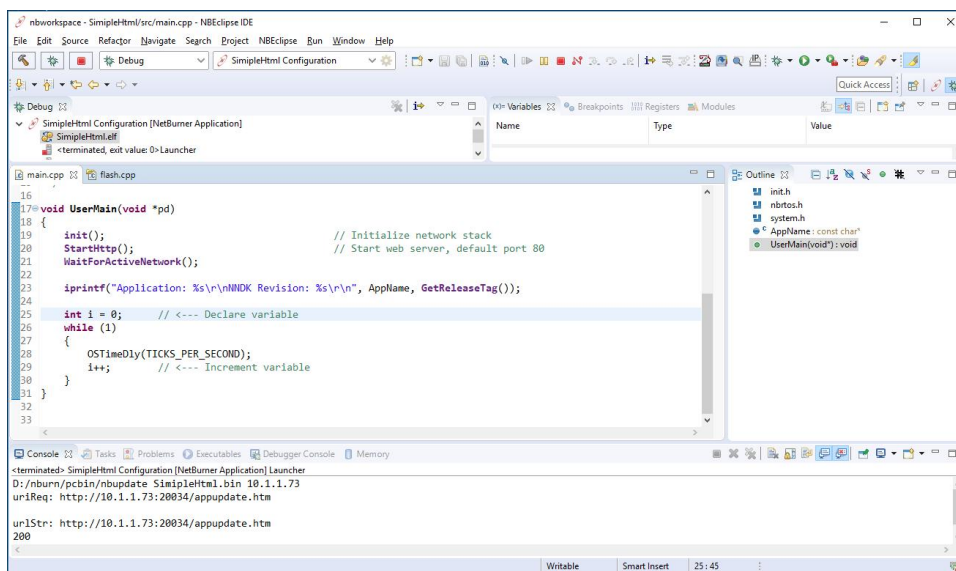


Figure 6.26 Debug Perspective

At this point the code is running. To set a breakpoint on our new variable, double click to the left of the line of code. In this example it is just to the left of line 29:

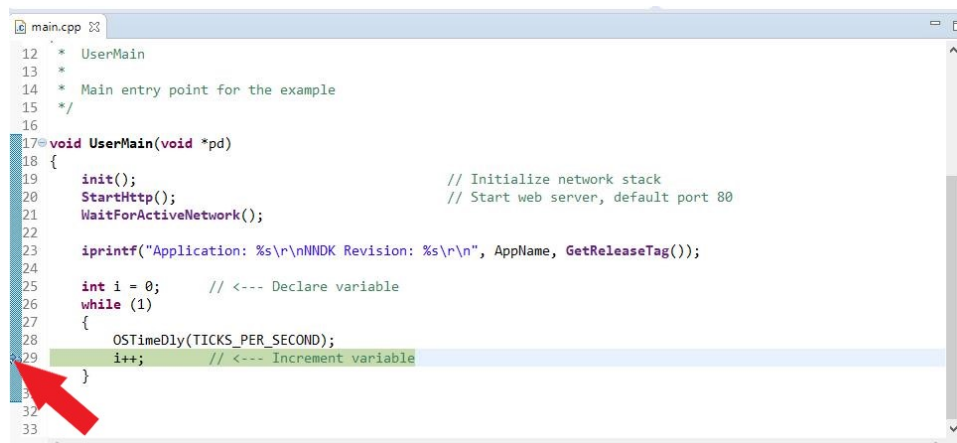


Figure 6.27 Set Breakpoint

The screen shot below shows the program stopped at line 29. The Variables window in the upper right shows the variable 'i', which has incremented 7 times so far.

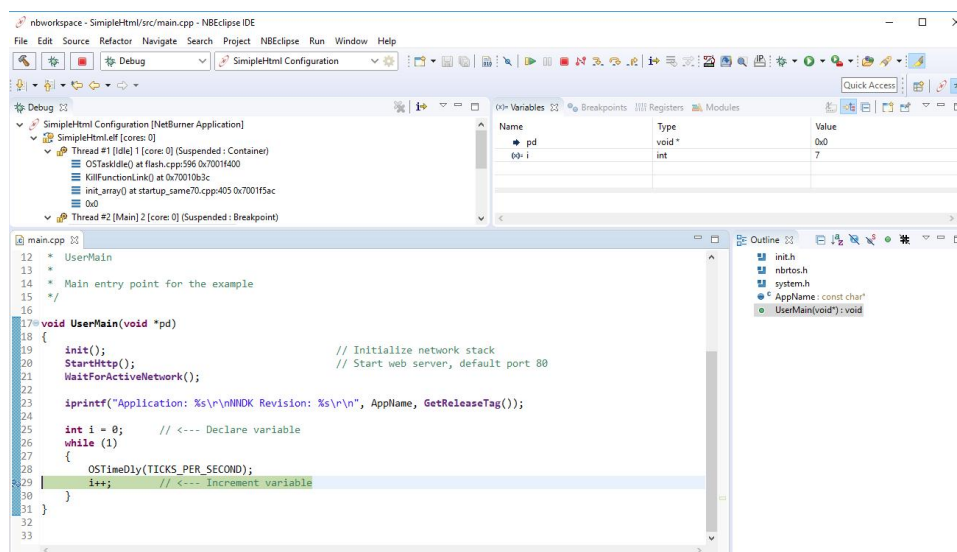


Figure 6.28 Stopped at Breakpoint

The debugger operations are controlled with the tool bar at the top of the screen. If you hover the cursor over the icon the tool tip will describe what it does. Common features are:

- Resume: Available when stopped at a breakpoint. Selecting it will resume the application and stop at the next breakpoint.
- Terminate: Disconnects the debugger from the target device. The device will resume execution without any breakpoints.

- Disconnect: Identical to Terminate.
 - Step Into: Single step into a function call.
 - Step Over: Single step to the next line of code in the current function.
 - Step Return: Execute until current function returns.
- Notes on debugging:
- You can switch between the Debug perspective and NetBurner perspective using the NetBurner and Debug icons in the upper right corner of the screen.
 - You can edit and build your application from either perspective.
 - Hovering the cursor over variables, macros and functions in the source code window will display information on that item.
 - The upper left window displays the RTOS task stack.
 - It is recommended to terminate a debug session before starting a new one.
 - The three icons at the top left of the screen: build, debug and stop, are not used by either perspective.

6.1.6 Rebuilding Projects & Libraries

6.1.6.1 Rebuild Project Source Files

A project "clean" will rebuild all of the project's source code files. It does not affect the library system files associated with the project. From the Project Explorer pane, right-click on your project, then select Clean Project:

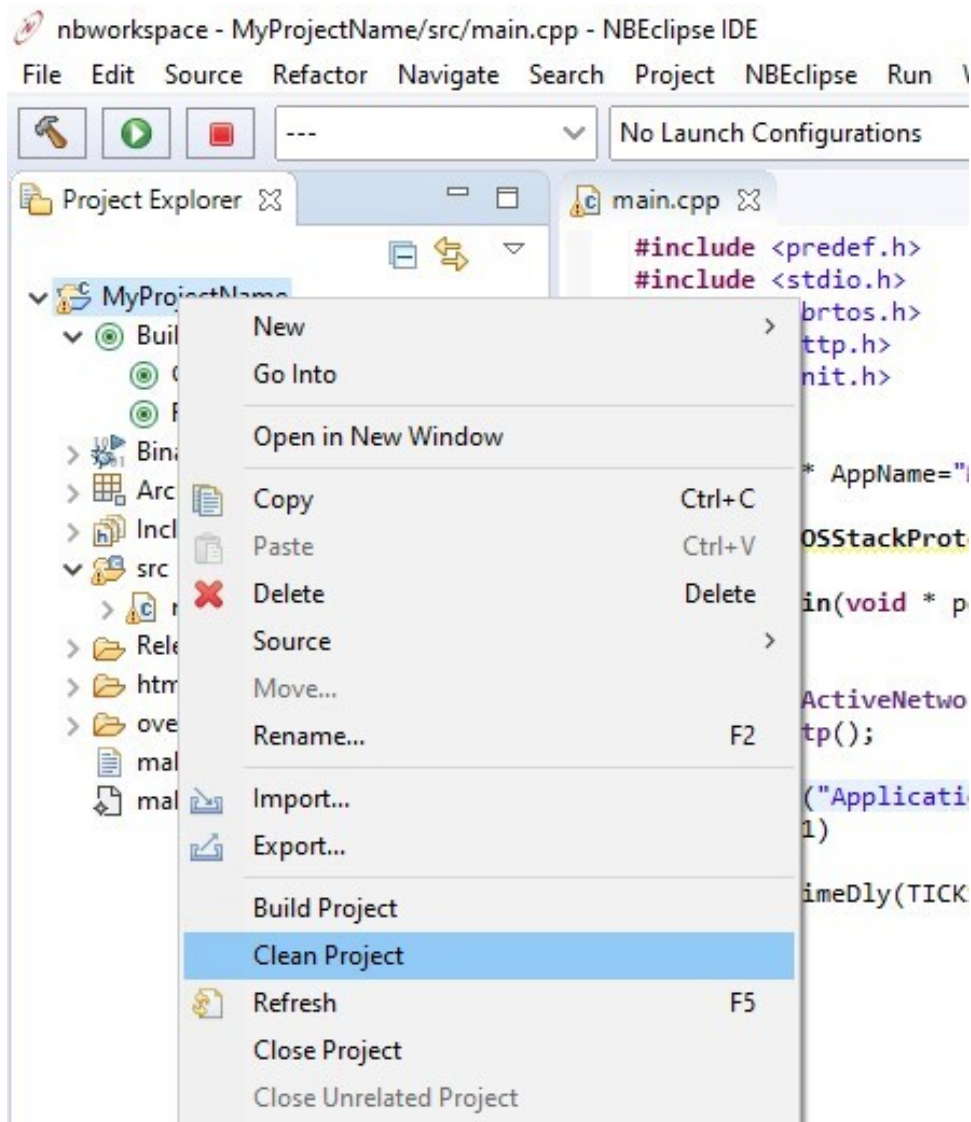


Figure 6.29 Rebuild project files

6.1.6.2 Rebuild Library Files

This is normally not necessary, but there are cases in which the user may want to rebuild the NetBurner system library files. For example, if the user made changes to system source files or added a system source file to a project's overload directory. In these cases, it's recommended to rebuild the NetBurner system library files followed by "cleaning" the project's source files.

To force a rebuild of the NetBurner library files associated with a project, double-click on Clean Project Library in the Build Targets tree:

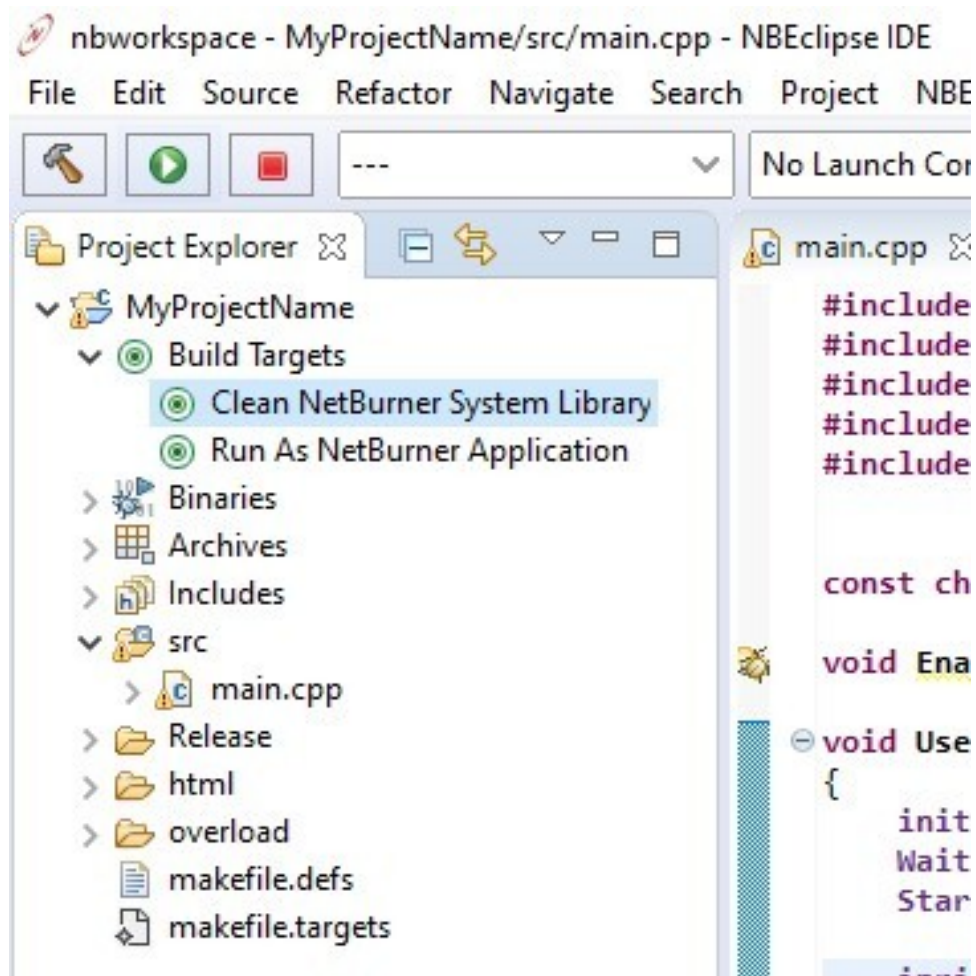


Figure 6.30 Rebuild a project library

6.1.7 Overload Directory

The overload directory, located inside a project, is used to override any system file by including local copies of system source or header files. This allows the user to isolate system files changes to a specific project. The overload directory provides an alternative to modifying the original system files, which would affect all projects.

One file that is frequently overridden to modify how NBRTOS works is `predef.h`, however we suggest using `predef-overload.h` instead for portability.

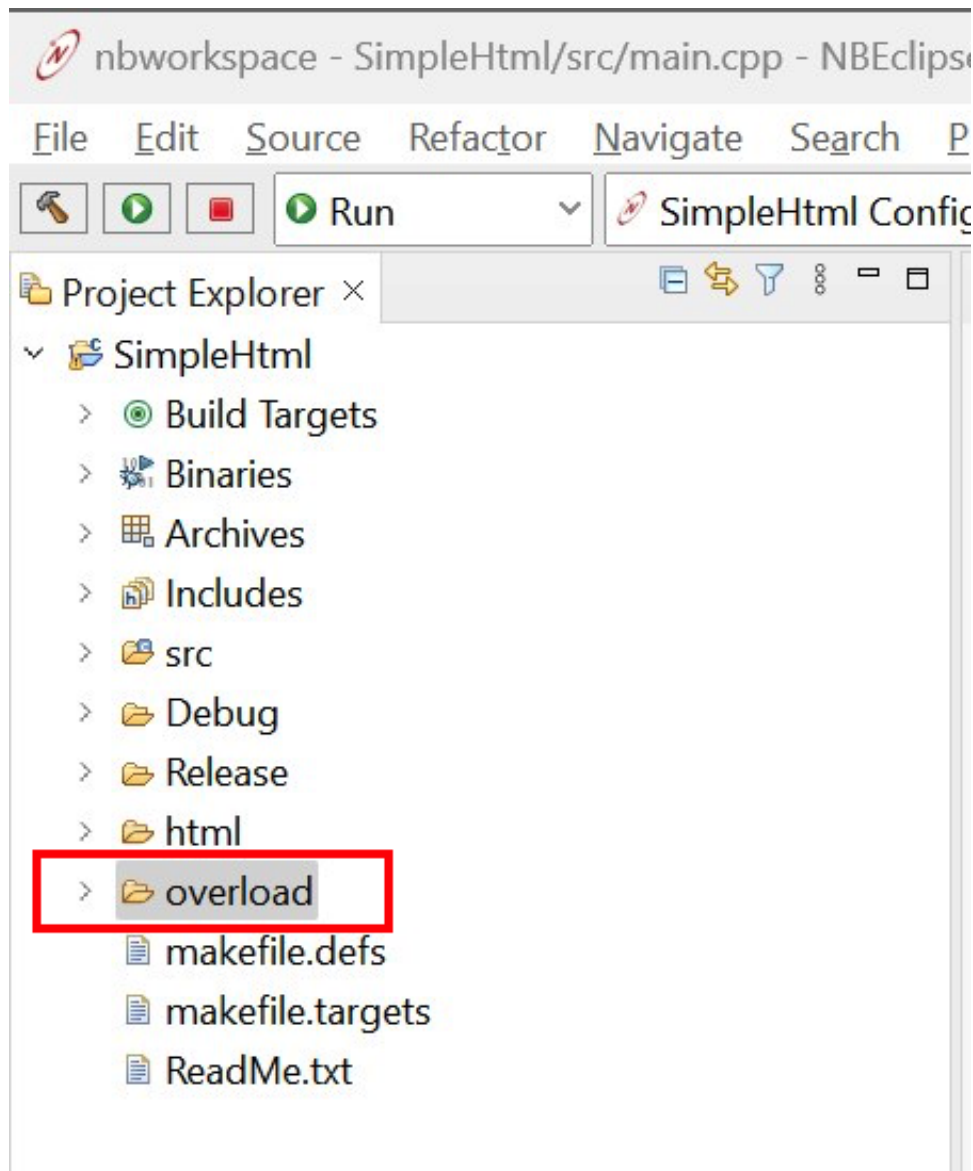


Figure 6.31 Project's Overload Directory

To use this feature, the file to be overloaded should be placed in the overload directory under a directory structure that matches the file's location relative to the NNDK install directory.

To demonstrate the usage and configuration of the overload directory, we will show an example. Let's say the user wants to overload the system's timezone structure so that they can change the name of a timezone. Perhaps the user is deploying their device in a non-English speaking country and would like to change the language used for timezone names. The timezone names are defined in a system file located in the following directory:

```
/nburn/nbrtos/source/timezones.cpp
```

Therefore, the directory structure is replicated inside the overload directory with the following structure:

```
<project root>/overload/nbrtos/source/timezones.cpp
```

The directory structure can be created by right-clicking the overload directory and selecting New->Folder. The overload directory must have a matching directory structure to the path of the system file to be overloaded. For this example, the overload directory should look like the following:

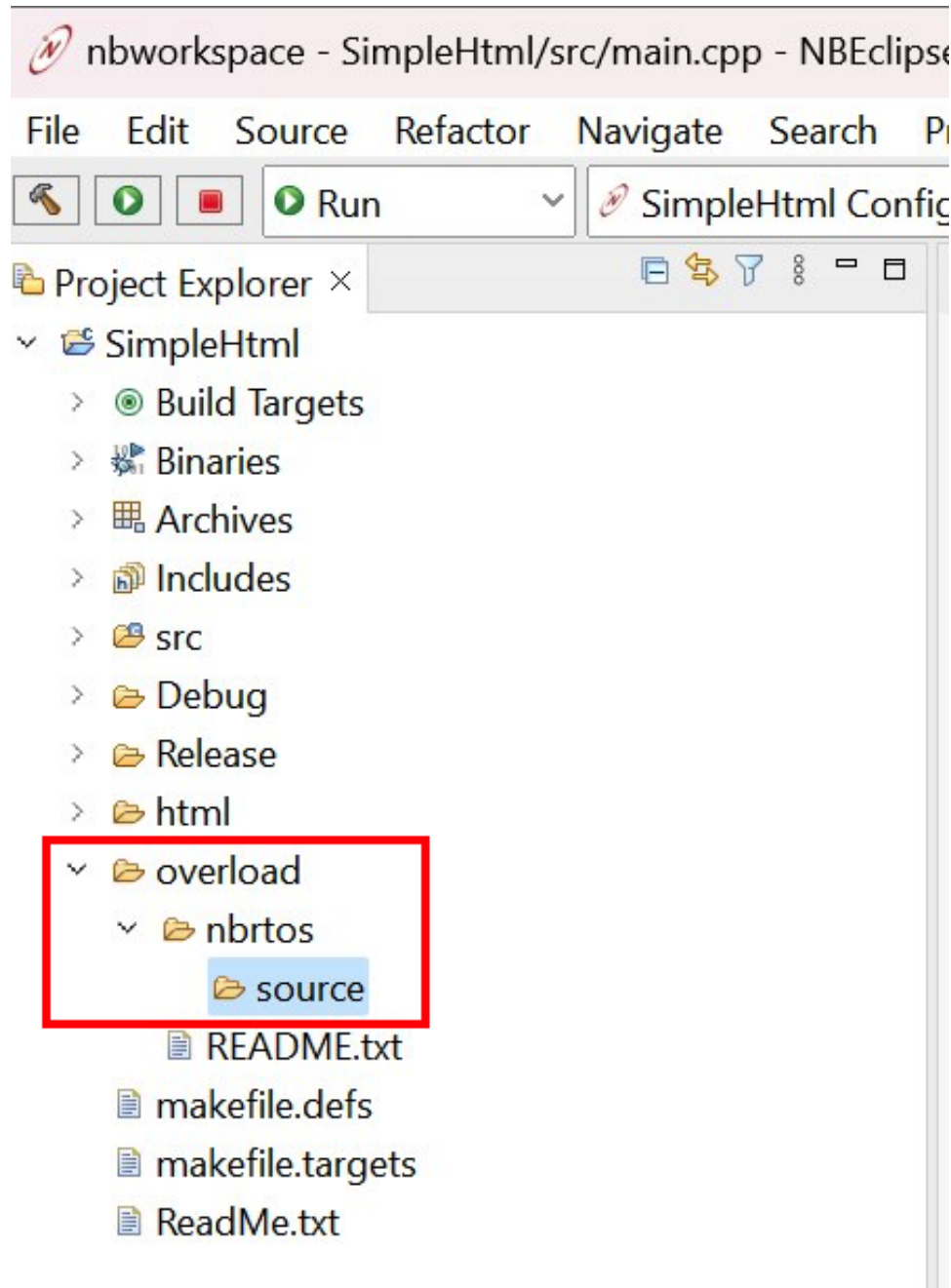


Figure 6.32 Project's Overload Directory Structure

Next, the modified source file must be placed in the correct path in the overload directory. The file can be drag and dropped in place. Alternatively, the file can be imported into the overload directory by right-clicking the overload directory and navigating to Import->General->File System and navigating to the source file. After adding timezone.cpp, our overload example would look like the following:

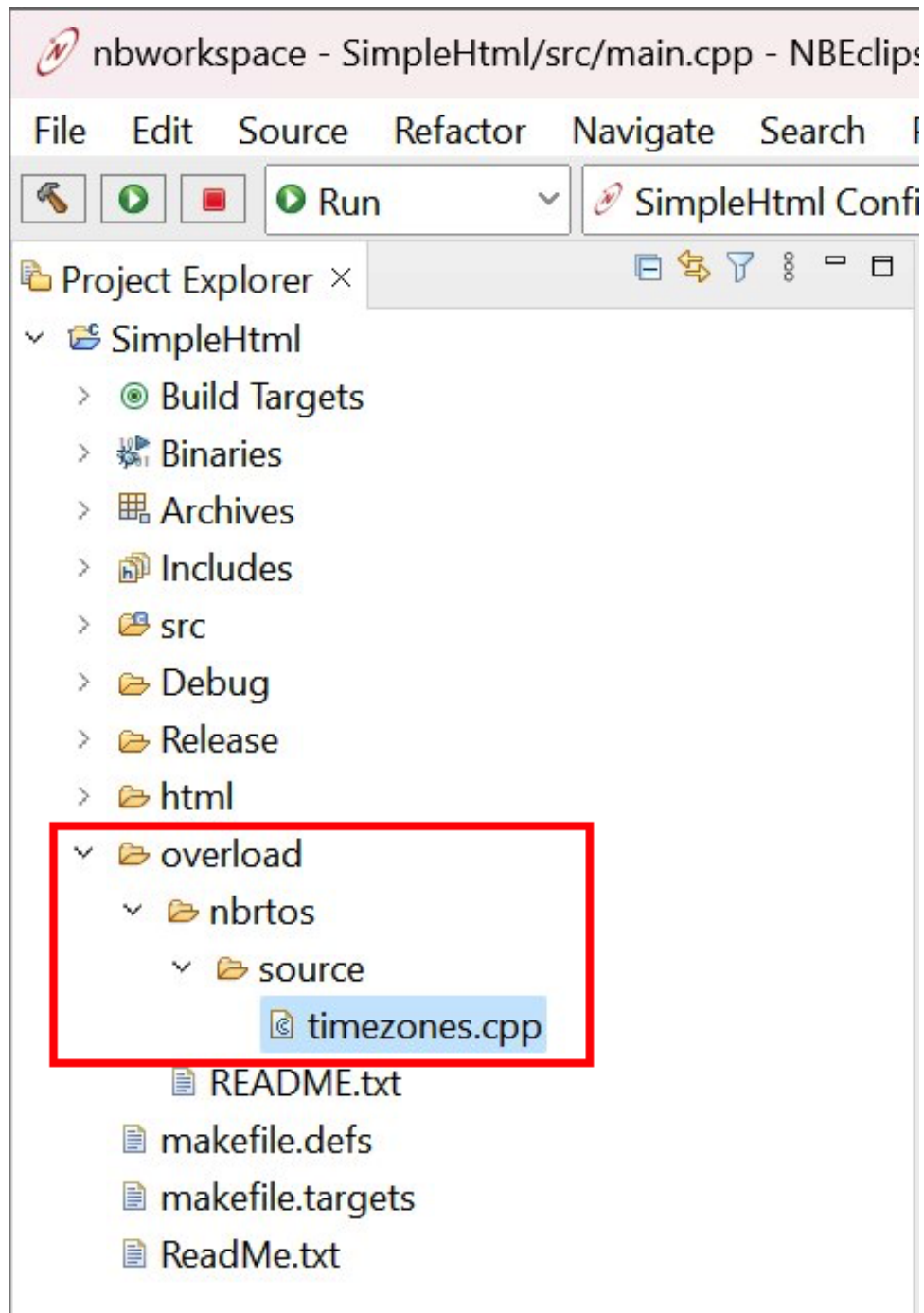


Figure 6.33 Overload Source File Added

Next, the project properties must be configured so that the C++ compiler knows to include the path to the overloaded source file. This can be done by right-clicking the project and selecting "Properties" as shown in the image below.

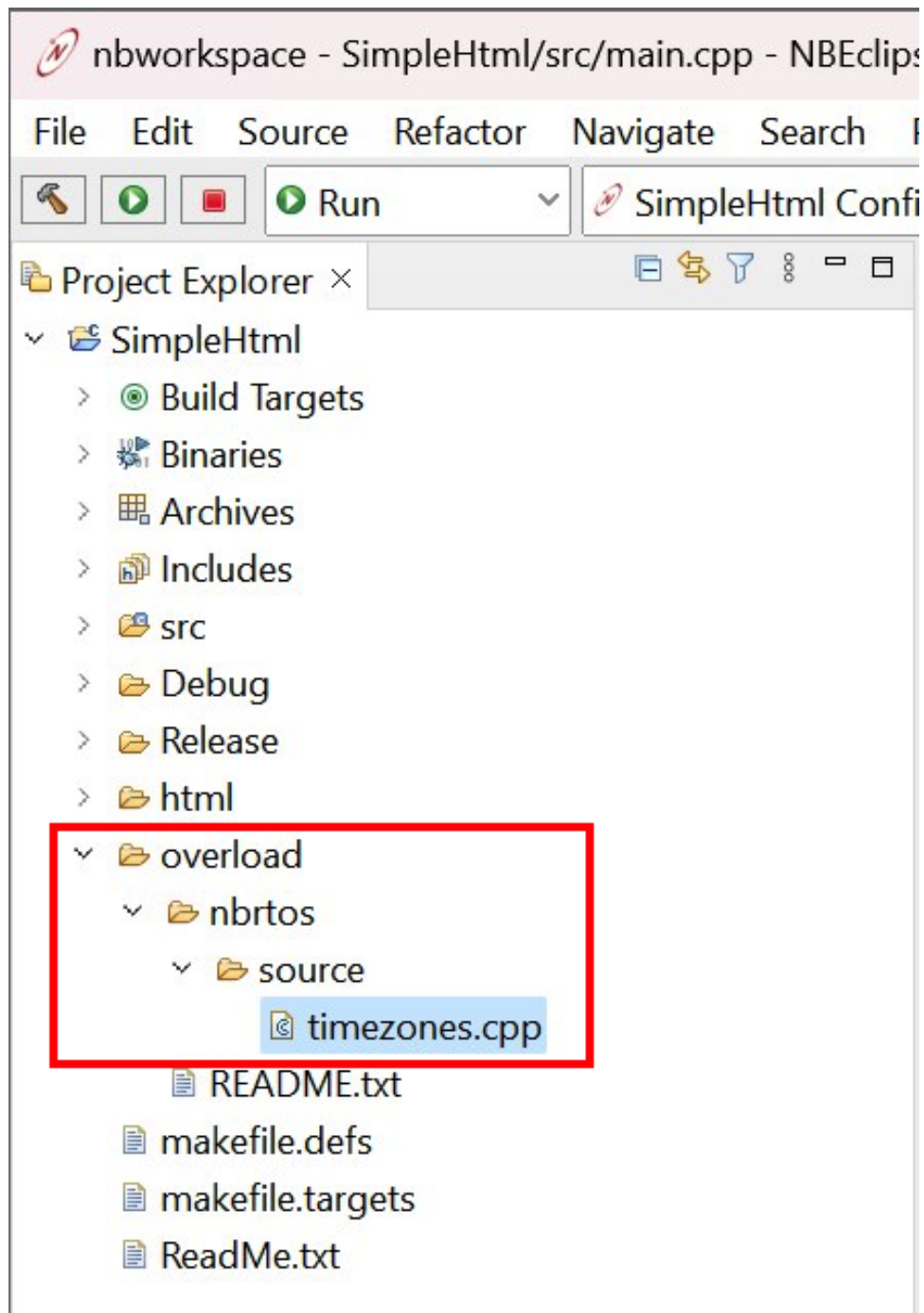


Figure 6.34 Project Properties

Navigate to C/C++ Build->Settings, select GNU C++ Compiler->Includes.

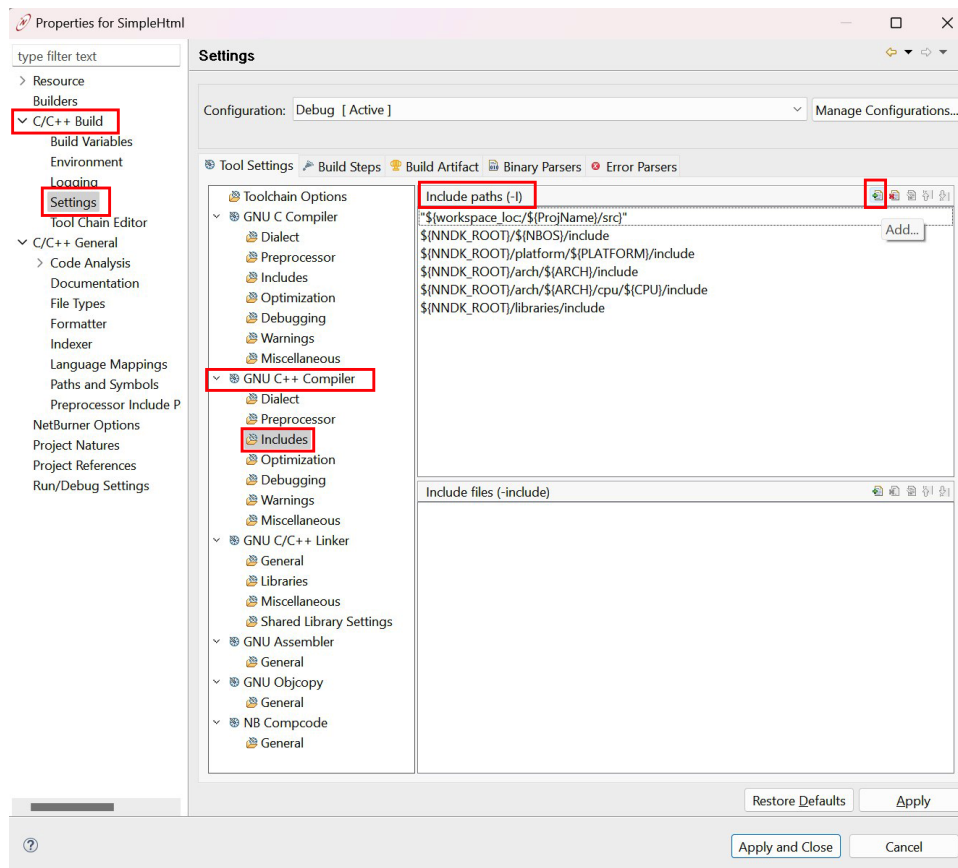


Figure 6.35 Overload Include Paths

Add the overload directory path to the "Include Paths" list by selecting the "+" logo. If utilizing C code, then the path to the overload directory should be added to GNU C Compiler->Includes as well.

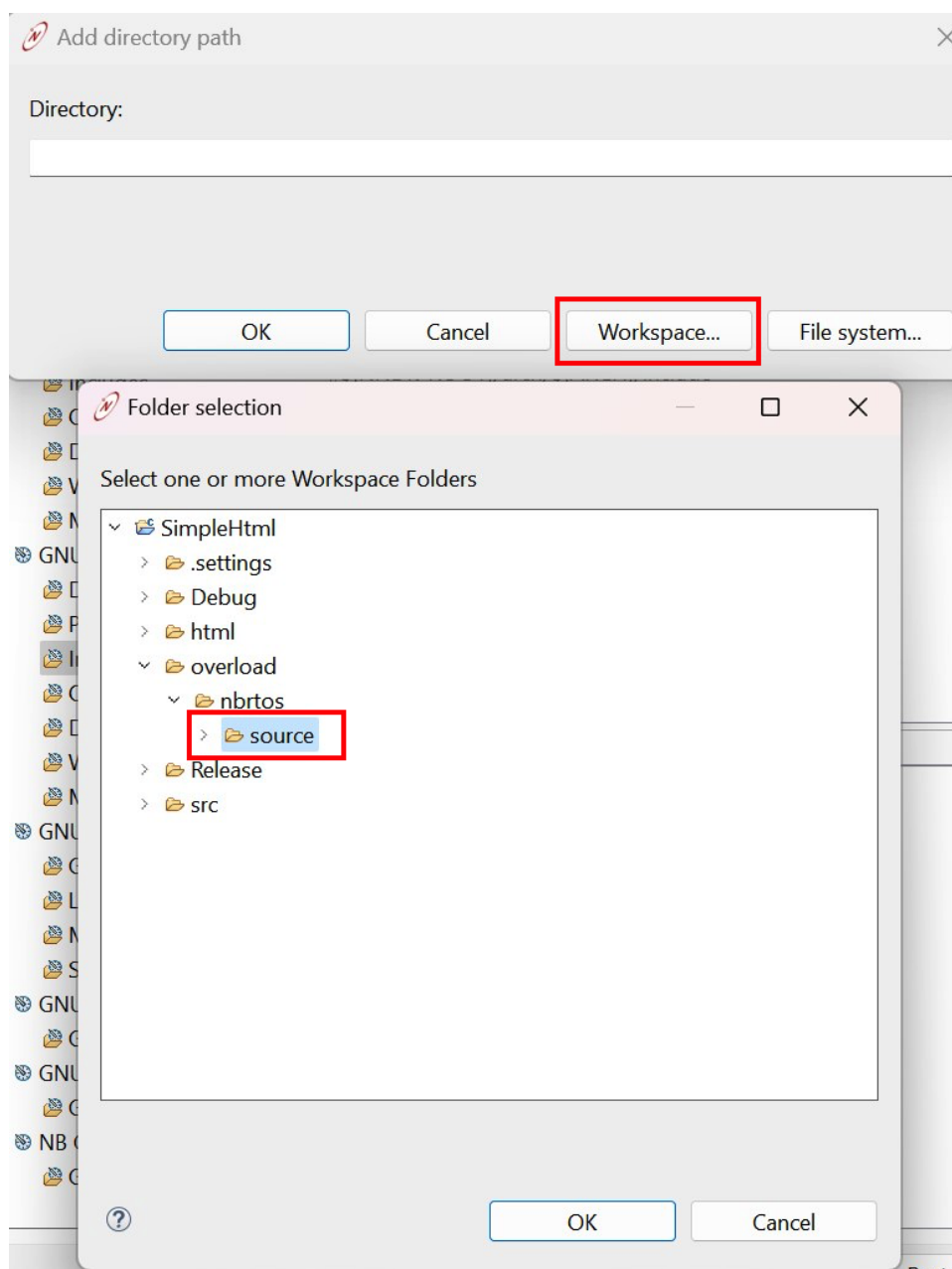


Figure 6.36 Overload Include Path Selection

After adding the overload directory path to the list of "Include Paths," the list should look like the following↵
:

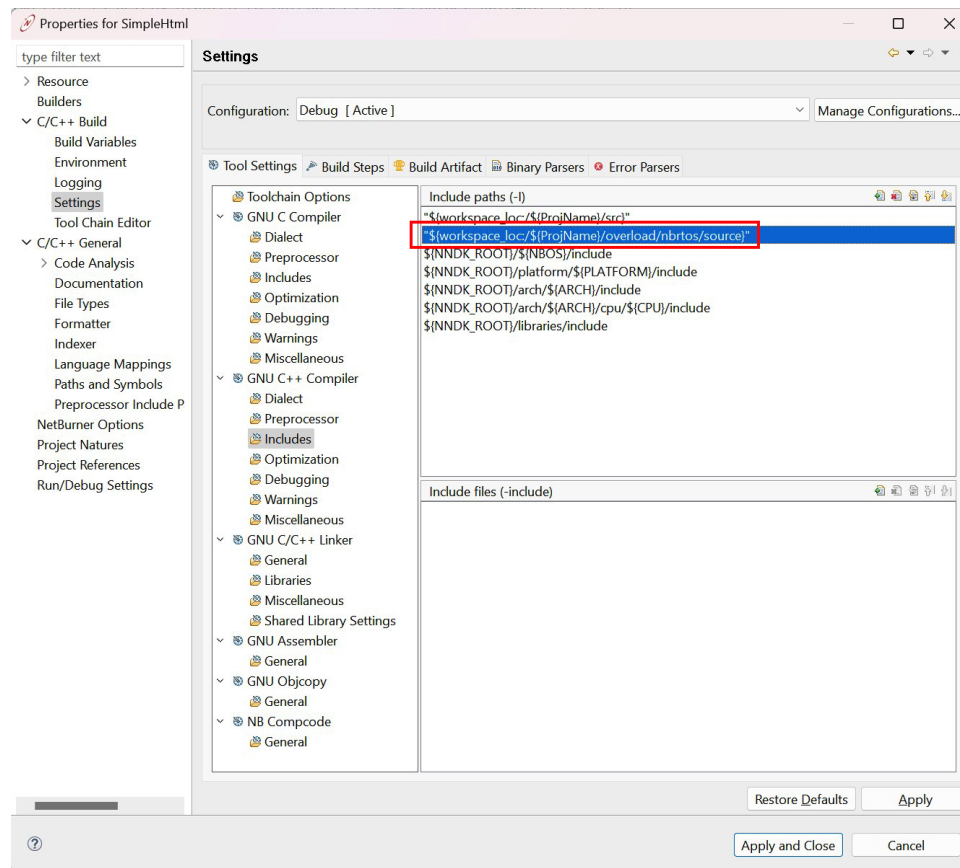


Figure 6.37 Overload Directory Path Added to Include Paths List

Confirm that the new overload directory path has been added to the "Include paths" list above the paths that begin with "\$NNDK_ROOT)/" as shown in the image above. If not, use the up-arrow button to move the overload directory path higher in the list.

Select "Apply and Close" to apply the modifications.

Last, we need to rebuild the NetBurner system library files then clean the project files (the order of operations matters). For instructions on doing so, see [Rebuilding Projects & Libraries](#). From this point on, modifying the overloaded source file, `timezone.cpp`, will automatically re-build the necessary files in the NetBurner system files when the project builds.

6.2 Add a Library to a Project

NBEclipse includes the most common libraries by default. However, there are times you will need to tell the build system to include additional libraries, such as if your application uses the Embedded Flash File System (EFFS) for flash memory cards, in which case you need to add the FatFile library.

Note: If you are building a project that uses EFFS functions, and have not added the FatFile library, you will see many link errors for functions that start with `f_` and `fm_`.

To add a library:

- In NBEclipse, right-click on your project, and select "Properties".

- Select "C/C++ Builds -> Settings" on the left-hand side.
- Select "GNU C/C++ Linker -> Libraries" under the "Tool Settings".
- In the "Libraries" list box, add "FatFile" by using the add icon in top-right corner of the list.
- In the "Libraries Search Path" list box, add the path to the library. For example, for MOD5441x device, the path is: `\nburn\platform\MOD5441X\original\lib`. Note: the naming convention for gcc is as follows: the name of the library is actually `libFatFile.a`, but in the Libraries entry box you only use what comes after "lib" and do not use the `.a` suffix. In this case, we just use "FatFile".

Note

The other common library to add is the EDFS-STD library: `StdFile`. EDFS-FAT is a FAT32 library for flash memory cards. EDFS-STD is used to create a file system in the flash memory chip on the device.

6.3 Change The IP Targeted By A Project

If the IP of your device changes (for example, if you unplug it and it is assigned a different IP when you plug it back in), you will need to tell the build system to target a different IP.

To change the targeted IP:

- In NBEclipse, click on "Project" from the top menu, then click "Properties".
- In the left-hand panel of the Properties menu, click "NetBurner Options".
- Change the targeted IP address in the right-hand side of the panel.

6.4 Change CompCode Settings for EDFS

The CompCode utility is used in the final stage of a build to compress an application image. The COMPCODE memory address range defines the amount of space available as application space so programming an application in flash memory does not overwrite any of the space allocated for the EDFS-STD file system. Documentation on the memory map for your particular device can be found in the Platform Reference section of this document.

To change the CompCode memory address range for the application in NBEclipse:

1. Right-click on your project, and select "Properties".
2. Select "C/C++ Builds > Settings" on the left-hand side.

3. Select "NB Compcode". In the All Options section the current value of the application memory range is displayed. In this example it is a NANO54415 platform. To make the next step easier, highlight and copy the memory address range.

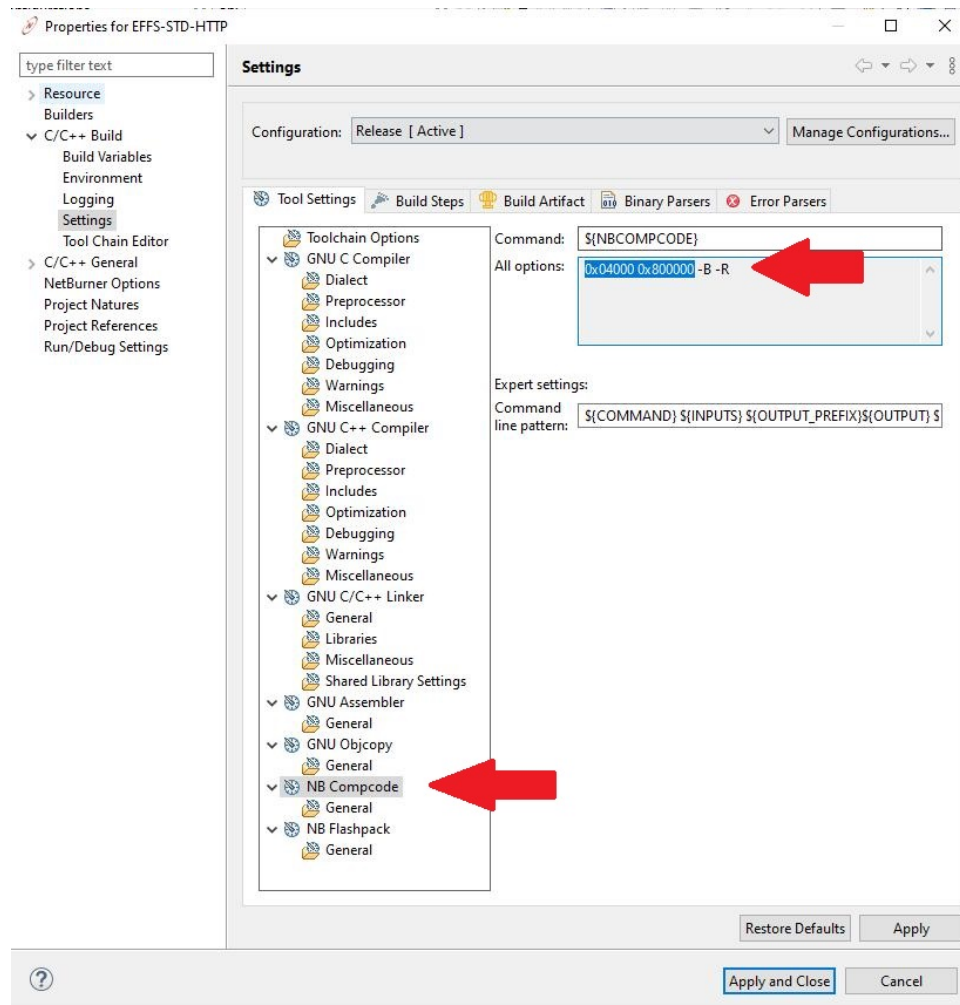


Figure 6.38 Compcode default values

4. Select the NB Compcode > General tab. If the current value is the factory default, you will see the macro `$(COMPCODEFLAGS)` displayed. This is the value that is read from the factory default file.

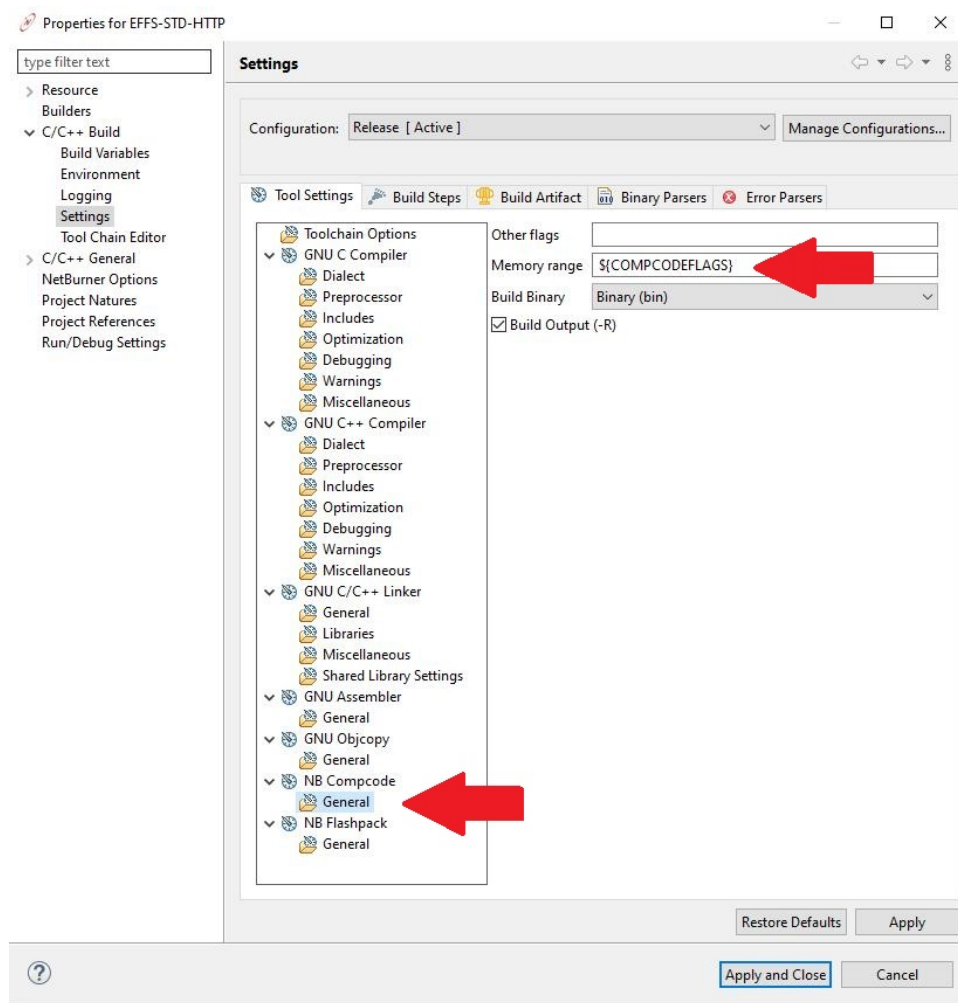


Figure 6.39 Compcode default values

5. Delete `${COMPCODEFLAGS}` and paste the memory range you copied in the previous step, then edit it to specify the desired memory range. In this example we will change `0x800000` to `0x700000`. So you end up with: `0x04000 0x700000`. This would be used to allocate `0x100000` bytes to the EFFS-STD file system space. When you are finished editing, click on Apply or Apply and Close.

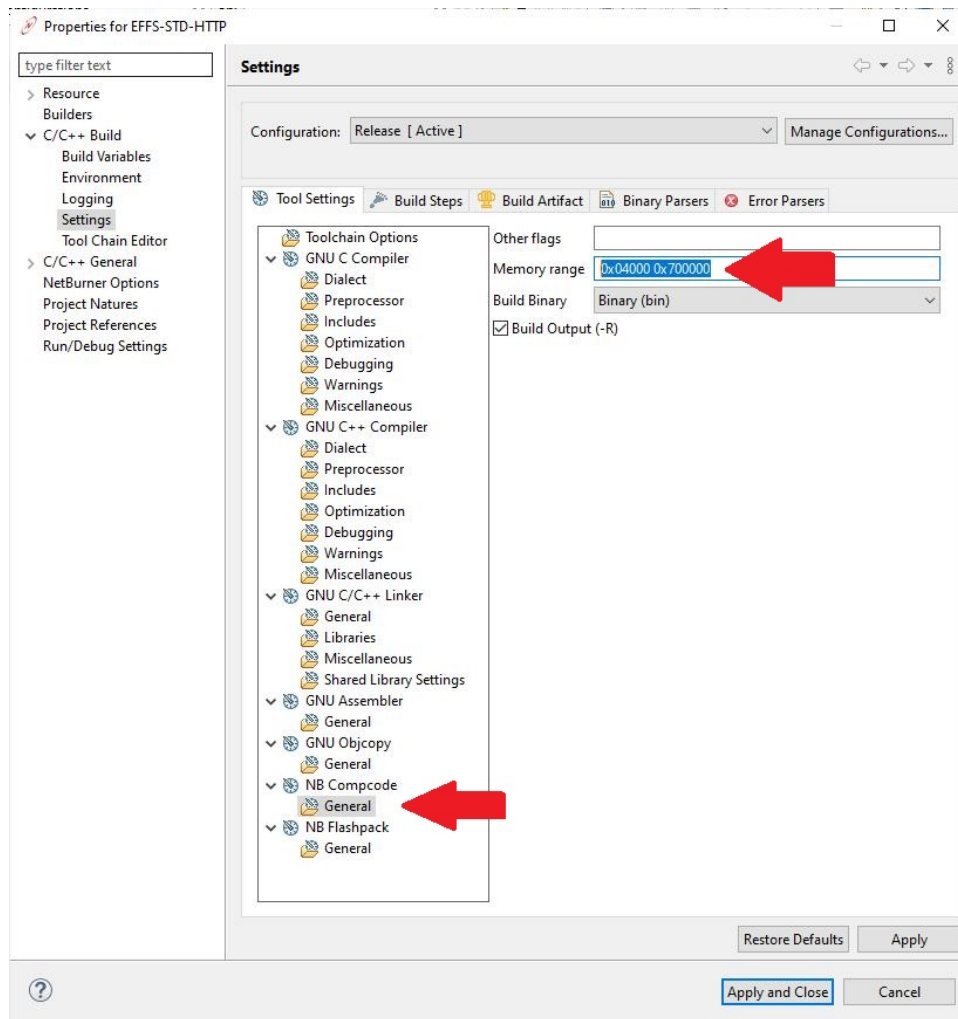


Figure 6.40 Compcode default values

Chapter 7

PC Tools and Utilities

This guide describes the PC tools and utilities that run as stand-alone programs, as well as automatically as part of the NetBurner development tools suite. The executable version of the tools are located in `\nburn\pcbin`, and the source code is located in `\nburn\pctools`.

7.1 Multi-threaded TTY Serial Terminal (MTTTY)

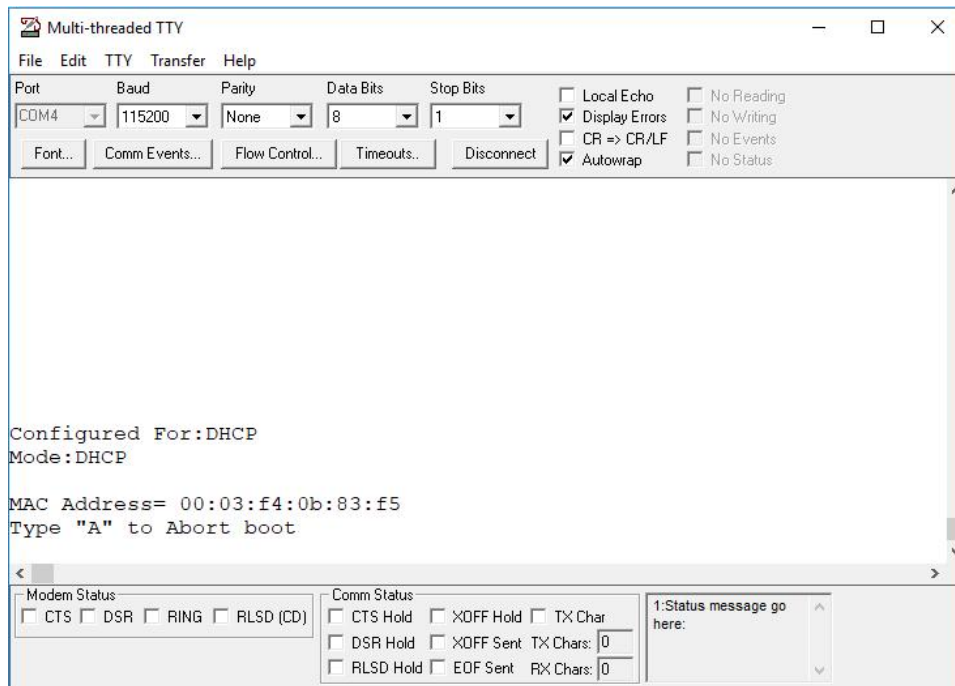
Multi-Threaded TTY (MTTTY) is a serial terminal utility that you can use to communicate with the RS-232 serial ports on your NetBurner device or development kit. It is multi-threaded in that you can run multiple instances of MTTTY for each serial port.

MTTTY is commonly used for:

- Displaying status messages sent from the NetBurner device, such as when using `iprintf()`
- As a serial interface to any application

MTTTY can be started from:

- Inside NBEclipse by clicking on the MTTTY icon
- From the Windows NetBurner program group
- From a command line. The path is: `\nburn\pcbin\mttty.exe`



7.1.1 MTTY FAQ

- The default serial port used for stdio for NetBurner devices is UART0.
- The DB9 connector pinout on NetBurner development boards will have the RS-232 rx and tx signals reversed, so that a null-modem adapter is not required and you can use a straight-through serial cable.
- The DB9 connector pinout on NetBurner Serial-To-Ethernet devices, such as the SB700EX and SB800EX, will be identical to a standard PC type pinout, so a null-modem adapter will be required. (The serial cable that comes with the NetBurner development kits for these specific devices is a null-modem cable)

7.2 Wget

Primary method to program an application into a NetBurner device by posting the application file to the device's config server. It is used automatically by both NBEclipse and the Makefile command line tools. The default flags to call wget are as follows: `wget -nv -O- --progress=dot:micro --post-file=<file name> <Device IP>:20034/appupdate.htm`

Wget also supports loading code onto a device that has been protected by username/password authentication. On the command line, the device username and password are configured by setting the environment flags NBUSER and NBPASSWORD. For example, in a make load, set the username and password as follows: `make load NBUSER=foo NBPASSWORD=bar`. In NBEclipse, username and password are set via the project options. Right click on a project and select Properties. Username and password text input boxes can be found under NetBurner Options. Finally, if manually utilizing wget, use the following wget flags to set a username and password: `--user=foo --password=bar`.

7.3 NBUpdate

Alternative method used to program an application into a NetBurner device by posting the application file to the device's config server. It is invoked as part of the make system for NetBurner development kit NBEclipse and command line builds. It can be replaced with WGET. The format is `nbupdate <file name> <device IP>`

7.4 Smart Traps

While smart traps is not a specific PC application, the interface is accomplished by running the MTTTY serial terminal. During software development it is possible for a software developer to download an application with a coding error severe enough to cause an application to crash. Common causes of a system crash are: task stack overflow, indexing an array out of bounds and bad pointer assignments. When a system crash occurs a device will usually recover by a reboot of the system. The purpose of the SmartTrap utility is to provide more information about the system when the crash occurred to assist with debugging the problem. Smart traps are automatically enabled by the `init()` function. Note that smart traps are disabled when building an application in debug mode.

- SmartTraps are commonly used for debugging an application crash
- SmartTraps are used in conjunction with the MTTTY serial terminal program. The SmartTraps information will be displayed on the debug serial port

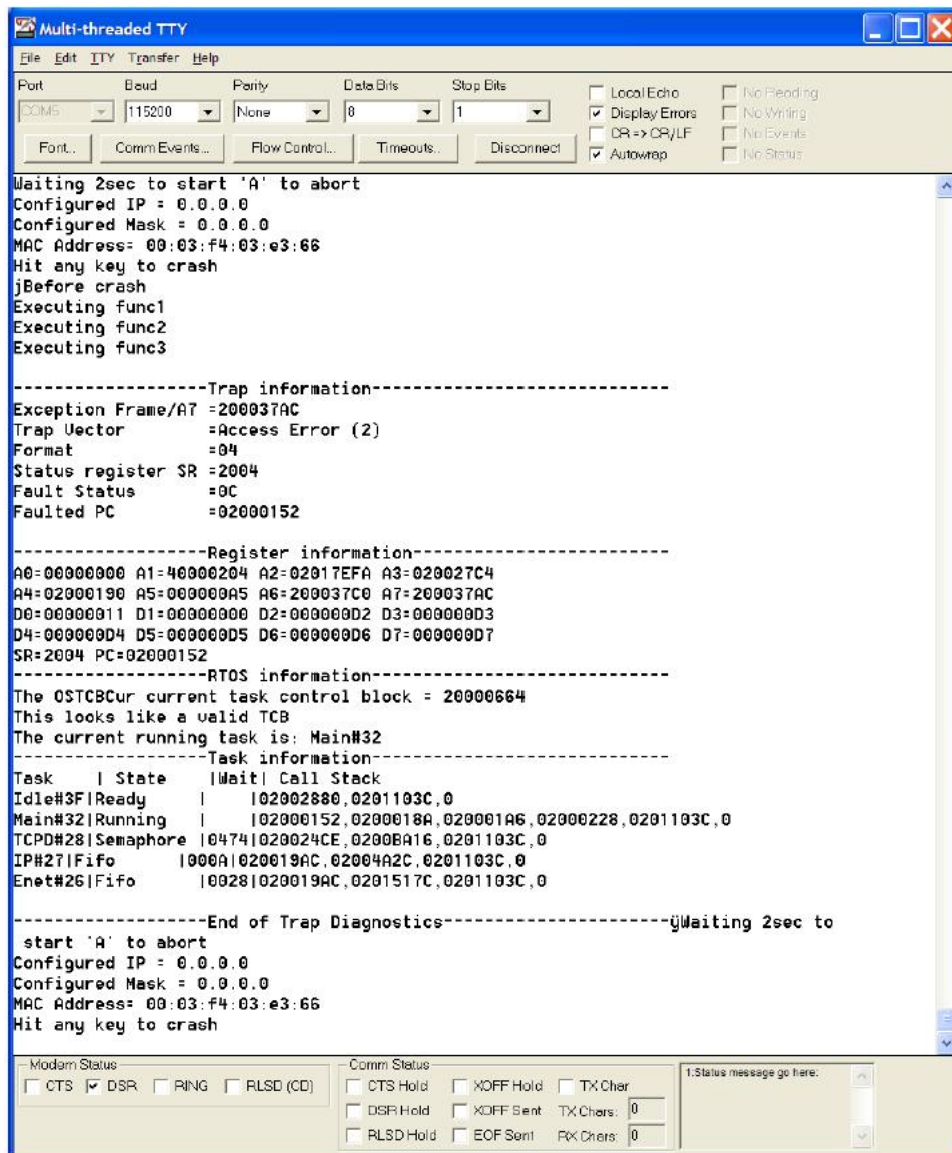


Figure 7.1 Example Smart Traps Output

The output has four sections. It is beyond the scope of this document to go into detail on each of the processor registers, please refer to the processor manual for more information, located in the `\nburn\docs` directory of your development tools installation.

7.4.1 Trap Information

This section displays the processor status and error registers. The most significant is the Faulted PC, with is the Program Counter register indicating the area where the fault occurred. In this example the program counter value at the time a trap was detected is 0x02000152.

7.4.2 Register Information

This section displays the processor's Address and Data registers.

7.4.3 RTOS Information

Identifies the RTOS Task Control Block and the current running task. The priority number is in hexadecimal. Traps caused by stack overflows may corrupt this section of the report.

7.4.4 Task Information

This is a very useful section of the report. It identifies each task, as well as its current state, wait time, and call stack. If you look at the Main task, you can see 6 entries with 0x02000152 (the faulted pc) at the left, and 0 at the right. The value of 0 is the start of the call stack. Each hexadecimal address between those two numbers represents each function that was called.

Each address can be related to the corresponding line number in the source code by using the WinAddr2Line utility. By selecting the .elf file location and each address in the call stack you can determine the calling sequence starting from 0 on the right and moving to the left. The last line signifies the end of the SmartTrap output. Since the boot monitor is configured to "boot to application", the device reboots after the trap occurs.

7.5 Task Scan

TaskScan is a network connected debugging tool that can be used to view the RTOS tasks and status of your running NetBurner application. This tool is unique in that it is active in the release build of your code, rather than the debug build which is compiled without optimization. It is very useful to determine such things as why a specific task is not running the way you had anticipated, if it is waiting on other tasks, and its call stack. TaskScan is only active when the PC program generates a request, so it will not impact your release code execution speed otherwise.

The `init()` function will automatically include `taskmon.h` and call `EnableTaskMonitor()`. TaskScan requires that you have the Executable and Linking Format (ELF) file for the specific application that is running on your NetBurner device. The .elf file is created by the compiler each time you build your application. It contains information TaskScan needs to link the application executable to the source code and task information. For example, if you have a project named MyProject, a file named MyProject.elf will be created.

TaskScan is commonly used for:

- Determining which tasks are running, their state and call stack.
- Looking for blocked tasks.

TaskScan can be started from:

- From the Windows NetBurner program group
- From a command line. The path is: `\nburn\pcbin\taskscan.exe`

7.5.1 TaskScan FAQ

- You must have a valid IP address and mask that can communicate with your PC that is running TaskScan
- TaskScan essentially takes a snapshot picture of what your application is doing at the time you press the Scan button. It does not otherwise impact the application
- You must have the exact .elf file used to create the application that is running on your NetBurner device to get valid information

After starting the Task Scan application use the Browse button to locate your project's elf file, and the Find button to locate the IP address.

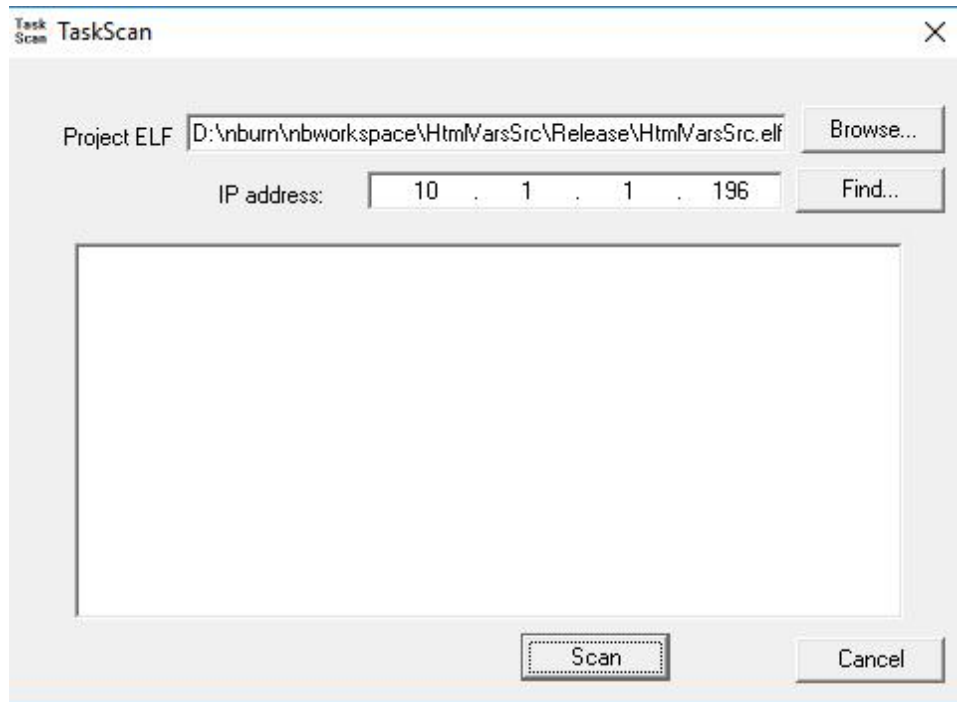


Figure 7.2 Starting Task Scan

Then select the Scan button:

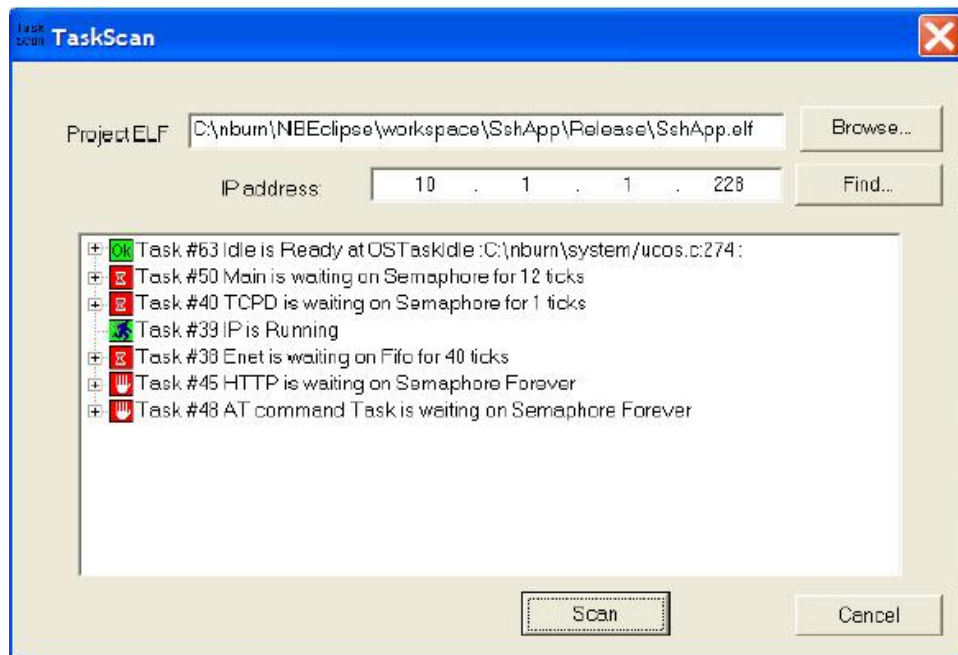


Figure 7.3 Scan Results

Icon	Description
	Green OK
	Green Running Man
	Red Hour Glass
	Red Hand

Figure 7.4 Task Scan Icons

7.6 WinAddr2Line

WinAddr2Line only works for ColdFire-based platforms. For ARM, see [Addr2line](#)

Utility to determine the source code location for a specified memory address and .elf file. This application is typically used in conjunction with the SmartTraps utility when debugging an application that is crashing.

WinAddr2Line is commonly used for:

- Determining the source code address using a faulted program counter memory address from SmartTraps.
- If the cause of a crash is severe memory corruption due to bad pointers, it is possible for the Faulted Program Counter to be pointing to an address in which not code exists. In this case WinAddr2Line will not be able to provide a source code reference because no source code exists at that location.

WinAddr2Line can be started from:

- NBEclipse main tool bar
- The Windows NetBurner program group.
- The command line. The path is: \nburn\pcbin\winaddr2line.exe.

7.6.1 Using WinAddr2Line

- Use the Browse button to select the .elf file for your application and type in the hexadecimal memory address, usually obtained by SmartTraps as the Faulted PC.
- Click on the Decode button.

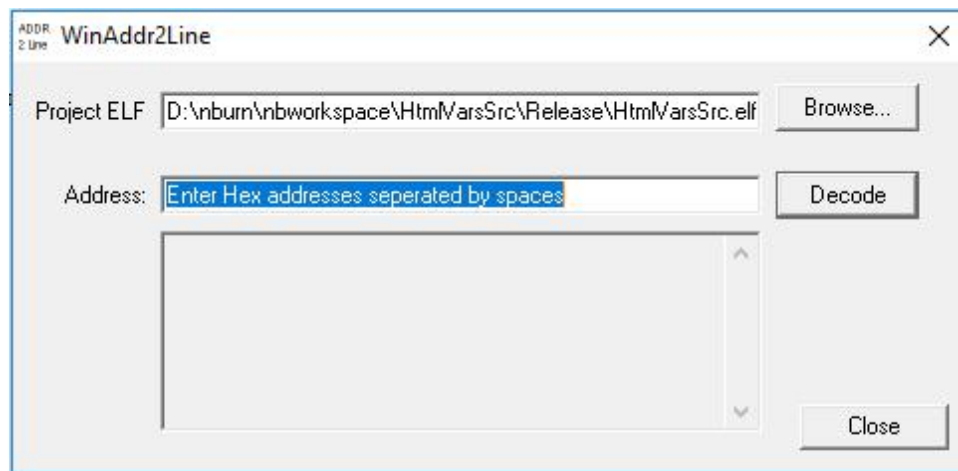


Figure 7.5 Select file and enter address in hexadecimal

7.7 IPSetup

The Windows IPSetup utility was used for network configuration in releases prior to 3.x. Network configuration is now down through the network configuration server on the NetBurner device. See [Device Discovery and Configuration](#) and [Config Server Programming Guide](#) for documentation on the 3.x network configuration system.

7.8 AutoUpdate

Autoupdate was used for loading applications to a device in releases prior to 3.x. Updating code is now done through the configuration server, nbupdate utility, or WGET.

7.9 Virtual Comm Port

The virtual comm port driver for Windows creates a virtual serial port and associates it with a network IP address and port number. It is typically used for legacy Windows applications without the ability to communicate over a network.

7.10 CompHtml

This utility is used by the NetBurner make system. It takes the contents of a project's html directory, implements the NetBurner dynamic web content scheme, callbacks, variables, access priorities, and mime types, and creates a file named `htmldata.cpp`.

7.11 CompCode

This utility is used by the NetBurner make system for 5441x platforms. It processes a `.s19` file into a compressed binary with checksum and header.

7.12 flashpack

This utility is used by the NetBurner make system for ARM platforms. It processes the `application.elf` file into a compressed binary with checksum and header.

Chapter 8

Platform References

This section provides the hardware reference information for your NetBurner platform, including memory maps, boot monitor and module recovery procedures in the event you download an application that is crashing the device.

- [MODM7AE70 Platform Reference](#)
- [MOD54415, MOD54417 Platform Reference](#)
- [NANO54415 Platform Reference](#)
- [SB800EX Platform Reference](#)
- [SBE70LC Platform Reference](#)
- [SOMRT1061 Platform Reference](#)
- [Recovery: MODM7AE70, SBE70LC](#)
- [Recovery: MOD5441x, NANO54415, SB800EX](#)
- [Recovery: SOMRT1061](#)
- [ARM Cache Management](#)

8.1 MODM7AE70 Platform Reference

8.1.1 Introduction

This document provides the memory map and locations of reference materials for those who wish to add additional hardware to their NetBurner device.

8.1.2 Processor Information

The MODM7AE70 uses the Microchip ARM SAME70 microprocessor. The reference manual and datasheet provide in-depth information on the processor, including register settings, bus configuration and timing information. It is located in the `<nburn_install>\docs\Arm` directory of your NetBurner installation.

8.1.2.1 Definitions

Definitions for the SAME70 can be found at [arch/cortex-m7/cpu/SAME70/include/same70q21.h](#)

8.1.3 Datasheet

The datasheet for the MODM7AE70 module includes information on connectors, signal names, and operational parameters. It is located on the MODM7AE70 product page under Documents and Downloads: [MODM7AE70 Datasheet](#).

8.1.4 Development Board Schematic

The MOD-DEV-70CR development board schematic is located in the `<nburn_install>\docs\NetBurner\platform\Schematics` directory. This schematic can be used for design ideas in your own hardware implementation for power, RS-232, RS-485, and SD Flash card implementation.

8.1.5 Memory Operation

The MODM7AE70 uses 3 types of memory:

- 2MB Flash memory for non-volatile system configuration storage, user application storage, the application in a compressed format, and the optional EFFS-STD file system (note this is different from the EFFS-FAT flash card file system).
- 8MB SDRAM containing the uncompressed application as well as any application data.
- 384KB on-chip SRAM located in the SAME70 microprocessor.

At power-up the application is decompressed from Flash memory to SDRAM and execution begins. If the application is continuously trapping the device can be recovered using the jumper recovery method as described in the Recover section, or if you have serial communication enabled the boot sequence can be aborted by sending the appropriate serial command when prompted after a power cycle.

8.1.6 Memory Map

Region	Size	Address Range	Description
Boot Loader	6k	0x00400000 to 0x004017FF	Initial boot and application extraction
Config Record	10k	0x00401800 to 0x00403FFF	System and User Configuration Data Flash
Cert. Storage	8k	0x00404000 to 0x00405FFF	Optional Security certificate storage
User Flash	8k	0x00406000 to 0x00407FFF	Unstructured binary User Flash Storage
App Flash	1.97M	0x00408000 to 0x005FFFFF	Compressed Application and opt file system
Vector Table	-	0x20400000 to 0x2040013F	SRAM - Interrupt Vector Table
Fast App RAM	384k	0x20400180 to 0x2045FFFF	SRAM - Fast Application RAM
Backup RAM	1k	0x40074000 to 0x400743FF	Low power backup RAM
External Bus	-	0x60000000 to 0x60FFFFFF	Chip Select 0
External Bus	16M	0x61000000 to 0x61FFFFFF	Chip Select 1
External Bus	16M	0x62000000 to 0x62FFFFFF	Chip Select 2
External Bus	16M	0x63000000 to 0x63FFFFFF	Chip Select 3
App RAM	8M	0x70000000 to 0x707FFFFFFF	SDRAM - Standard Application RAM where code is decompressed and variables are allocated
QSPI XIP	-	0x80000000 to 0x80FFFFFF	QSPI (User add-on) - For utilizing memory Mapped Quad SPI memories

8.1.7 External Bus Interface

The MODM7AE70 provides 17 address lines (A0-A16) and a 16-bit data bus line (D0-D15). The data and address lines shared with the SDRAM are buffered to the module's P1 header pins. Please refer to the MODM7AE70 schematic and datasheet for connector and signal locations. The Microchip SAME70 datasheet, section 33 on External Bus Interface (EBI) and section 34 on Static Memory Controller (SMC), provide details on peripheral functionality and configuration.

8.2 MOD54415, MOD54417 Platform Reference

8.2.1 Introduction

This document provides the memory map and locations of reference materials for those who wish to add additional hardware to their NetBurner device. The MOD54415 and MOD54417 are very similar. The difference is the MOD54417 provides two hardware network interfaces. The designation MOD5441x in this document applies to both modules.

8.2.2 Processor Information

The MOD54415 uses the NXP MCF54415 microprocessor, and the MOD54417 uses the MCF54417 microprocessor. The NXP MCF5441x reference manual and datasheet provide in-depth information on both processors and is located in the `<nburn_install>\docs\NXP` directory of your NetBurner Network Development Kit installation.

8.2.2.1 Definitions

Definitions for the MCF5441X can be found at [arch/coldfire/cpu/MCF5441X/include/sim5441x.h](http://arch.coldfire/cpu/MCF5441X/include/sim5441x.h)

8.2.3 Datasheet

The datasheet for the MOD5441x module can be found on the MOD54415/MOD54417 product page under Documents and Downloads: [MOD5441x Datasheet](#). The datasheet is the primary control document for connector pin-outs and signal functions.

8.2.4 Development Board Schematic

The MOD-DEV-70CR development board schematic is located in the `<nburn_install>\docs\NetBurner\platform\Schematics` directory. This schematic can be used for design ideas in your own hardware implementation for power, RS-232, RS-485, and SD Flash card implementation.

8.2.5 Memory Operation

The MOD5441x uses 4 types of memory:

- A 4MB write protected serial SPI Flash chip containing the Boot Monitor and Alternate Boot monitor. This provides a recovery method should events such as a bad application that causes continuous traps, or application Flash memory corruption.
- A 32MB parallel Flash chip containing non-volatile system configuration storage, user application storage, the application in a compressed format, and the optional EDFS-STD file system (note this is different from the EDFS-FAT flash card file system).
- A 64MB DDR2 Memory chip that contains the uncompressed application as well as any application data.
- 64KB on-chip SRAM located on the MCF5441x microprocessor.

Note

The reference to "2.x" is there for customers migrating from the NetBurner 2.x development tools. 2.x System Configuration is not used when running 3.x applications.

4MB SPI Boot Flash	32MB Parallel Flash	64MB DDR2 RAM
Boot Monitor	2.x System Configuration	Application (decompressed)
Alternate Boot Monitor	User Parameters	System Stack
	Application (compressed)	

3.x System Configuration

The Boot Monitor executes at power-up and attempts to decompress the application into DDR2 RAM and begin execution. If the application is corrupted or does not exist, it will attempt to run the Alternate Boot Monitor. If the Alternate Boot Monitor cannot be run, the system will remain in the Boot Monitor. The Alternate Boot Monitor provides network communication to enable an application download, and is the normal recovery method should unrecoverable application errors occur. The Boot Monitor only provides serial communication.

8.2.6 Memory Map

The MOD5441x provides 16 address lines (A0-A15) and a 16-bit data bus (D16-D31) running in mode 2. An address bus latch is included on the MOD5441x so external latch is not required for this mode. If you are adding peripherals to your NetBurner device's address/data bus, you can choose unused memory locations from the table below. Once a range has been selected, you will need to configure the appropriate chip select address and option registers in the MCF5451x processor. Please refer to the chip select sections of the NXP MCF5441x processor manual for details on the register configuration. Unlike many ColdFire processors, you are not free to use any unused address for chip selects. The range is restricted. In this table we have used "undefined" for restricted ranges and "unused" for ranges that are valid for additional chip selects. Please see the MCF5441x Manual Section 20.3.1 and the Note above Figure 20-1 for additional details.

Memory Region	Address Range	Description
Undefined	0x00000000 to 0x01FFFFFF	Undefined area to catch null pointers
Unused	0x02000000 to 0x3FFFFFFF	Available to applications
DDR2 RAM	0x40000000 to 0x43FFFFFF	64MB of DDR2 RAM
VBR	0x40000000 to 0x400003FF	1kB processor vector base register
RAMBAR	0x80000000 to 0x8000FFFF	64kB Processor internal SRAM
Parallel Flash	0xC0000000 to 0xC1FFFFFF	32MB parallel flash memory
System Config	0xC0000000 to 0xC001FFFF	128kB system configuration flash sector
2.x Config Rec	0xC0000000 to 0xC0000400	1kB 2.x configuration storage
3.x Certificate	0xC0000400 to 0xC0002400	8kB 3.x certificate storage
3.x System Config.	0xC0002400 to 0xC001FFFF	119kB 3.x system configuration
User Params	0xC0020000 to 0xC003FFFF	128kB user parameter storage
Application Code	0xC0040000 to 0xC1FFFFFF	Compressed application code
Unused	0xC2000000 to 0xDFFFFFFF	Available to applications
IPSBAR	0xE0000000 to 0xFFFFFFFF	Processor internal device registers - accessible using the SIM structure defined in sim5441x.h

8.3 NANO54415 Platform Reference

8.3.1 Introduction

This document provides the memory map and locations of reference materials for those who wish to add additional hardware to their NetBurner device.

8.3.2 Processor Information

The NANO54415 uses the NXP MCF54415 microprocessor. The reference manual and datasheet provide in-depth information on the microprocessor, including register settings, bus configuration and timing information. It is located

in the <nburn_install>\docs\NXP directory of your NetBurner installation.

8.3.2.1 Definitions

Definitions for the MCF5441X can be found at arch/coldfire/cpu/MCF5441X/include/sim5441x.h

8.3.3 Datasheet

The datasheet for the NANO54415 module includes information on connectors, signal names, and operational parameters. It is located on the NANO54415 product page under Documents and Downloads:

[NANO54415 Datasheet](#).

8.3.4 Development Board Schematic

The NANO54415 development board schematic is located in the <nburn_install>\docs\NetBurner\Platform\Schematics directory. This schematic can be used for design ideas in your own hardware implementation for power, RS-232, RS-485, and SD Flash card implementation.

8.3.5 Memory Operation

The NANO54415 uses 4 types of memory:

- A 4MB write protected serial SPI Flash chip containing the Boot Monitor and Alternate Boot monitor. This provides a recovery method should events such as a bad application that causes continuous traps, or application Flash memory corruption.
- A 8MB serial SPI Flash chip containing non-volatile system configuration storage, user application storage, the application in a compressed format, and the optional EFFS-STD file system (note this is different from the EFFS-FAT flash card file system).
- A 64MB DDR2 memory chip that contains the uncompressed application as well as any application data.
- 64KB on-chip SRAM located on the MCF54415 microprocessor.

The Boot Monitor executes at power-up and attempts to decompress the application into DDR2 RAM and begin execution. If the application is corrupted or does not exist, it will attempt to run the Alternate Boot Monitor. If the Alternate Boot Monitor cannot be run, the system will remain in the Boot Monitor. The Alternate Boot Monitor provides network communication to enable an application download, and is the normal recovery method should unrecoverable application errors occur. The Boot Monitor only provides serial communication.

Note

The reference to "2.x" is there for customers migrating from the NetBurner 2.x development tools. 2.x System Configuration is not used when running 3.x applications.

4MB SPI Boot Flash	8MB SPI Flash	64MB DDR2 RAM
Boot Monitor	2.x System Configuration	Application (decompressed)
Alternate Boot Monitor	User Parameters	System Stack
	Application (compressed)	
	3.x System Configuration	

8.3.6 Processor Runtime Memory Map

The runtime memory map for the location of the microprocessors onboard SRAM and DDR2 RAM is shown below:

Memory Region	Address Range	Description
Undefined	0x00000000 to 0x01FFFFFF	Undefined area to detect null pointers
DDR2 RAM	0x40000000 to 0x43FFFFFF	64MB of DDR2 RAM
VBR	0x40000000 to 0x400003FF	1kB Processor Vector Base Register
RAMBAR	0x80000000 to 0x8000FFFF	64kB Processor internal SRAM
IPSBAR	0xE0000000 to 0xFFFFFFFF	Processor internal device registers

8.3.7 Application SPI Flash Memory

There is a total of 8MB SPI Flash memory used to store the compressed application and non-volatile parameters. Read/write operations are only accessible through system API functions. The memory does not appear in the microprocessor's address map.

SPI Address	Description
0x000000	8kB 2.x System Configuration Record storage
0x002000	8kB User Parameter Storage
0x004000	7.9MB Compressed Application Code
0x7EE000	8kB 3.x Reserved System Configuration Storage
0x7F0000	64kB 3.x System Configuration

8.4 SB800EX Platform Reference

8.4.1 Introduction

This document provides the memory map and locations of reference materials for those who wish to add additional hardware to their NetBurner device.

8.4.2 Processor Information

The SB800EX uses the NXP MCF54415 microprocessor. The reference manual and datasheet provide in-depth information on the microprocessor, including register settings, bus configuration and timing information. It is located in the <nburn_install>\docs\NXP directory of your NetBurner installation.

8.4.2.1 Definitions

Definitions for the MCF5441X can be found at arch/coldfire/cpu/MCF5441X/include/sim5441x.h

8.4.3 Datasheet

The datasheet for the SB800EX module includes information on connectors, signal names, and operational parameters. It is located on the SB800EX product page under Documents and Downloads: [SB800EX Datasheet](#).

8.4.4 Memory Operation

The SB800EX uses 4 types of memory:

- A 4MB write protected serial SPI Flash chip containing the Boot Monitor and Alternate Boot monitor. This provides a recovery method should events such as a bad application that causes continuous traps, or application Flash memory corruption.

- An 8MB serial SPI Flash chip containing non-volatile system configuration storage, user application storage, the application in a compressed format, and the optional EDFS-STD file system (note this is different from the EDFS-FAT flash card file system).
- A 64MB DDR2 memory chip that contains the uncompressed application as well as any application data.
- 64K on-chip SRAM located on the MCF54415 microprocessor.

The Boot Monitor executes at power-up and attempts to decompress the application into DDR2 RAM and begin execution. If the application is corrupted or does not exist, it will attempt to run the Alternate Boot Monitor. If the Alternate Boot Monitor cannot be run, the system will remain in the Boot Monitor. The Alternate Boot Monitor provides network communication to enable an application download, and is the normal recovery method should unrecoverable application errors occur. The Boot Monitor only provides serial communication.

Note

The reference to "2.x" is there for customers migrating from the NetBurner 2.x development tools. 2.x System Configuration is not used when running 3.x applications.

4MB SPI Boot Flash	8MB SPI Flash	64MB DDR2 RAM
Boot Monitor	2.x System Configuration	Application (decompressed)
Alternate Boot Monitor	User Parameters	System Stack
	Application (compressed)	
	3.x System Configuration	

8.4.5 Processor Runtime Memory Map

The runtime memory map for the location of the microprocessors onboard SRAM and DDR2 RAM is shown below:

Memory Region	Address Range	Description
Undefined	0x00000000 to 0x01FFFFFF	Undefined area to detect null pointers
DDR2 RAM	0x40000000 to 0x43FFFFFF	64MB of DDR2 RAM
VBR	0x40000000 to 0x400003FF	1kB Processor Vector Base Register
RAMBAR	0x80000000 to 0x8000FFFF	64kB Processor internal SRAM
IPSBAR	0xE0000000 to 0xFFFFFFFF	Processor internal device registers

8.4.6 Application SPI Flash Memory

There is a total of 8MB SPI Flash memory used to store the compressed application and non-volatile parameters. Read/write operations are only accessible through system API functions. The memory does not appear in the microprocessor's address map.

SPI Address	Description
0x000000	8kB 2.x System Configuration Record storage
0x002000	8kB User Parameter Storage
0x004000	7.9MB Compressed Application Code

0x7EE000	8kB 3.x Reserved System Configuration Storage
0x7F0000	64kB 3.x System Configuration

8.5 SBE70LC Platform Reference

8.5.1 Introduction

This document provides the memory map and locations of reference materials for those who wish to add additional hardware to their NetBurner device.

8.5.2 Processor Information

The SBE70LC uses the Microchip ARM SAME70 microprocessor. The reference manual and datasheet provide in-depth information on the processor, including register settings, bus configuration and timing information. It is located in the `<nburn_install>\docs\Arm` directory of your NetBurner installation.

8.5.2.1 Definitions

Definitions for the SAME70 can be found at arch/cortex-m7/cpu/SAME70/include/same70q21.h

8.5.3 Datasheet

The datasheet for the SBE70LC module includes information on connectors, signal names, and operational parameters. It is located on the SBE70LC product page under Documents and Downloads: [SBE70LC Datasheet](#).

8.5.4 Development Board Schematic

The SB70LC-ADPT-100 development board schematic is located in the `<nburn_install>\docs\NetBurner\platform\Schematics` directory. This schematic can be used for design ideas in your own hardware implementation for power, RS-232, RS-485, and SD Flash card implementation.

8.5.5 Memory Operation

The SBE70LC uses 3 types of memory:

- 2MB Flash memory for non-volatile system configuration storage, user application storage, the application in a compressed format, and the optional EDFS-STD file system (note this is different from the EDFS-FAT flash card file system).
- 8MB SDRAM containing the uncompressed application as well as any application data.
- 384KB on-chip SRAM located in the SAME70 microprocessor.

At power-up the application is decompressed from Flash memory to SDRAM and execution begins. If the application is continuously trapping the device can be recovered using the jumper recovery method as described in the Recover section, or if you have serial communication enabled the boot sequence can be aborted by sending the appropriate serial command when prompted after a power cycle.

8.5.6 Memory Map

Region	Size	Address Range	Description
Boot Loader	6k	0x00400000 to 0x004017FF	Initial boot and application extraction
Config Record	10k	0x00401800 to 0x00403FFF	System and User Configuration Data Flash

Region	Size	Address Range	Description
Cert. Storage	8k	0x00404000 to 0x00405FFF	Optional Security certificate storage
User Flash	8k	0x00406000 to 0x00407FFF	Unstructured binary User Flash Storage
App Flash	1.97M	0x00408000 to 0x005FFFFFFF	Compressed Application and opt file system
Vector Table	-	0x20400000 to 0x2040013F	SRAM - Interrupt Vector Table
Fast App RAM	384k	0x20400180 to 0x2045FFFF	SRAM - Fast Application RAM
Backup RAM	1k	0x40074000 to 0x400743FF	Low power backup RAM
External Bus	-	0x60000000 to 0x60FFFFFFF	Chip Select 0
External Bus	16M	0x61000000 to 0x61FFFFFFF	Chip Select 1
External Bus	16M	0x62000000 to 0x62FFFFFFF	Chip Select 2
External Bus	16M	0x63000000 to 0x63FFFFFFF	Chip Select 3
App RAM	8M	0x70000000 to 0x707FFFFFFF	SDRAM - Standard Application RAM where code is decompressed and variables are allocated
QSPI XIP	-	0x80000000 to 0x80FFFFFFF	QSPI (User add-on) - For utilizing memory Mapped Quad SPI memories

8.6 SOMRT1061 Platform Reference

8.6.1 Introduction

This document provides the memory map and locations of reference materials for those who wish to add additional hardware to their NetBurner device.

8.6.2 Processor Information

The SOMRT1061 uses the NXP i.MX RT1061 microprocessor. The reference manual and datasheet provide in-depth information on the microprocessor, including register settings, bus configuration and timing information. It is located in the `<nburn_install>\docs\NXP` directory of your NetBurner installation.

8.6.2.1 Definitions

Definitions for the MIMXRT1061 can be found at <arch/cortex-m7/cpu/MIMXRT10xx/include/MIMXRT1061.h>

8.6.3 Datasheet and Pins

The datasheet for the SOMRT1061 module includes information on connectors, signal names, and operational parameters. It is located on the [SOMRT1061 product page](#) under Documents and Downloads.

An Excel spreadsheet of pin assignments is also available under the same Datasheet link.

Finally, NetBurner has created an interactive [Pin Configurator tool which is available online](#) as well as in your NNDK installation under `<nburn_install>\docs\NetBurner\Platform\PinConfigurator`

You may also find `pinconstant.h` to be a useful reference.

8.6.4 Development Board Schematic

The DEV-SOMRT1061 development board schematic is located in the `<nburn_install>\docs\NetBurner\platform\Schematics` directory. This schematic can be used for design ideas in your own hardware implementation for power, RS-232, RS-485, and SD Flash card implementation.

8.6.5 SOMRT1061 Platform General Information

The SOMRT1061 uses the NXP i.MX RT1061 microcontroller. The reference manual and datasheet provide in-depth information on the processor, including register settings, bus configuration and timing information. The reference manual is located in the <nburn_install>\docs\NXP directory of your NetBurner installation.

- i.MX RT1061 running at 528MHz (ind temp)
- 8MB SPI Flash (stores compressed application image)
- 32MB PSRAM
- 1MB onboard SRAM:
 - 128k ITCM single cycle instruction fetch memory,
 - 128K DTCM single cycle data fetch memory
 - 768k general purpose fast SRAM

8.6.5.1 The 8MB of Flash uses a Flash File System and is used for all storage purposes:

- Sectors are 8k
- The standard NetBurner configuration, certificate and user flash operate transparently through the file system.
- First sector reserved for file partition table. Note: If you need direct flash access, please contact NetBurner support.
- You can add a secondary Flash File System as demonstrated in the EDFS-STD examples. Multiple partitions are supported. Please refer to the documentation and examples, such as the EDFS-STD multiple partition (multi-part) example.
- Application space varies by use case, but is typically 7.5MB for the compressed application. Items that can reduce the 8MB total space include: file system overhead, certificates, configuration data, secondary file system.

8.6.5.2 External Memory Bus

- Chip selects configurable anywhere in the external bus range

8.6.6 Boot Sequence

At power-up or reset, the application is decompressed from Flash memory to PSRAM, verified by checksum, and execution begins. If the checksum fails or the application crashes, the device will reboot to the Configuration Server to facilitate a recovery by downloading a new application.

If the application causes an issue in which continuous traps occur, or the state of the system is such the Configuration Server cannot run, a recovery hardware jumper procedure can be used to reset the device.

Note

It is always a good idea to have a serial port terminal connected to the boot/debug serial port to view status messages, or use the 'A' to abort the boot sequence command when prompted to abort application execution and boot to the Configuration Server.

8.6.7 UART Signals and Port Number Mapping

There can be a maximum of 7 serial port configured on the device. Note that each signal pin on the microprocessor can be configured for up to 5 different functions, so to achieve the maximum number of serial ports some peripheral functions will be unavailable.

Please refer to the Open All Serial Ports example for hardware and software port mapping: [Open All UARTs](#). Port number mapping for software API calls is defined in [SOMRT1061/include/serial_platdefs.h](#) in the `<nburn_install>\platform` folder.

8.7 Recovery: MODM7AE70, SBE70LC

8.7.1 Introduction

Applicable Hardware Platforms: MODM7AE70 and SBE70LC.

Device recovery in the event of an errant application running on the device is an important part of the development process. This document describes recovery procedures for a device stuck in a repeating trap/reset condition, or a totally unresponsive device.

8.7.2 Repeating Traps or Resets

The easiest method of recovery is to abort the user application boot process and run only the Configuration Server. The Configuration server will enable a new application to be downloaded using:

- The Configuration Server web page at `<device IP>:20034`
- The nbupdate utility
- The wget utility
- The device serial port

8.7.2.1 Step 1: Abort to the Configuration Server

1. Run a serial terminal such as MTTTY and connect to the device serial port.
2. Power cycle the module.
3. Press "A" when the message, `Type "A" to Abort boot` is displayed. Note that "A" is the default abort command, but the command is configurable in the Configuration Server settings. If the abort command is different, use that command instead of "A".
4. Wait for the `>` prompt to display.

The Configuration Server is now active on both network and the device serial port.

8.7.2.2 Step 2: Download a New Application

At this point a new application can be downloaded through the device web page, nbupdate or serial port.

8.7.2.2.1 Web Page

1. Locate the device using [discover.netburner.com](#), the `localdiscover` utility, or open a web browser and type the device address in the URL field: `<device IP>:20034`. For example, `192.168.1.10:20034`.
2. In the Update Application section, select the new .bin file to download and select the Send File button.

8.7.2.2.2 NBUpdate

1. Open a command prompt.
2. Type `nbupdate <file name> <device IP>`

8.7.2.2.3 Serial Port

Note

MTTTY is a serial terminal program. You should be able to use any serial terminal program of your choice. When starting the download the binary file must be sent as text or raw data.

Warning

The .bin file is a binary file. You must have serial software flow control disabled. If using MTTY, from the main menu select `TTY > Flow Control` and uncheck both `Xon/Xoff Input Control` and `Xon/Xoff Input Control`. Otherwise the data stream will be corrupted.

1. At the `>` prompt, type `fla` into the serial prompt and then hit enter.
2. At the message 'Begin Download', start the file download (if using MTTY, type the `F5` key).
3. When the file window opens, navigate to the .bin application file that you want to load.
4. Hit 'Open', and the application will start the load process.

8.7.3 Unresponsive Device or No Serial Port Access

In a situation in which the device is completely unresponsive and aborting to the Configuration Server is not possible, or if there is no access to the device serial port, shorting two hole locations on the device PCB (also called "jumper recovery") will reset it to a factory default state. The PCB hole locations are 0.1 inches apart so that a shorting jumper can be used. This procedure will erase flash memory including the configuration and application. The device will then be ready to receive a new application using any of the procedures described in the download a New Application section.

1. Power off the device.
2. Short the jumper recovery PCB locations with a shorting jumper or wire.
3. Power on the device.
4. Wait for the device to finish erasing the flash and loading the default application. Progress is indicated by the two blinking LEDs on the device (D1 and D2). When the LEDs stop blinking the reset is complete. For the SBE70LC, the jumper should be removed after 2 seconds. If it is left inserted for 30 seconds or more, it will continually reset.
5. Remove the short jumper or wire.

8.7.3.1 Recovery Jumper Locations

8.7.3.1.1 MODM7AE70

The reset jumper location is labeled "Recovery", located on the end of the board opposite the RJ-45 connector.

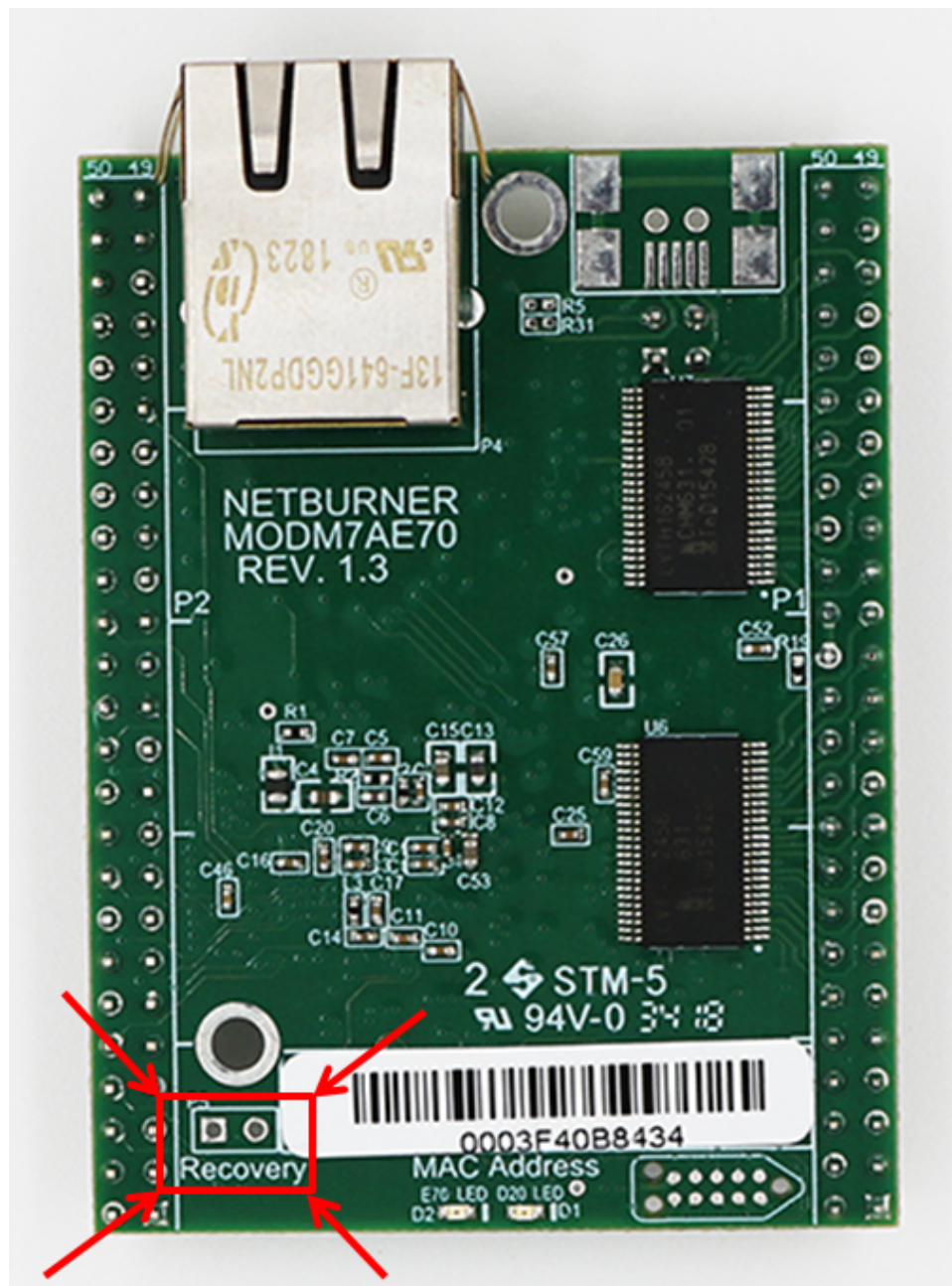


Figure 8.1 MODM7AE70 Recovery Jumper Location

8.7.3.1.2 SBE70LC

The reset jumper location is labeled "JP6", located on the end of the board opposite the RJ-45 connector.

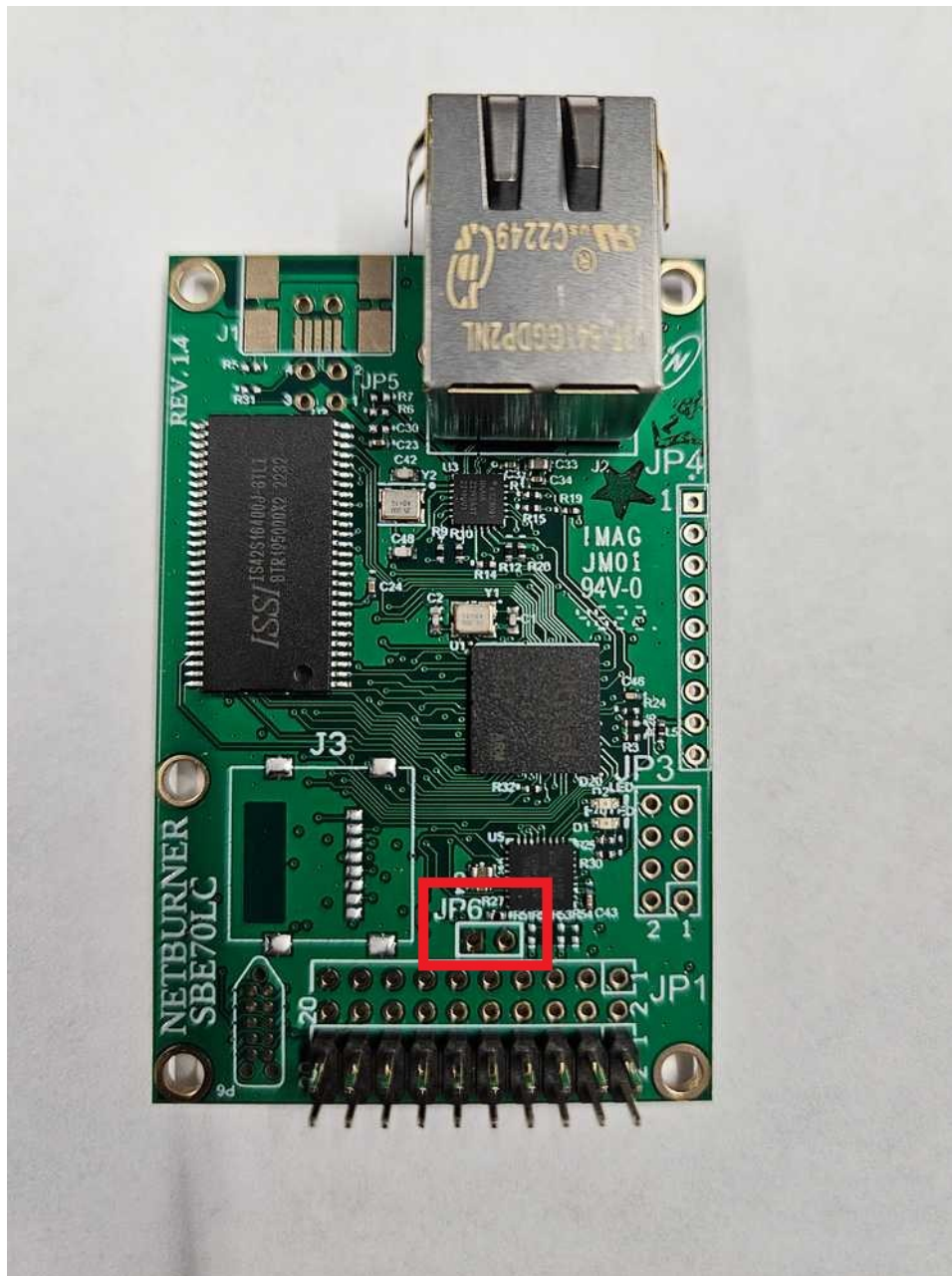


Figure 8.2 SBE70LC Recovery Jumper Location

8.8 Recovery: MOD5441x, NANO54415, SB800EX

NetBurner 5441x based devices include: MOD54415, MOD54417, NANO54415 and the SB800EX. These devices have a Recovery Jumper. This is a 2 hole pin location on the circuit board. When shorted during the boot process, the device will boot to the Alternate Boot Monitor rather than the application. If the configuration section of flash has been corrupted, it will also use a safe default version instead of what is in flash memory.

8.8.1 Step 1: Abort to the Alternate Boot Monitor

There is the primary boot monitor, accessible by entering the 'A' (0x41) character immediately on boot. This should allow you to recover the module normally. If for some reason that fails, or you cannot enter the 'A' character, the

have a secondary boot monitor to allow the user to recover from (virtually) all software or configuration faults. The procedure is:

1. Power off the module.
2. Short the 2 pins at the alternate boot jumper location (see below).
3. Power on the module.
4. Wait for the module to finish boot and enter the monitor.
5. Remove the short on the alternate boot jumper location.

8.8.2 Recovery Jumper Locations

8.8.2.1 NANO54415

The boot jumper is a pair of circular pads located near the middle line of the board near the connector.

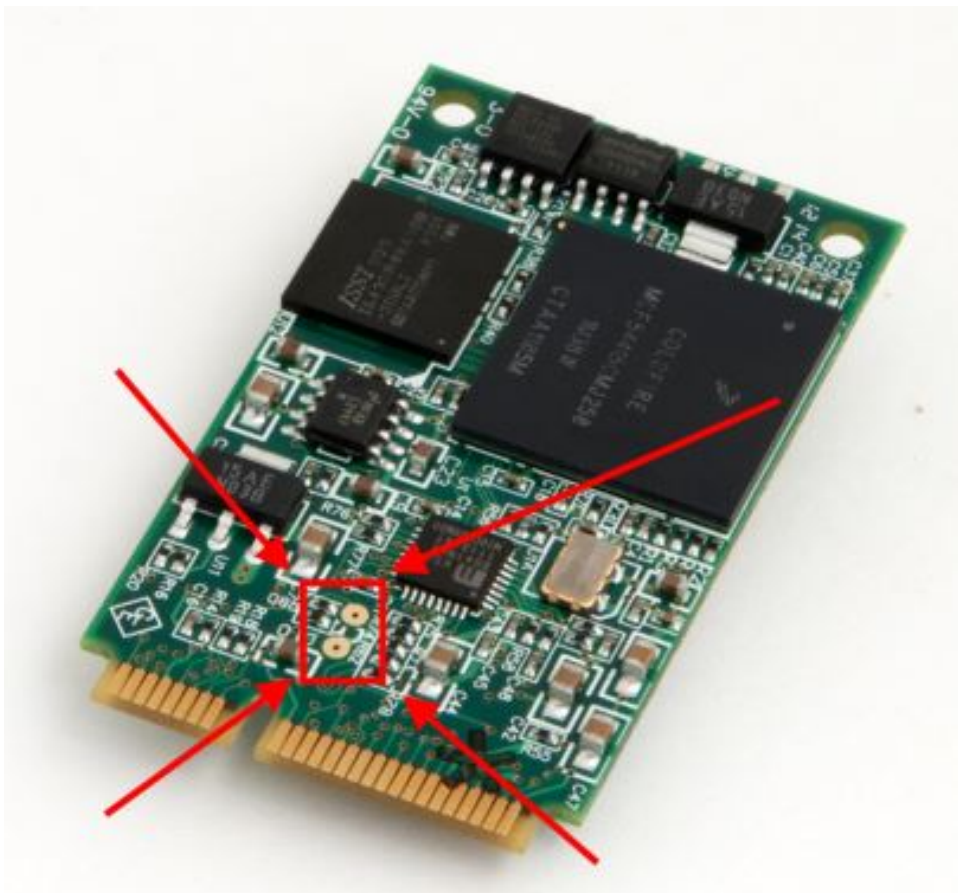


Figure 8.3 NANO54415 Recovery Jumper Location

8.8.2.2 MOD5441x

The boot jumper is the unpopulated header 'TP1', located near the Ethernet jack.

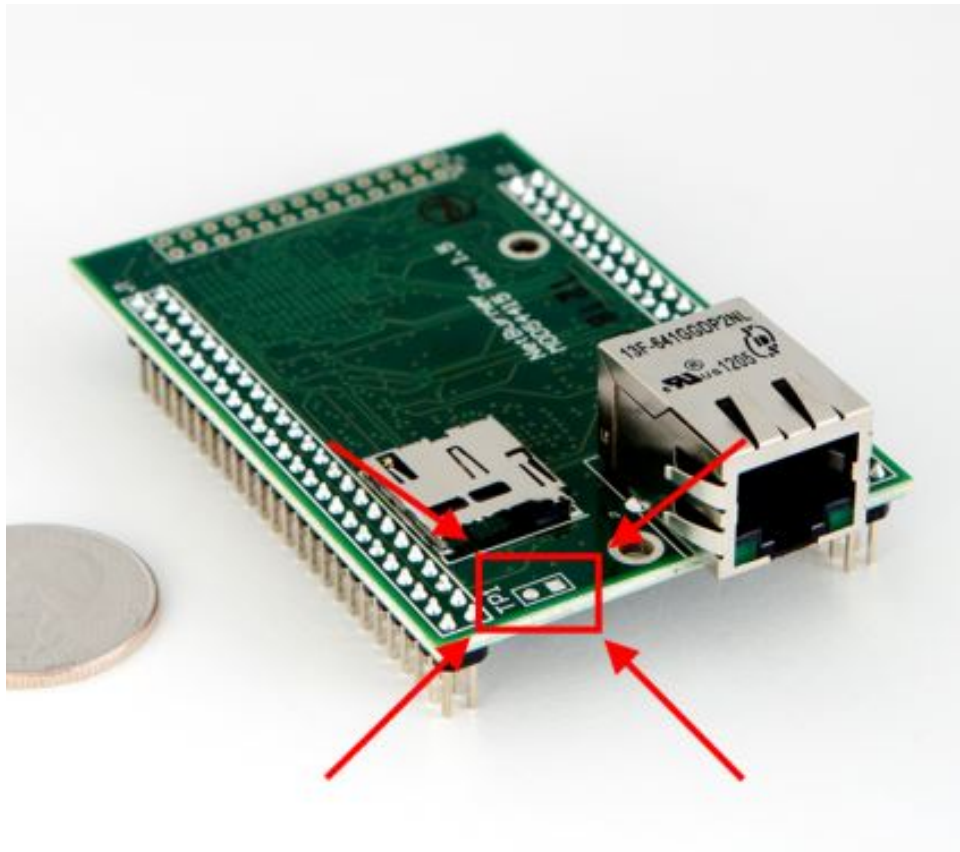


Figure 8.4 MOD5441x Recovery Jumper Location

8.8.2.3 SB800EX

The boot jumper is the unpopulated header 'JP1', located near the center of the board.

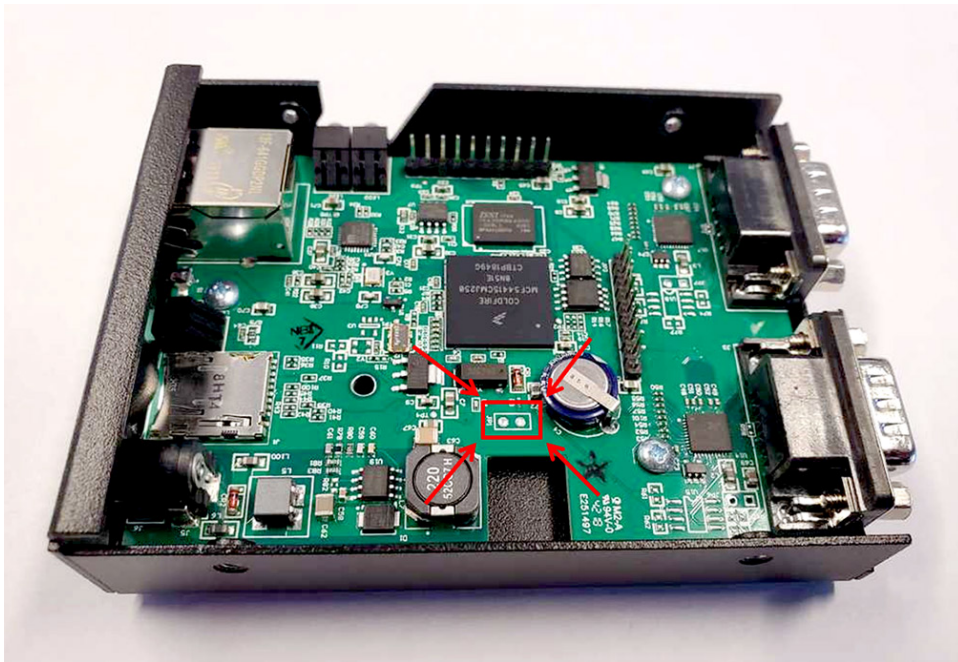


Figure 8.5 SB800EX Recovery Jumper Location

8.8.3 Step 2: Download a New Application

Once the Alternate Boot Monitor is running you should be able to locate your device using:

8.8.3.1 Network

The 5441x devices predate the NetBurner 3.x Development Tools and network recovery requires the use of the AutoUpdate utility. Run the AutoUpdate utility and select the device and new application to download. Autoupdate requires `_APP.s19` file format. The original, factory application is a safe application to load on to a device that needs to be recovered. It can be found in the `platform\<PLATFORM>\original` directory.

8.8.3.2 Serial Port

Note

MTTTY is a serial terminal program. You should be able to use any serial terminal program of your choice. When starting the download the binary file must be sent as text or raw data.

Warning

The `.bin` file is a binary file. You must have serial software flow control disabled. If using MTTY, from the main menu select `TTY > Flow Control` and uncheck both `Xon/Xoff Input Control` and `Xon/Xoff Input Control`. Otherwise the data stream will be corrupted.

1. At the `NB>` prompt type `fla` then hit enter.
2. At the message 'Begin Download', start the file download (if using MTTY, type the `F5` key).
3. When the file window opens, navigate to the `.bin` application file that you want to load.
4. Hit 'Open', and the application will start the load process.

Once the download is complete the device will reboot and run the new application.

8.9 Recovery: SOMRT1061

8.9.1 Introduction

Applicable Hardware Platforms: SOMRT1061

Device recovery in the event of an errant application running on the device is an important part of the development process. This document describes recovery procedures for a device stuck in a repeating trap/reset condition, or a totally unresponsive device.

8.9.2 Boot Components and Features Overview

An overview of the key components and features of the boot process are listed below. The remainder of this article will provide further details and usage on each.

- **Boot Monitor.** Executes at power-up or reset. Decompresses Application, performs system verification and jumps to application to begin execution.
- **Application.** Your custom program/application that runs when things are working properly.
- **Configuration Server.** Runs as part of the Application. For recovery, the boot sequence can be aborted in the initialization process (the `init()` function in `UserMain()` will not return), enabling you to change configuration parameters or upload a new application. Note that the application must be able to boot and run to the `init()` function to use this feature. The Configuration Server can be accessed through the web server interface or the serial console port. A new application can be uploaded to the device through any of the following:
 - Configuration Server web page at `<device IP>:20034`
 - `nbupdate` utility
 - `wget` utility
 - serial console interface
- **Alternate Boot Monitor.** Used as part of a recovery method when aborting to the Configuration Server is not possible. There are two use cases:
 - If you can abort to the Configuration Monitor but are unable to load a new application for some reason, you can issue the command "ABOOT" through the serial console and the device will reboot to the Alternate Boot Monitor without having to use the jumper recovery procedure. You can then issue the "Erase All" command to erase Flash memory.
 - If there is no application programmed into the device, it will boot to the Alternate Boot Monitor.
 - If the application is crashing before `UserMain()` starts, such as a crashing global constructor, you will need to boot to the Alternate Boot Monitor with the recovery jumper procedure described later in this article.

8.9.3 Boot Sequence Overview

8.9.3.1 Normal Boot

For a properly running application. Boot Monitor starts, decompresses application to RAM, performs system checks, then jumps to the start of the application.

8.9.3.2 Incorrect Network Settings

Symptom:** The application boots and runs, but the network IP settings are such that the device cannot communicate. Resolution:** Abort to the **Configuration Server** and modify network settings with serial console interface.

8.9.3.3 Application Starts, But Traps After Application Initialization

Symptom:** The application boots and can run to the point of initialization (the `init()` function in `UserMain()`), but traps afterwards.

Resolution:** Abort to the **Configuration Server** and upload corrected application.

8.9.3.4 Application Starts, But Traps Before Application Initialization

Symptom:** The application traps immediately upon boot. The Configuration Server is not accessible.

Resolution:** Abort to the **Alternate Boot Monitor** and upload corrected application. Access through the serial console interface or telnet.

8.9.4 Procedure to Abort to the Configuration Server

Abort the boot sequence and run the Configuration Server, accessible on the web interface and serial console interface.

- Run a serial terminal such as MTTY and connect to the device serial console interface.
- Reset or power cycle the module.
- Press the "A" key when the message: `Type "A" to Abort` is displayed. Note that "A" is the default abort command, but the command is configurable in the Configuration Server settings. If the abort command is different, use that command instead of "A".
- At the `>` prompt you can now type serial console commands, or use the web interface if you have network connectivity.
- A new application can be uploaded as described in the Uploading a New Application section.

8.9.5 Uploading a New Application

The following application update methods can be used from the Configuration Server or Alternate Boot Monitor.

8.9.5.1 Web Page Upload

- Locate the device using `discover.netburner.com`, the `localdiscover` utility, or open a web browser and type the device address in the URL field: `<device IP>:20034`. For example, `192.168.1.10:20034`.
- In the Update Application section, select the new .bin file to download and select the Send File button.

8.9.5.2 NBUpdate

- Open a command prompt.
- Type `nbupdate <file name> <device IP>`

8.9.5.3 Serial Console Interface

Note

MTTY is a serial terminal program. You should be able to use any serial terminal program of your choice. When starting the download the binary file must be sent as text or raw data.

Warning

The .bin file is a binary file. You must have serial software flow control disabled. If using MTTTY, from the main menu select `TTY > Flow Control` and uncheck both `Xon/Xoff Input Control` and `Xon/Xoff Input Control`. Otherwise the data stream will be corrupted.

- At the `>` prompt, type `fla` into the serial prompt and then hit enter.
- At the message 'Begin Download', start the file download (if using MTTTY, type the `F5` key).
- When the file window opens, navigate to the .bin application file that you want to load.
- Hit 'Open', and the application will start the load process.

8.9.6 Procedures to Abort to the Alternate Boot Monitor

There are three instances in which the Alternate Boot Monitor is needed for device recovery. A new application can be uploaded as described in the Uploading a New Application section.

8.9.6.1 No Valid Application

If no application had been programmed into the device, or the checksum verification fails when an application is decompressed from Flash to RAM, the Boot Monitor will automatically jump to the Alternate Boot Monitor.

8.9.6.2 Configuration Server Access Operational

If the Configuration Server can run but code updates are not working correctly, the "ABOOT" command can be used on the Serial Console Interface to invoke the Alternate Boot Monitor without having to perform the jumper recovery procedure.

8.9.6.3 Application Traps Before UserMain() Starts

If the application is crashing before `UserMain()` starts, such as a crashing global constructor, access to the Configuration Server is not possible as it is part of the application. Your only choice will be to access the Alternate Boot monitor with the recovery jumper. Please see the recovery jumper procedure section.

8.9.7 Hardware Jumper Recovery

In a situation in which the device is completely unresponsive, continually trapping prior to the "Type A to Abort" prompt, or if there is no access to the device's serial console interface, shorting the recovery pad to ground on the device (referred to as "jumper recovery") can be used to abort the boot process and run the Alternate Boot Monitor. The device will then be ready to receive a new application using any of the procedures described in the download a New Application section.

8.9.7.1 Recovery Jumper Operation

The recovery pad can be shorted with a jumper in two ways:

- **Short Hold:** Boot to Configuration Server. Loads existing configuration and mount file system. Remove shorting jumper right after reset or power-on. The device should now be running the Configuration Server which can be accessed via the serial console interface or web interface.

- **Long Hold:** Boot to Alternate Boot Monitor. Do not load existing configuration or mount file system. Hold shorting jumper in place throughout the entire boot process until the Alternate Boot Monitor command prompt appears in the serial console interface or the network LEDs on the RJ-45 jack become active. Application Flash is not accessible except for the `Erase All` command in the Alternate Boot Monitor. After erasing Flash memory, the next time the device boots it will reset to factory defaults.

Note

A Long Hold will initialize the device to factory default settings, such as network configuration set to [DHCP](#), AutoIP enabled and Local Discovery enabled.

8.9.7.2 Procedure

- Power off the device.
- Short the jumper recovery pad to the nearby ground pad or some other accessible ground location with a shorting jumper or wire.
- Reset or power on the device.
- Hold shorting jumper in place for either a Short Hold or Long Hold.
- Remove the jumper or wire.

8.9.7.3 Recovery Jumper Location

The recovery pad and ground pad are located on the corner of the module between P2 and P3. Using a pair of tweezers may be easiest for shorting the two pads. Alternatively, the recovery pad can be shorted to any ground pin to achieve the same result, which may be easier, especially if using the DEV-SOMRT1061 development board.

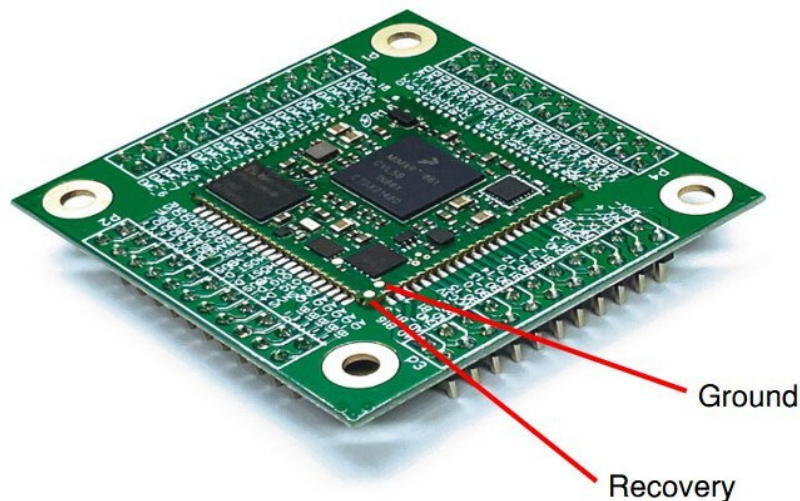


Figure 8.6 SOMRT1061 Recovery Jumper Location

8.10 ARM Cache Management

When using some features like DMA or otherwise experiencing cache coherency issues and instability, you may want to disable memory caching.

8.10.1 Details

What happens if memory changes a region AND there is a delayed write to the same region? Who wins? Is this an issue for things like ethernet where both CPU and DMA hardware can both change an area? Whoever writes to RAM later wins: For example if the CPU has the address in the cache and writes to it, and then the Hardware writes to it, then the CPU evicts the cache line and flushes, the CPU wins. If the CPU writes to the RAM first, then the Hardware writes to it, the Hardware wins. Either way, this will be a poorly defined system that will fail in real operations.

Whenever DMA is expected to write to memory, the CPU shall not write to that region. Whenever the CPU needs to read data that DMA has written, unless the region is known to be non-cached, the code shall issue a cache invalidation for the region in question.

Whenever the CPU writes to a region that is cached and needs DMA to read a coherent state of the memory, the software shall issue a cache cleaning for the region in question.

8.10.2 Instructions

8.10.2.1 Turning cache on/off

- `SCB_EnableDCache();`
- `SCB_DisableDCache();`
- `SCB_EnableICache();`
- `SCB_DisableICache();`

8.10.2.2 Declaring static objects non cacheable

Non cacheable objects only live in On Chip FlexRAM which is SRAM. PSRAM is all cached.

To declare a variable as non-cached, use the variable tag `DO_NOT_CACHE`, akin to `FAST_USER_VAR: uint32_t foo DO_NOT_CACHE;`

Note

You cannot assign a static value to non-cached objects.

8.10.2.3 Cleaning Cache

(Making sure all cache -> memory transactions are complete)

- For whole cache:


```
asm("dsb");
SCB_CleanDCache();
```
- For a defined region of memory:


```
asm("dsb");
SCB_CleanDCache_by_Addr(<start address>, <buffer size>);
```

8.10.2.4 Invalidating Cache

(Making sure memory changes make it to cache)

- For whole cache.


```
asm("dsb");
SCB_InvalidateDCache();
```
- For a defined region of memory:


```
asm("dsb");
SCB_InvalidateDCache_by_Addr(<start address>, <buffer size>);
```

Chapter 9

Production & Deployment

9.1 Production Releases and Customer Deployment

Once product development is complete, there are a number of choices for device location and configuration:

- How to find the device on a network
- How to program the application into devices in production
- How to configure devices in production
- How Customers can change configuration settings

Note

Utility source code is located in `\nburn\pctools`, and built versions are in `\nburn\pbin`. You may modify or rebrand these utilities and distribute to your end customers, so long as they are only run with NetBurner target hardware.

9.2 Finding a Device on a Network

A device can be located the following ways:

- If the device and computer have Internet access, open a web browser and go to discover.netburner.com
- Use the `localdiscover` utility. This is a multi-platform utility written in Golang for Windows, macOS and Linux. The compiled version can be distributed to your customers, but not the Golang source code.
- The Python `find` utility, located in `\nburn\pctools\find`.
- Use the debug/console serial port

9.3 Application Programming

Applications are programmed into flash memory by sending an application image to the device's configuration server, located at its IP address on port 20034. This can be done in a number of ways, all of which are multi-platform (Windows, macOS, and Linux):

- A web browser from the device's configuration server web page
- The nbupdate command-line utility
- The WGET command-line utility
- A serial port using the serial command processor

9.3.1 Configuration Web Page

To send a file from a web browser, enter the URL of your device. The URL format is "<ip address>:<port number>", such as: "10.1.1.100:20034". Once at the web page, browse for the application image file and select "Update".

9.3.2 nbupdate Utility

nbupdate is a multi-platform command-line utility written by NetBurner. The format is: `nbupdate <filename> <ip address>`. For example, `nbupdate myApp.bin 10.1.1.196`, or `nbupdate myApp.s19 10.1.1.196`.

If the environment variable DEVIP is set the IP address parameter is not required. For example, `SET DEVIP=10.1.1.196`. Now you can call nbupdate with only the application file name. To determine the current setting of DEVIP, type "SET DEVIP".

9.3.3 wget Utility

WGET is an industry standard utility that allows you to download files over TCP/IP protocols FTP, HTTP and HTTPS. If you are using macOS or Linux, WGET is usually already installed (or can using your package manager). Windows users will need to install it, please refer to <https://www.gnu.org/software/wget/>.

The WGET utility can be used for a number of things:

- Updating the device application image
 - Retrieving device configuration information in JASON
 - Updating device configuration information in JASON
- In the examples below we use an example device IP address of 10.1.1.60, and an application name of myApplication.bin.

9.3.3.1 Program an Application

To program a new application into the device:

```
wget --post-file=myApplication.bin 10.1.1.60:20034/appupdate.html
```

All updates are processed by the Configuration Server running on the device. The `--post-file` specifies a POST operation is to be performed. The IP address and configuration port number are next, along with the server post destination that will always be appupdate.html.

If using authentication with a username of "user", and password of "pass":

```
wget --auth-no-challenge --user=user --password=pass --post-file=myApplication.↵
bin 10.1.1.60:20034/appupdate.html
```


9.3.3.2 Read The Configuration

Read the current configuration into a file named config.txt: `wget 10.1.1.60:20034/Config`.

In this case the destination is Config, rather than appupdate.html.

If you would like to specify a directory to store the file, use the `-P` option: `wget 10.1.1.60:20034/Config -P \myDir`

9.3.3.3 Program New Configuration Settings

The easiest way to do this is to first read the configuration and save to a file so you have a template to work with, then edit the settings you need. Then post the configuration file to each new device:

```
wget --post-file=newconfig 10.1.1.60:20034/Config
```

9.4 NetBurner SSL Certificate Upload Commands

9.4.1 Overview

NetBurner devices provide a web interface for uploading SSL/TLS certificates and private keys. The web form submits to the `/httpsform` endpoint using HTTP POST with `multipart/form-data` encoding, expecting two file fields:

- `certificateFile` - The SSL certificate file (*.crt)
- `keyFile` - The private key file (*.key)

9.4.2 Requirements

- RSA key sizes must be at least 512 and no more than 4096 bits in OpenSSH/OpenSSL format
- ECDSA key sizes must be at least 192 and no larger than 256 bits
- Files should be in standard certificate (*.crt) and key (*.key) formats

9.4.3 Linux/Unix Commands

9.4.3.1 Using curl (Recommended)

```
curl -X POST \
  -F "certificateFile=@certificate.crt" \
  -F "keyFile=@private.key" \
  http://YOUR_DEVICE_IP/httpsform
```

9.4.3.2 Using wget (Simple approach)

```
# Create the multipart body in a temporary file
cat > /tmp/ssl_upload.txt « 'EOF'
-----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="certificateFile"; filename="certificate.crt"
Content-Type: application/x-x509-ca-cert

EOF

cat certificate.crt » /tmp/ssl_upload.txt

cat » /tmp/ssl_upload.txt « 'EOF'

-----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="keyFile"; filename="private.key"
Content-Type: application/x-pem-file

EOF

cat private.key » /tmp/ssl_upload.txt

echo "-----WebKitFormBoundary7MA4YWxkTrZu0gW--" » /tmp/ssl_upload.txt

# Submit with wget
wget --post-file=/tmp/ssl_upload.txt \
```

```
--header="Content-Type: multipart/form-data; boundary=---WebKitFormBoundary7MA4YWxkTrZu0gW" \
http://YOUR_DEVICE_IP/httpsform

# Clean up
rm /tmp/ssl_upload.txt
```

9.4.4 Windows PowerShell Commands

9.4.4.1 Method 1: Using Invoke-RestMethod (Recommended)

```
$deviceIP = "YOUR_DEVICE_IP"
$certFile = "certificate.crt"
$keyFile = "private.key"

$form = @{
    certificateFile = Get-Item -Path $certFile
    keyFile = Get-Item -Path $keyFile
}

Invoke-RestMethod -Uri "http://$deviceIP/httpsform" -Method Post -Form $form
```

9.4.4.2 Method 2: Using Invoke-WebRequest

```
$deviceIP = "YOUR_DEVICE_IP"
$certFile = "certificate.crt"
$keyFile = "private.key"

$form = @{
    certificateFile = Get-Item -Path $certFile
    keyFile = Get-Item -Path $keyFile
}

Invoke-WebRequest -Uri "http://$deviceIP/httpsform" -Method Post -Form $form
```

9.4.4.3 Method 3: Manual multipart construction

```
$deviceIP = "YOUR_DEVICE_IP"
$certFile = "certificate.crt"
$keyFile = "private.key"
$boundary = "---WebKitFormBoundary7MA4YWxkTrZu0gW"

# Read file contents
$certContent = Get-Content -Path $certFile -Raw
$keyContent = Get-Content -Path $keyFile -Raw

# Build multipart body
$body = @"
--$boundary
Content-Disposition: form-data; name="certificateFile"; filename="certificate.crt"
Content-Type: application/x-x509-ca-cert

$certContent
--$boundary
Content-Disposition: form-data; name="keyFile"; filename="private.key"
Content-Type: application/x-pem-file

$keyContent
--$boundary--
"@

# Convert to bytes
$bodyBytes = [System.Text.Encoding]::UTF8.GetBytes($body)

# Set headers
$headers = @{
    "Content-Type" = "multipart/form-data; boundary=$boundary"
}

# Send request
Invoke-RestMethod -Uri "http://$deviceIP/httpsform" -Method Post -Body $bodyBytes -Headers $headers
```

9.4.5 Windows Command Prompt

9.4.5.1 Method 1: Using curl (Windows 10/11)

```
curl -X POST -F "certificateFile=@certificate.crt" -F "keyFile=@private.key" http://YOUR_DEVICE_IP/httpsform
```

9.4.5.2 Method 2: PowerShell one-liner from Command Prompt

```
powershell -Command "Invoke-RestMethod -Uri 'http://YOUR_DEVICE_IP/httpsform' -Method Post -Form
@{certificateFile = Get-Item -Path 'certificate.crt'; keyFile = Get-Item -Path 'private.key'}"
```

9.4.5.3 Method 3: Batch file approach

Create upload_ssl.bat:

```
@echo off
set DEVICE_IP=YOUR_DEVICE_IP
set CERT_FILE=certificate.crt
set KEY_FILE=private.key
set BOUNDARY=----WebKitFormBoundary7MA4YWxkTrZu0gW

echo Creating multipart data...

(
echo --%BOUNDARY%
echo Content-Disposition: form-data; name="certificateFile"; filename="%CERT_FILE%"
echo Content-Type: application/x-x509-ca-cert
echo.
type "%CERT_FILE%"
echo.
echo --%BOUNDARY%
echo Content-Disposition: form-data; name="keyFile"; filename="%KEY_FILE%"
echo Content-Type: application/x-pem-file
echo.
type "%KEY_FILE%"
echo.
echo --%BOUNDARY%--
) > temp_upload.txt

powershell -Command "$body = Get-Content -Path 'temp_upload.txt' -Raw; $headers = @{'Content-Type' =
'multipart/form-data; boundary=%BOUNDARY%'}; Invoke-RestMethod -Uri 'http://%DEVICE_IP%/httpsform'
-Method Post -Body ([System.Text.Encoding]::UTF8.GetBytes($body)) -Headers $headers"

del temp_upload.txt
echo Upload complete!

Then run:
upload_ssl.bat
```

9.4.6 Usage Notes

1. **Replace placeholders:** Change YOUR_DEVICE_IP to your actual NetBurner device IP address
2. **File paths:** Adjust certificate and key file names/paths as needed
3. **File location:** Ensure certificate and key files are accessible from the command execution directory
4. **Network access:** The NetBurner device must be reachable on your network
5. **File formats:** Use standard OpenSSL/OpenSSH certificate and key formats

9.4.7 Recommended Approaches

- **Linux/Unix:** Use curl for simplicity and reliability
- **Windows PowerShell:** Use Invoke-RestMethod with -Form parameter
- **Windows Command Prompt:** Use curl if available, otherwise PowerShell one-liner

9.4.8 Troubleshooting

- Verify the device IP address is correct and reachable
- Ensure certificate and key files are in the correct format
- Check that files exist in the specified paths
- Confirm the NetBurner device web interface is accessible
- Verify RSA key size requirements (512-4096 bits) or ECDSA (192-256 bits)

9.4.9 Security Considerations

- Use secure connections when possible
- Protect private key files with appropriate file permissions
- Consider using HTTPS if the NetBurner device supports it
- Store certificates and keys securely

9.5 Customer Configuration

Once deployed, your customers can modify the configuration by:

- Using the interactive web interface at 10.1.1.60:20034. Note that you can create your own web interface to override the default NetBurner interface, and use your custom brand.
- If connected to the Internet, a web browser at: discover.netburner.com, which provides a link to the device configuration page and also the device's main web page on port 80.
- The NetBurner localdiscover utility, which provides a link to the device configuration page, and also the device's main web page on port 80.
- Using WGET
- Your application can provide the means to change configuration

Chapter 10

Programmers Guide

10.1 Overview

This guide provides comprehensive information on fundamental concepts in the NetBurner development system. The documents cover both introductory material and advanced topics to support embedded network application development.

10.2 Table of Contents

- [About This Guide](#)
- [Introduction](#)
- [Config Server Programming Guide](#)
- [Custom Branding](#)
- [IPv4/IPv6 Dual Stack Guide](#)
- [EFS Programming Guide](#)
- [File Descriptors](#)
- [HTML Processing](#)
- [Interrupt Handling](#)
- [JSON Lexer](#)
- [NetBurner RTOS](#)
- [Network Protocols TCP and UDP](#)
- [SSL/TLS and Certificates](#)
- [SSH Programming Guide](#)
- [Web Server](#)
- [Web Sockets](#)

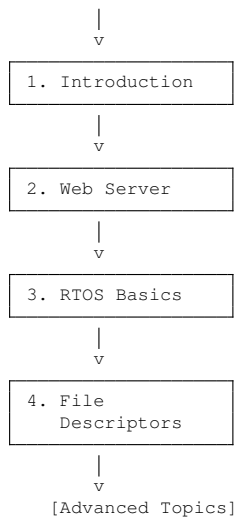
10.3 About This Guide

10.3.1 Guide Navigation

10.3.1.1 Quick Start Path

For developers new to NetBurner, follow this learning path:

[Start Here](#)



10.3.1.2 By Use Case

10.3.1.2.1 Web Application Development

- Introduction to NetBurner Programming
- Web Server Programming
- HTML Processing Techniques
- Configuration Server

10.3.1.2.2 Real-Time Systems

- NetBurner RTOS
- Interrupt Handling
- File Descriptors

10.3.1.2.3 Network Programming

- Network Protocols Guide
- IPv4-IPv6 Dual Stack
- SSL/TLS Security
- SSH Protocol

10.3.1.2.4 Data Management

- JSON Lexer
- EDFS File System
- Configuration Server

10.3.1.2.5 Product Deployment

- Custom Branding Options
- Configuration Management
- Security Implementation

10.3.2 Document Descriptions

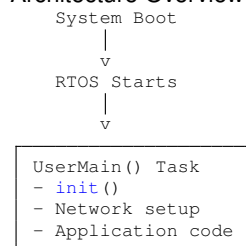
10.3.2.1 Introduction to NetBurner Programming

Focus:** Getting started with NetBurner development Key Topics:**

- Development environment setup
- First application creation
- Template application structure
- System initialization
- Core functions and constants

When to Read:** Start here if you are new to NetBurner

Architecture Overview:**



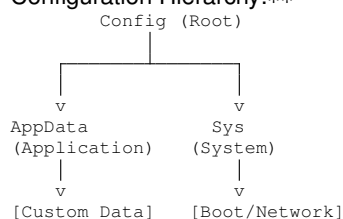
10.3.2.2 Configuration Server

Focus:** Device configuration and settings management Key Topics:**

- Configuration system architecture
- Web-based configuration interface
- Serial configuration access
- AppData structure usage
- System settings management

When to Read:** When you need persistent device settings

Configuration Hierarchy:**



10.3.2.3 Custom Branding

Focus:** OEM customization and white-label solutions Key Topics:**

- Logo replacement options
- Custom web interfaces
- Physical device labeling
- Discovery utility customization

- Cloud infrastructure replacement

When to Read:** When deploying products with your brand

Branding Layers:**

Your Brand Identity
Custom Web Interface
Device Labels
Network Discovery
NetBurner Platform

10.3.2.4 IPv4-IPv6 Dual Stack

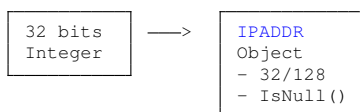
Focus:** Dual-stack networking implementation Key Topics:**

- IPADDR object changes
- IPv6 address assignment
- DNS resolution strategies
- Migration from IPv4-only code
- Platform-specific considerations

When to Read:** When supporting both IPv4 and IPv6

IP Address Evolution:**

IPv4 Only (Pre-2.8.0)	Dual Stack (2.8.0+)



10.3.2.5 EFFS File System

Focus:** Embedded Flash File System operations Key Topics:**

- File system initialization
- File operations (open, read, write, close)
- Directory management
- Flash memory considerations
- Performance optimization

When to Read:** When using file storage in your application

File System Stack:**

Application File I/O
POSIX-like API
EFFS Implementation
Flash Memory Hardware

10.3.2.6 File Descriptors

Focus:** Unified I/O abstraction layer Key Topics:**

- File descriptor architecture
- Standard I/O operations
- TCP socket integration
- Serial port access
- Custom I/O driver creation

When to Read:** When implementing I/O operations or custom drivers

File Descriptor Map:**

FD Range Purpose

0-2	>	Standard I/O
3-4	>	Serial Ports
5-128	>	TCP Sockets
129-250	>	Custom Drivers

10.3.2.7 HTML Processing

Focus:** Dynamic web content generation Key Topics:**

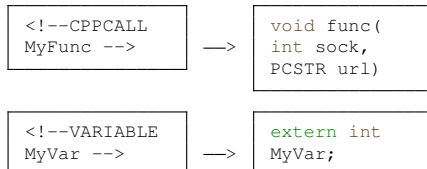
- CPPCALL tag usage
- VARIABLE tag implementation
- INCLUDE tag for linking
- Custom data type display
- Web server integration

When to Read:** When creating dynamic web interfaces

Dynamic Content Flow:**

HTML File

C++ Code



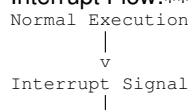
10.3.2.8 Interrupt Handling

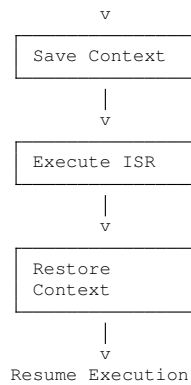
Focus:** Hardware interrupt processing Key Topics:**

- Preemptive interrupt architecture
- ISR implementation patterns
- Platform-specific setup
- Interrupt priorities
- Safe ISR practices

When to Read:** When handling hardware events and interrupts

Interrupt Flow:**





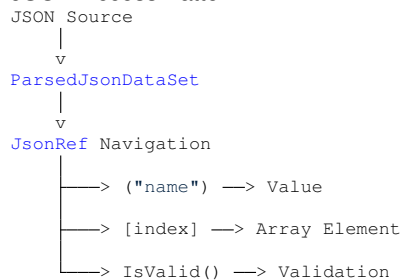
10.3.2.9 JSON Lexer

Focus:** JSON parsing and data access Key Topics:**

- [ParsedJsonDataSet](#) usage
- [JsonRef](#) navigation
- Type conversion
- Validation functions
- Performance optimization

When to Read:** When parsing JSON data from web APIs or files

JSON Access Pattern:**



10.3.2.10 NetBurner RTOS

Focus:** Real-Time Operating System concepts Key Topics:**

- Task creation and management
- Priority-based scheduling
- Inter-task communication
- Synchronization primitives
- System configuration

When to Read:** When implementing multi-tasking applications

RTOS Components:**

User Tasks
- Semaphores - Mailboxes - Queues - Flags
Task Scheduler
Hardware Platform

10.3.2.11 Network Protocols

Focus:** TCP/IP stack and protocol implementation Key Topics:**

- TCP socket programming
- UDP communication
- [MQTT](#) client usage
- Protocol best practices
- Network debugging

When to Read:** When implementing network communication

Network Stack:**

Application Protocols (HTTP, MQTT , Custom)
Transport Layer (TCP, UDP)
Network Layer (IPv4, IPv6)
Link Layer (Ethernet, WiFi)

10.3.2.12 SSL-TLS Security

Focus:** Secure communication implementation Key Topics:**

- SSL/TLS configuration
- Certificate management
- Secure server setup
- Client authentication
- Security best practices

When to Read:** When implementing secure communications

Security Layers:**

Application Data
SSL/TLS Encryption
TCP Transport
IP Network

10.3.2.13 SSH Protocol

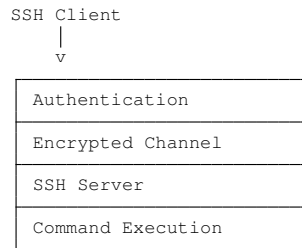
Focus:** Secure Shell implementation Key Topics:**

- SSH server configuration
- Key management
- Secure remote access
- Authentication methods

- Session management

When to Read:** When providing secure remote access

SSH Architecture:**



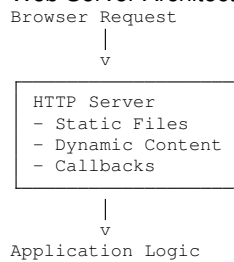
10.3.2.14 Web Server

Focus:** Embedded HTTP server programming Key Topics:**

- Web server architecture
- Static content serving
- Dynamic content generation
- Form processing
- Resource management

When to Read:** When building web-based interfaces

Web Server Architecture:**



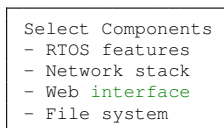
10.3.3 Development Workflow

10.3.3.1 Typical Application Development Process

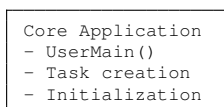
1. Requirements Analysis



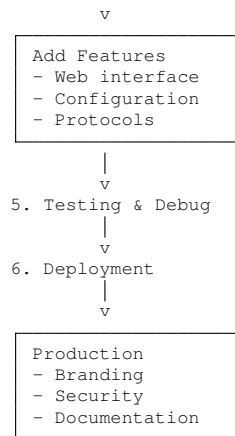
2. Architecture Design



3. Implementation



4. Integration



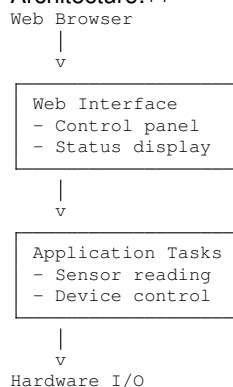
10.3.4 Common Implementation Patterns

10.3.4.1 Pattern 1: Web-Based Device Control

Components Used:**

- Introduction (basics)
- Web Server (interface)
- RTOS (task management)
- Configuration Server (settings)

Architecture:**

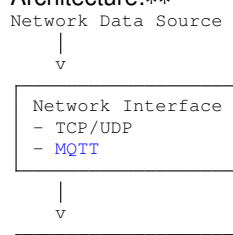


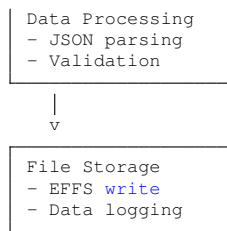
10.3.4.2 Pattern 2: Network Data Logger

Components Used:**

- Network Protocols (communication)
- EDFS (data storage)
- JSON Lexer (data format)
- Configuration (settings)

Architecture:**





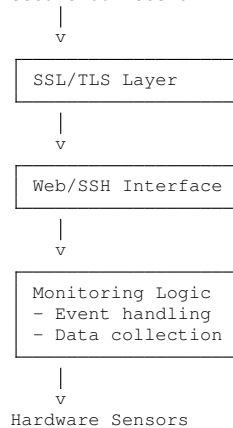
10.3.4.3 Pattern 3: Secure Remote Monitoring

Components Used:**

- SSL/TLS (security)
- SSH (remote access)
- Web Server (interface)
- Interrupt Handling (events)

Architecture:**

Secure Connection



10.3.5 Cross-Reference Guide

10.3.5.1 By Task

10.3.5.1.1 Setting Up Network Communication

1. Introduction - Basic network initialization
2. Network Protocols - Protocol implementation
3. IPv4-IPv6 Dual Stack - IP version support
4. File Descriptors - Socket I/O

10.3.5.1.2 Creating User Interfaces

1. Web Server - HTTP server basics
2. HTML Processing - Dynamic content
3. Configuration Server - Built-in UI
4. Custom Branding - UI customization

10.3.5.1.3 Managing Data

1. JSON Lexer - Data parsing
2. EDFS File System - Storage
3. Configuration Server - Settings persistence
4. File Descriptors - I/O operations

10.3.5.1.4 Implementing Real-Time Features

1. NetBurner RTOS - Task management
2. Interrupt Handling - Hardware events
3. File Descriptors - Device I/O
4. Network Protocols - Communication

10.3.5.1.5 Securing Your Application

1. SSL/TLS Security - Encrypted communication
 2. SSH Protocol - Secure access
 3. Configuration Server - Access control
 4. Custom Branding - Security customization
-

10.3.6 Quick Start Checklist

10.3.6.1 Essential Reading for New Developers

- [] 1. Read Introduction
 - Understand UserMain()
 - Learn `init()` function
 - Study example code
- [] 2. Review RTOS Basics
 - Task creation
 - Basic synchronization
 - System timing
- [] 3. Explore Web Server
 - Static content
 - Dynamic tags
 - Basic callbacks
- [] 4. Study Configuration
 - System settings
 - Application data
 - Web interface
- [] 5. Understand File Descriptors
 - I/O abstraction
 - Network sockets
 - Serial ports

10.3.6.2 Advanced Topics Preparation

- [] Master RTOS features
 - All synchronization types
 - Task priorities
 - System configuration
- [] Network programming
 - TCP/UDP sockets
 - Protocol implementation
 - IPv4/IPv6 support
- [] Security implementation
 - SSL/TLS setup
 - SSH configuration

```

- Certificate management
[ ] Custom development
  - Interrupt handlers
  - Custom drivers
  - Branding options

```

10.3.7 Document Conventions

10.3.7.1 Code Examples

Throughout this guide, code examples use the following conventions:

```

// Full working examples are marked
void ExampleFunction() {
    // Code here...
}
// Tested
// Conceptual examples show patterns
void ConceptualExample() {
    // Pseudocode or simplified logic
}

```

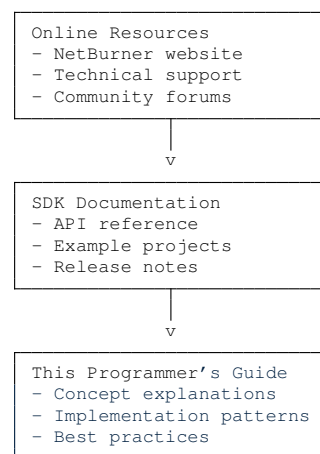
10.3.7.2 Platform References

Platform-specific information is clearly marked:

- **All Platforms** - Universal features
 - **MCF5441x** - ColdFire specific
 - **SAME70** - Cortex-M7 specific
 - **SOMRT1061** - i.MX RT specific
-

10.3.8 Getting Help

10.3.8.1 Documentation Resources



10.3.9 Version Information

This Programmer's Guide is maintained alongside the NetBurner Network Development Kit (NNDK). Always refer to the documentation version matching your NNDK release.

Document Organization:**

- Each major topic has a dedicated section
 - Cross-references guide related topics
 - Examples demonstrate practical implementation
-

10.3.10 Next Steps

After reviewing this main guide:

1. **Begin with Introduction** for fundamental concepts
2. **Choose topic areas** based on your application needs
3. **Study example projects** in the SDK
4. **Implement incrementally** building on working examples
5. **Refer to API documentation** for detailed function references

10.4 Introduction

10.4.1 Intro to NetBurner Programming

10.4.2 Overview

The NetBurner Programming Guide provides a comprehensive overview of the features and capabilities of the NetBurner Network Development Kit (NNDK). This guide focuses on practical implementation of network applications using NetBurner hardware, software, and development tools through a hands-on, example-driven approach.

10.4.3 Prerequisites

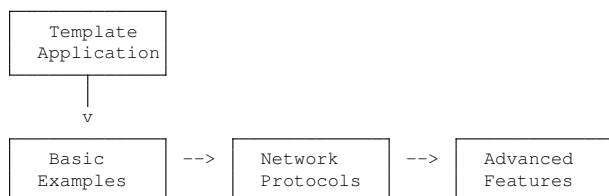
Before beginning this guide, ensure you have:

- **Installed** the NetBurner Network Development Kit
- **Configured** a running NetBurner device on your network
- **Familiarity** with either:
 - NBEclipse development environment (see NBEclipse Getting Started Guide), or
 - Command-line tools and your preferred development environment
- **Knowledge** of your target device's network configuration
- **Successfully** created and deployed a test project to your target device

10.4.4 Learning Approach

This guide employs a **learn-by-example** methodology. Each example builds upon a foundational template application that serves as a starting point for most NetBurner projects. Progressive examples demonstrate increasingly complex networking capabilities.

Development Flow



10.4.5 Source Code Location

All example programs referenced in this guide are located in the following directory:

<NNDK_ROOT>/examples/

Where <NNDK_ROOT> is your NetBurner tools installation directory.

10.4.6 Creating Your First Application

10.4.6.1 Using the NBEclipse Application Wizard

The quickest way to start is with the NBEclipse Application Wizard:

1. Launch NBEclipse and create a new project
2. Select the following wizard options:
 - **Standard Initialization**
 - **Web Server**
3. Complete the wizard to generate your project

If you're unfamiliar with NBEclipse, consult the NBEclipse Getting Started Guide before proceeding.

10.4.6.2 Template Application Structure

The Application Wizard generates a `main.cpp` file with the following structure:

```
#include <predef.h>           // System level options and definitions
#include <init.h>             // Initialization functions
#include <stdio.h>            // Standard I/O functions
#include <nbrtos.h>           // NetBurner Real Time Operating System (RTOS)
#include <http.h>             // HTTP functions

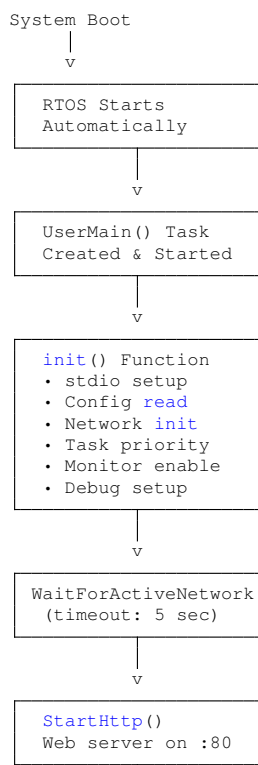
const char * AppName = "AppWizard";    // Application name for discovery

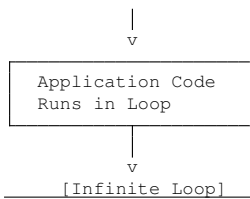
void UserMain(void * pd)
{
    init();                          // Initialize system
    WaitForActiveNetwork(TICKS_PER_SECOND * 5); // Wait up to 5 seconds
    StartHttp();                     // Start web server

    printf("Application %s started\n", AppName);

    while (1)                        // Main application loop
    {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

10.4.6.3 Application Execution Flow





10.4.7 Key System Functions

10.4.7.1 init()

The `init()` function performs critical system initialization:

- **stdio Configuration:** Initializes standard I/O to the debug/console port
- **Configuration Processing:** Reads system settings including:
 - Network interface parameters
 - Serial port configuration
 - Boot options
- **Network Stack:** Initializes the TCP/IP stack
- **Task Priority:** Sets `UserMain()` priority to `MAIN_PRIO`
- **Task Monitor:** Enables task monitoring utilities
- **Debug Support:** Activates GDB debugger in debug builds

10.4.7.2 WaitForActiveNetwork()

Blocks execution until a network link is established or timeout occurs.

Parameters:**

- Timeout value in system ticks
- Use `TICKS_PER_SECOND` constant for time calculations

Example:**

```
WaitForActiveNetwork(TICKS_PER_SECOND * 5); // Wait up to 5 seconds
```

10.4.7.3 StartHttp()

Launches the HTTP web server.

Default Configuration:**

- Port: 80 (standard HTTP)
 - Can specify alternate port: `StartHttp(8080)`
- Secure Alternative:**
- Use `StartHttps()` for SSL/TLS support

10.4.8 Output Functions

10.4.8.1 printf() vs printf()

NetBurner supports both standard output functions:

Function	Description	Use Case
<code>printf()</code>	Full formatting support including floating-point	General purpose output
<code>printf()</code>	Integer-only formatting	Resource-constrained applications

Recommendation:** Use `printf()` when floating-point output is not required to conserve system resources.

Example:**

```
printf("Application %s started\n", AppName);
printf("Network IP: %d.%d.%d.%d\n", ip[0], ip[1], ip[2], ip[3]);
```

10.4.9 Application Design Principles

10.4.9.1 The Main Loop

Your application **must never return** from `UserMain()`. The standard pattern uses an infinite loop:

```
while (1)
{
    OSTimeDly(TICKS_PER_SECOND); // Yield CPU periodically
}
```

10.4.9.2 Timing Considerations

Always use `TICKS_PER_SECOND` for time-based calculations:

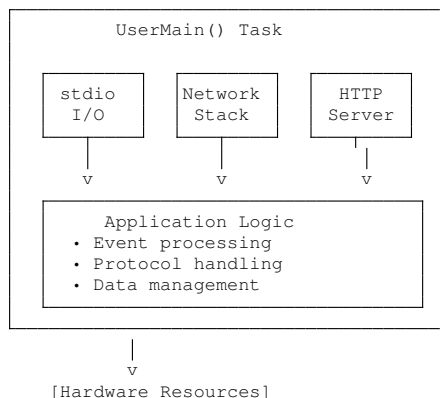
```
// Correct: Portable timing
OSTimeDly(TICKS_PER_SECOND * 2); // Delay 2 seconds

// Avoid: Hard-coded tick values
OSTimeDly(200); // Not portable!
```

Why? The `TICKS_PER_SECOND` constant ensures code portability if system tick rates change across platforms or configurations.

10.4.9.3 Resource Management

Application Resource Flow



10.4.10 Next Steps

With this foundation, you're ready to explore:

1. **Network Communication** - TCP/IP, UDP protocols
2. **Web Server Development** - Dynamic content, forms, APIs
3. **Serial Communications** - UART configuration and protocols
4. **Real-Time Features** - Task management, interrupts
5. **Advanced Topics** - SSL/TLS, [MQTT](#), custom protocols

Each subsequent section builds upon this template application, adding functionality progressively.

10.4.11 Quick Reference

10.4.11.1 Critical Constants

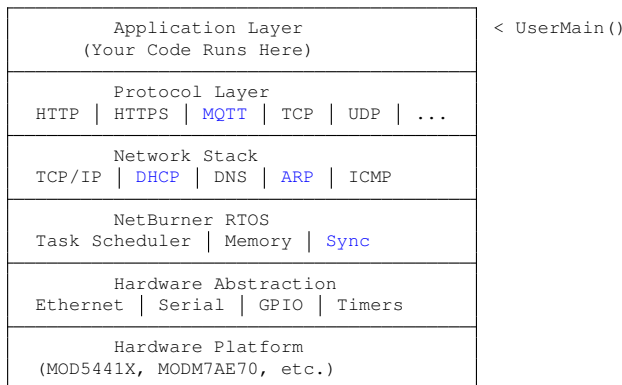
```
TICKS_PER_SECOND // System tick rate
MAIN_PRIO        // UserMain task priority
```

10.4.11.2 Core Functions

```
init() // System initialization
WaitForActiveNetwork(ticks) // Network wait
StartHttp() // HTTP server
StartHttps() // HTTPS server
OSTimeDly(ticks) // Task delay
printf(format, ...) // Stdout printf
```

10.4.12 Architecture Overview

NetBurner Application Stack



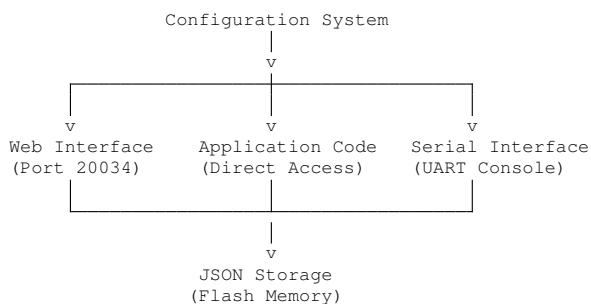
10.5 Config Server Programming Guide

10.5.1 Configuration Server Programming Guide

10.5.2 Introduction

The NetBurner Configuration System provides a comprehensive solution for managing system and application settings stored in flash memory. This system enables configuration through multiple interfaces while maintaining data persistence across power cycles.

10.5.2.1 Configuration System Overview



10.5.2.2 Access Methods

The configuration data can be accessed and modified through multiple interfaces:

1. **System Configuration Web Server** - Port 20034
2. **Application Code** - Direct programmatic access
3. **Custom Web Interface** - Your own web pages
4. **Serial Config Server** - Command-line interface via UART

10.5.2.3 Key Features

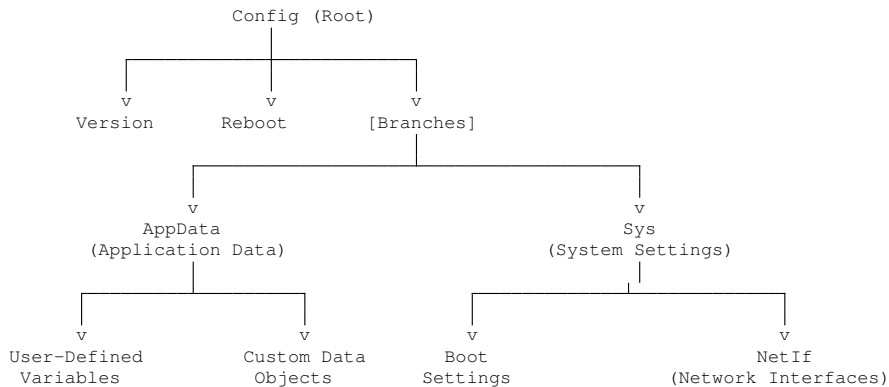
- **Persistent Storage** - All settings saved to flash memory
- **JSON Format** - Configuration stored as JSON blobs
- **Hierarchical Structure** - Tree-based organization with leaves
- **Flexible Access** - Modify entire JSON or individual components

- **Version Control** - Automatic version tracking on updates
- **Application Upload** - Built-in firmware update capability

10.5.3 Configuration Structure

The configuration system uses a hierarchical tree structure with distinct branches for system and application data.

10.5.3.1 Configuration Tree Hierarchy



10.5.3.2 Root Level: Config

The top-level Config object contains:

Field	Type	Description
Version	Integer	Automatically updated when settings saved
Reboot	Boolean	Forces module reboot when configuration updated
AppData	Object	User application data container
Sys	Object	System settings container

Example Structure:**

```

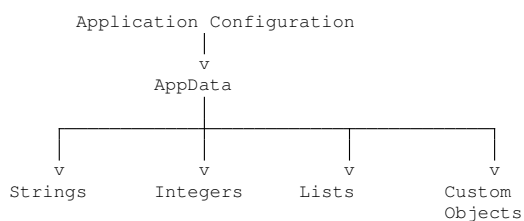
{
  "Config": {
    "Version": 15,
    "Reboot": false,
    "AppData": { ... },
    "Sys": { ... }
  }
}

```

10.5.4 Config.AppData

The AppData leaf is dedicated to user application data, providing a designated space for custom settings and parameters.

10.5.4.1 Purpose and Usage



10.5.4.2 Available Configuration Objects

The system provides various typed configuration objects:

- **config_string** - Text values
- **config_int** - Integer values
- **config_bool** - Boolean values
- **config_chooser** - Selection lists/dropdowns
- **Custom Objects** - User-defined structures

10.5.4.3 Example: Application Settings

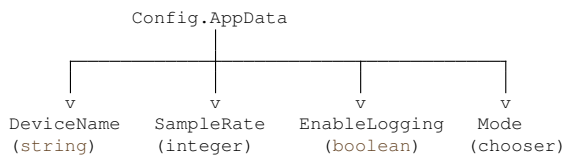
```
// Define application configuration
namespace AppConfig
{
    config_string deviceName("DeviceName", "Device-001",
                             "Display name for this device");

    config_int sampleRate("SampleRate", 1000,
                          "Sensor sampling rate (Hz)");

    config_bool enableLogging("EnableLogging", true,
                              "Enable data logging");

    config_chooser mode("Mode",
                       {"Normal", "Debug", "Production"},
                       0, "Operating mode");
}
```

10.5.4.4 Configuration Object Hierarchy



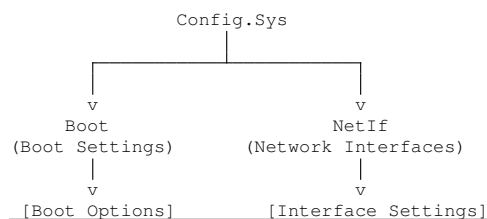
For detailed examples**, see:

- BasicConfigVariable example
- \examples\Configuration\BasicConfigVariable

10.5.5 Config.Sys - System Settings

The Sys leaf contains all device system settings including boot parameters and network configuration.

10.5.5.1 System Configuration Tree



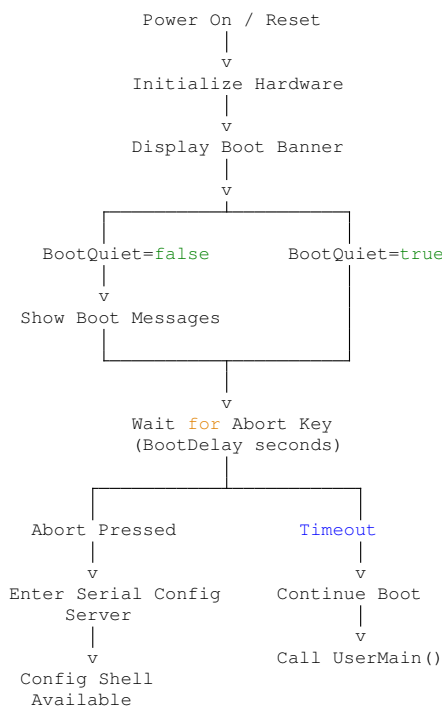
10.5.6 Config.Sys.Boot - Boot Settings

Boot settings control system startup behavior, security, and serial configuration access.

10.5.6.1 Boot Configuration Parameters

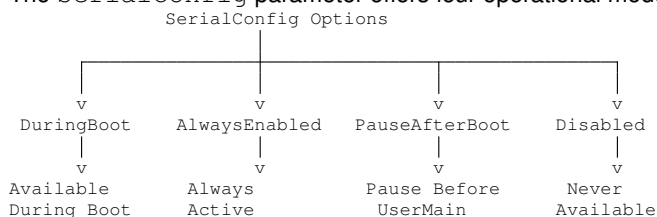
Parameter	Type	Description
Abort	String	Key sequence to interrupt boot (default: "A")
BootBaud	Integer	Baud rate for boot UART
BootDelay	Integer	Wait time for abort sequence (seconds)
BootQuiet	Boolean	Suppress boot output messages
BootUart	Integer	UART number for boot output
Password	String	Configuration/upload password
User	String	Configuration/upload username
SerialConfig	Enum	Serial config server mode

10.5.6.2 Boot Process Flow



10.5.6.3 Serial Config Server Modes

The `SerialConfig` parameter offers four operational modes:



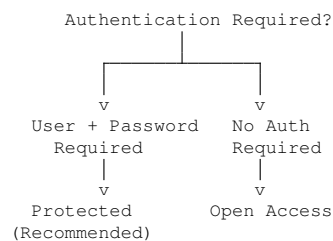
10.5.6.3.1 Mode Descriptions

DuringBoot (Recommended)**

- Serial config available only during boot
 - Use abort character sequence to enter
 - Safe for production use
- AlwaysEnabled (Use with Caution)**
- Serial config always accessible via BootUart

- May interfere with application serial I/O
- Risk of accidental config corruption
PauseAfterBoot**
- Boot pauses after `init()` called
- Serial config enabled automatically
- Type `boot` to resume application
Disabled**
- Serial config completely disabled
- No boot interruption possible
- Maximum security, no serial access

10.5.6.4 Security Configuration



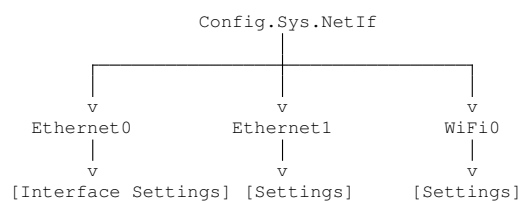
When both User and Password are set:

- Required for configuration changes
- Required for application uploads
- Protects against unauthorized access

10.5.7 Config.Sys.NetIf - Network Interfaces

The `NetIf` section contains settings for all network interfaces including Ethernet and WiFi.

10.5.7.1 Network Interface Tree

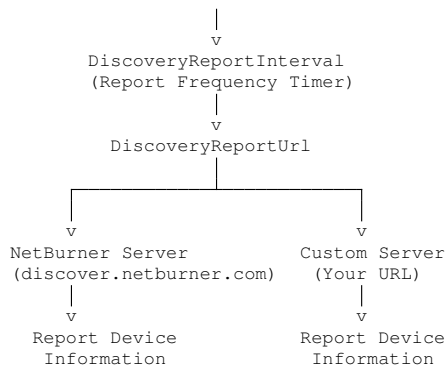


10.5.7.2 Common Interface Parameters

Parameter	Description
DeviceName	Interface name for DDNS/NetBIOS registration
DhcpDiscoverSec	Delay before DHCP Discover (seconds)
DiscoveryReportInterval	Device discovery report frequency
DiscoveryReportUrl	Discovery server URL
MAC	Interface MAC address

10.5.7.3 Discovery Configuration

Device Discovery



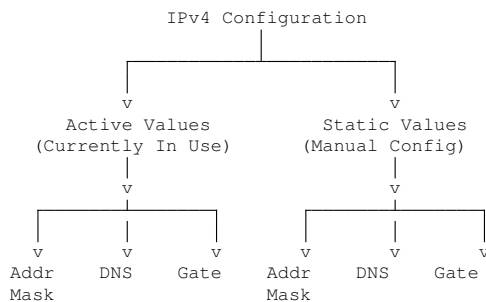
Discovery Options:**

- **Default:** Reports to NetBurner's discovery server
- **Custom:** Set your own discovery server URL
- **Disabled:** Leave URL blank to disable reporting

10.5.8 Config.Sys.NetIf.Ethernet0.IPv4

IPv4 configuration settings for Ethernet interfaces.

10.5.8.1 IPv4 Settings Overview



10.5.8.2 IPv4 Parameters

10.5.8.2.1 Active Settings (Read-Only)

Parameter	Description
ActiveAddr	Current IPv4 address in use
ActiveDNS1	Current primary DNS server
ActiveDNS2	Current secondary DNS server
ActiveGate	Current gateway address
ActiveMask	Current subnet mask
AutoIPAddr	Auto-assigned IP (if applicable)
AutoIPEn	Auto-IP enabled status

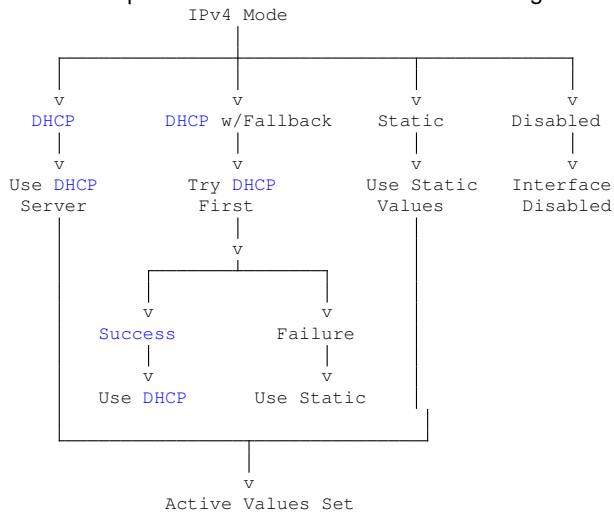
10.5.8.2.2 Static Settings (User Configurable)

Parameter	Description
StaticAddr	Manually configured IP address
StaticDNS1	Manually configured primary DNS
StaticDNS2	Manually configured secondary DNS
StaticGate	Manually configured gateway

StaticMask	Manually configured subnet mask

10.5.8.3 IPv4 Mode Selection

The Mode parameter determines how IPv4 settings are obtained:



10.5.8.3.1 Mode Descriptions

DHCP**

- Obtain all settings from **DHCP** server
- If **DHCP** fails, no network connectivity
- **DHCP with Fallback****
- Try **DHCP** first
- Fall back to static values if **DHCP** unavailable
- Recommended for most applications

Static**

- Use manually configured static values
- No **DHCP** attempts

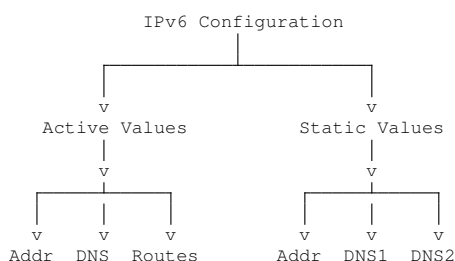
Disabled**

- Interface completely disabled
- No network activity

10.5.9 Config.Sys.NetIf.Ethernet0.IPv6

IPv6 configuration settings for Ethernet interfaces.

10.5.9.1 IPv6 Settings Overview



10.5.9.2 IPv6 Parameters

10.5.9.2.1 Active Settings (Read-Only)

Parameter	Description
ActiveAddr	Array of active IPv6 addresses
ActiveDNS	Array of active IPv6 DNS servers
ActiveRoutes	Array of active IPv6 routes

10.5.9.2.2 Static Settings (User Configurable)

Parameter	Description
StaticAddr	Manually configured IPv6 address
StaticDNS1	Manually configured primary IPv6 DNS
StaticDNS2	Manually configured secondary IPv6 DNS

10.5.9.3 IPv6 Mode Selection

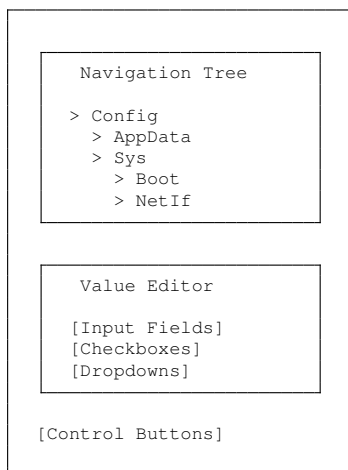
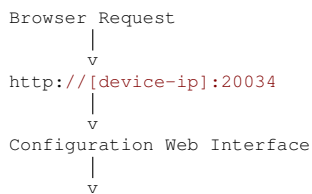
Similar to IPv4, the Mode parameter offers:

- **DHCP** - DHCPv6/SLAAC automatic configuration
- **DHCP w/Fallback** - Try auto-config, fall back to static
- **Static** - Use manually configured values
- **Disabled** - Disable IPv6 on interface

10.5.10 Configuration Web Interface

The configuration web server runs on **port 20034** and provides a full-featured interface for viewing and modifying settings.

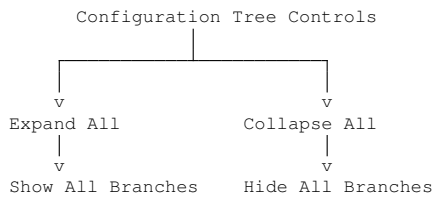
10.5.10.1 Accessing the Web Interface



Example URL**: <http://10.1.1.100:20034>

10.5.10.2 Web Interface Features

10.5.10.2.1 Navigation Controls

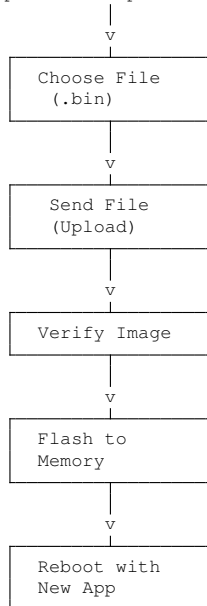


10.5.10.2.2 Action Buttons

Button	Function	Action
Expand All	Expand entire tree	Shows all configuration branches
Collapse All	Collapse entire tree	Hides all sub-branches
Update Record	Save changes	Writes current values to flash
Download Config	Export configuration	Downloads JSON file
Restore Config	Import configuration	Uploads JSON file
Raw Config Page	View raw interface	Shows unformatted JSON editor
NB Discover	NetBurner discovery	Opens discover.netburner.com
Discover Local	Local network scan	Finds devices without internet

10.5.10.3 Application Update Feature

Application Update Process



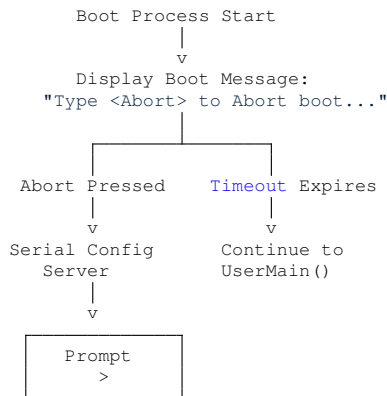
Update Steps:**

1. Click "Choose File" button
2. Select compiled .bin file
3. Click "Send File" to upload
4. Wait for verification and programming
5. Device automatically reboots with new application

10.5.11 Serial Configuration Interface

The serial configuration interface provides command-line access to the configuration system via UART.

10.5.11.1 Accessing Serial Config



10.5.11.2 Serial Config Prompt

When successfully entered, you'll see:

```
>
```

As you navigate, the prompt shows your location:

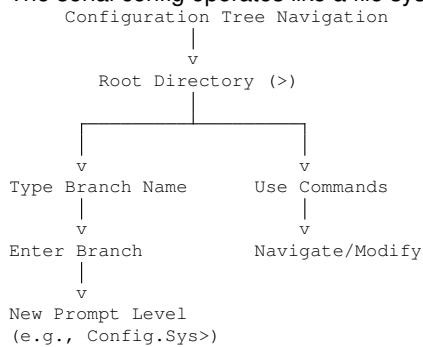
```

>                                     # Root level
Config.Sys>                         # In Sys branch
Config.Sys.Boot>                   # In Boot settings

```

10.5.11.3 Navigation Model

The serial config operates like a file system:



10.5.11.4 Serial Config Commands

Command	Syntax	Description
help	help	Display all available commands
boot	boot	Exit config server and boot application
reboot	reboot	Restart the module
flash	flash	Upload new application via serial
ls	ls	List current level contents
..	..	Move up one level in tree
<var>?	Config?	Display variable contents
<var>=<val>	Boot.Abort="B"	Set variable value
save	save	Save changes to flash

10.5.11.5 Command Examples

10.5.11.5.1 Viewing Configuration

```

> Config?
{
  "Version": 15,
  "Reboot": false,

```

```

    "AppData": {...},
    "Sys": {...}
}

> Config.Sys.Boot?
{
  "Abort": "A",
  "BootBaud": 115200,
  "BootDelay": 3,
  ...
}

```

10.5.11.5.2 Modifying Settings

```

> Config.Sys.Boot
Config.Sys.Boot> Abort="B"
Config.Sys.Boot> BootDelay=5
Config.Sys.Boot> save
Configuration saved.
Config.Sys.Boot> ..
Config.Sys> boot
Continuing boot...

```

10.5.11.5.3 Navigation Example

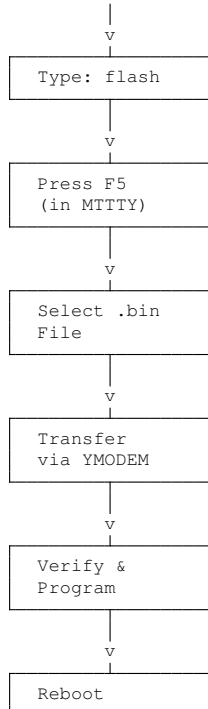
```

> Config
Config> Sys
Config.Sys> NetIf
Config.Sys.NetIf> Ethernet0
Config.Sys.NetIf.Ethernet0> IPv4
Config.Sys.NetIf.Ethernet0.IPv4> ls
ActiveAddr
ActiveDNS1
ActiveDNS2
Mode
StaticAddr
...
Config.Sys.NetIf.Ethernet0.IPv4> ..
Config.Sys.NetIf.Ethernet0> ..
Config.Sys.NetIf> ..
Config.Sys> ..
Config> ..
>

```

10.5.11.6 Uploading Application via Serial

Serial Flash Process



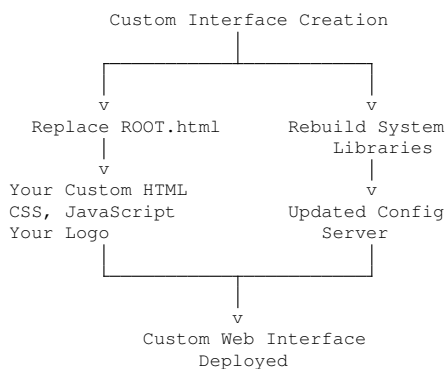
Steps:**

1. Enter serial config mode
2. Type `flash` and press Enter
3. In MTTTY: Press F5 to open file dialog
4. Select your `.bin` file
5. Wait for transfer and programming
6. Device reboots automatically

10.5.12 Creating a Custom Web Interface

The configuration web interface can be completely customized with your own design, branding, and functionality.

10.5.12.1 Customization Workflow



10.5.12.2 Steps to Customize

10.5.12.2.1 Option 1: Replace System Interface

1. **Locate system file:**
`<NNDK Install Directory>\nbrtos\source\ROOT.html`
2. **Create your custom ROOT.html**
3. **Rebuild system libraries:**
 - Follow NBEclipse Getting Started Guide
 - Rebuild system libraries section

10.5.12.2.2 Option 2: Replace Logo Only

1. **Locate logo file:**
`<NNDK Install Directory>\nbrtos\source\LOGO.gif`
2. **Replace with your logo image**
3. **Rebuild system libraries**

10.5.12.3 Custom Interface Example

A complete example demonstrating the current config web interface structure is available:

Location:**

`<NNDK Install Directory>\examples\Configuration\BasicWebConfig`

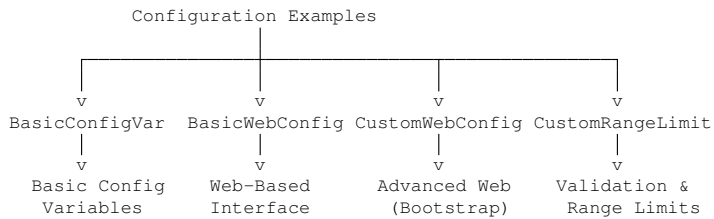
This example shows:

- HTML structure
- JavaScript interaction
- CSS styling
- Configuration API usage

10.5.13 Configuration API Examples

NetBurner provides comprehensive examples demonstrating various aspects of the configuration system.

10.5.13.1 Example Projects Overview



10.5.13.2 Available Examples

10.5.13.2.1 1. BasicConfigVariable

Location:** <NNDK>\examples\Configuration\BasicConfigVariable
Demonstrates:**

- Basic config object usage
- Creating config variables
- Reading and writing values
- Saving to flash

Key Concepts:**

```

// String configuration
config_string deviceName("Name", "Device-001", "Device identifier");

// Integer configuration
config_int sampleRate("Rate", 1000, "Sample rate in Hz");

// Boolean configuration
config_bool enabled("Enabled", true, "Feature enabled");

// Reading values
const char* name = deviceName.c_str();
int rate = sampleRate;
bool isEnabled = enabled;

// Writing values
deviceName = "Device-002";
sampleRate = 2000;
enabled = false;

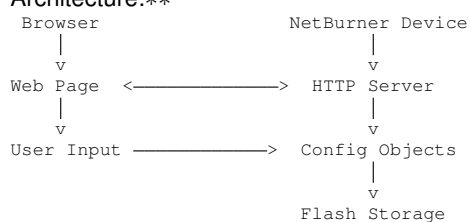
// Save to flash
SaveConfigToStorage();
  
```

10.5.13.2.2 2. BasicWebConfig

Location:** <NNDK>\examples\Configuration\BasicWebConfig
Demonstrates:**

- Config system web interface basics
- Dynamic web page generation
- Real-time value updates
- Form handling

Architecture:**



10.5.13.2.3 3. CustomWebConfig

Location:** <NNDK>\examples\Configuration\CustomWebConfig

Demonstrates:**

- Advanced web interface design
- Bootstrap framework integration
- jQuery for dynamic updates
- Professional UI/UX

Features:**

- Responsive design
- Interactive controls
- Real-time validation
- Modern web technologies

10.5.13.2.4 4. CustomRangeLimit

Location:** <NNDK>\examples\Configuration\CustomRangeLimit

Demonstrates:**

- Extending config objects
- Custom validation logic
- Min/max range checking
- Input constraints

Example:**

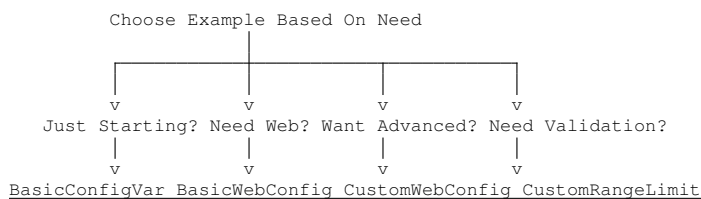
```
// Custom range-limited integer
class config_int_limit : public config_int
{
public:
    config_int_limit(const char* name, int defaultVal,
                    int minVal, int maxVal, const char* desc)
        : config_int(name, defaultVal, desc),
          min(minVal), max(maxVal) {}

    bool SetValue(int val) override
    {
        if (val < min || val > max)
            return false; // Reject out-of-range
        return config_int::SetValue(val);
    }

private:
    int min, max;
};

// Usage
config_int_limit temperature("Temp", 25, 0, 100,
                             "Temperature (0-100C)");
```

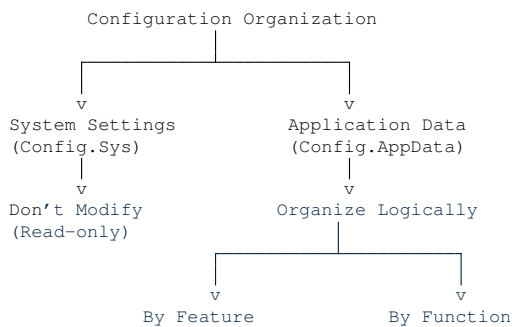
10.5.13.3 Example Selection Guide



10.5.14 Best Practices

10.5.14.1 Configuration Design Principles

10.5.14.1.1 1. Organization Strategy



Guidelines:**

- Keep application data in `Config.AppData`
- Don't modify `Config.Sys` unless necessary
- Group related settings together
- Use descriptive names
- Add helpful descriptions

10.5.14.1.2 2. Variable Naming

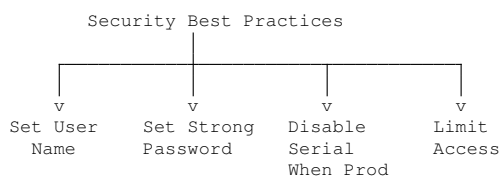
Good Variable Names:

<code>deviceName</code>	Clear purpose
<code>sampleRateHz</code>	Includes units
<code>enableDataLogging</code>	Boolean prefix
<code>motorSpeedRPM</code>	Descriptive + units

Poor Variable Names:

<code>var1</code>	Meaningless
<code>x</code>	Too short
<code>temp</code>	Ambiguous
<code>d</code>	No context

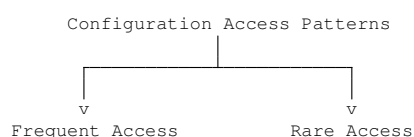
10.5.14.1.3 3. Security Considerations

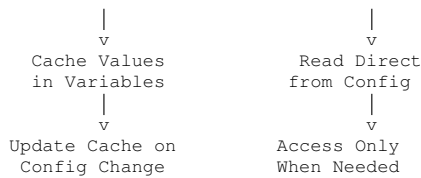


Recommendations:**

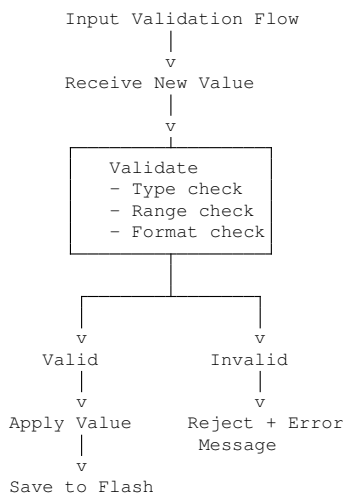
- Always set User and Password for production
- Use strong passwords (8+ characters, mixed case, numbers)
- Disable serial config in deployed systems
- Limit web interface access by network
- Consider custom web interface with authentication

10.5.14.1.4 4. Performance Optimization



**Tips:****

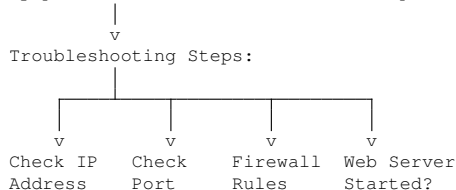
- Cache frequently accessed values
- Update cache when config changes
- Minimize flash writes
- Batch configuration updates

10.5.14.1.5 5. Validation and Error Handling**Implementation:****

- Always validate user input
- Use custom config classes for complex validation
- Provide clear error messages
- Log configuration changes

10.5.15 Troubleshooting**10.5.15.1 Common Issues and Solutions****10.5.15.1.1 Issue: Cannot Access Web Interface**

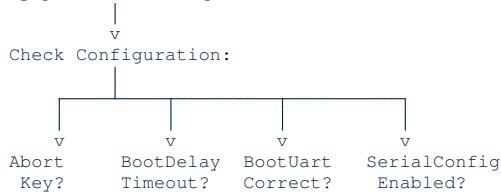
Symptom: Browser cannot connect to port 20034

**Solutions:****

1. Verify device IP address is correct
2. Ensure using port 20034: <http://ip:20034>
3. Check firewall allows port 20034
4. Verify config web server is enabled in code

10.5.15.1.2 Issue: Serial Config Not Accessible

Symptom: Abort sequence doesn't work

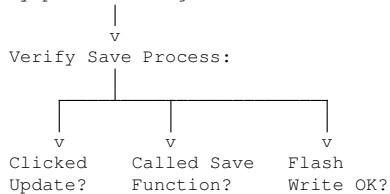


Solutions:**

1. Verify Abort key is correct (default: "A")
2. Send abort key during BootDelay window
3. Ensure using correct UART port
4. Check SerialConfig not set to "Disabled"

10.5.15.1.3 Issue: Configuration Changes Not Saved

Symptom: Settings revert after reboot

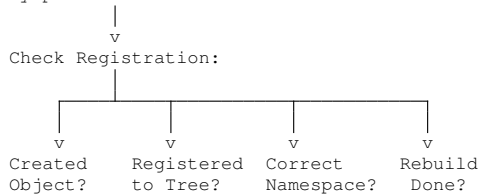


Solutions:**

1. Web Interface: Click "Update Record" button
2. Serial: Execute save command
3. Code: Call `SaveConfigToStorage()`
4. Check flash memory not full/corrupted

10.5.15.1.4 Issue: Custom Config Variables Not Appearing

Symptom: Variables not visible in interface

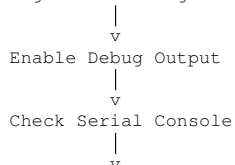


Solutions:**

1. Ensure config object created globally
2. Check object registered to Config.AppData
3. Verify namespace/path is correct
4. Rebuild project completely

10.5.15.2 Debug Checklist

Configuration Debug Process



```

Verify Network Settings
    |
    v
Test with Examples
    |
    v
Review System Logs
    |
    v
Check Flash Memory

```

Debug Steps:**

1. Enable serial debug output with `printf()`
2. Monitor boot messages on UART
3. Verify network connectivity
4. Test with provided examples first
5. Check system logs for errors
6. Verify flash memory health

10.5.16 Advanced Topics

10.5.16.1 Custom Configuration Objects

You can create custom configuration types by extending base classes:

```

// Custom enum configuration
class config_enum : public config_chooser
{
public:
    enum MyEnum { OPTION_A, OPTION_B, OPTION_C };

    config_enum(const char* name, MyEnum defaultVal, const char* desc)
        : config_chooser(name, {"A", "B", "C"},
                          (int)defaultVal, desc) {}

    MyEnum GetValue() const {
        return (MyEnum)GetSelection();
    }
};

```

10.5.16.2 Configuration Callbacks

Implement change notifications:

```

class config_int_notify : public config_int
{
public:
    using NotifyCallback = void (*)(int oldVal, int newVal);

    config_int_notify(const char* name, int defaultVal,
                     const char* desc, NotifyCallback cb)
        : config_int(name, defaultVal, desc), callback(cb) {}

    bool SetValue(int val) override {
        int oldVal = GetValue();
        if (config_int::SetValue(val)) {
            if (callback) callback(oldVal, val);
            return true;
        }
        return false;
    }

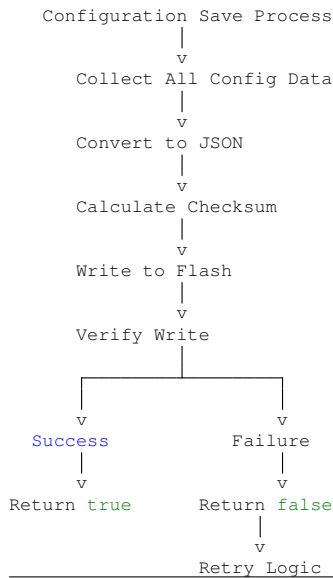
private:
    NotifyCallback callback;
};

// Usage
void onSpeedChanged(int oldVal, int newVal) {
    printf("Speed changed from %d to %d\n", oldVal, newVal);
    UpdateMotorSpeed(newVal);
}

config_int_notify motorSpeed("Speed", 1000, "Motor speed",
                             onSpeedChanged);

```

10.5.16.3 Configuration Persistence



10.5.17 API Quick Reference

10.5.17.1 Core Configuration Functions

```

// Initialize configuration system
void InitConfigSystem();

// Save configuration to flash
bool SaveConfigToStorage();

// Load configuration from flash
bool LoadConfigFromStorage();

// Reset to factory defaults
void ResetConfigToDefaults();

// Get configuration as JSON
std::string GetConfigJSON();

// Set configuration from JSON
bool SetConfigJSON(const char* json);

```

10.5.17.2 Configuration Object Types

```

// String configuration
config_string myString("name", "default", "description");

// Integer configuration
config_int myInt("name", 0, "description");

// Boolean configuration
config_bool myBool("name", false, "description");

// Chooser/dropdown configuration
config_chooser myChoice("name", {"opt1", "opt2"}, 0, "description");

```

10.5.17.3 Accessing Values

```

// Reading
const char* str = myString.c_str();
int value = myInt;
bool flag = myBool;
int selection = myChoice.GetSelection();

// Writing
myString = "new value";
myInt = 42;
myBool = true;
myChoice.SetSelection(1);

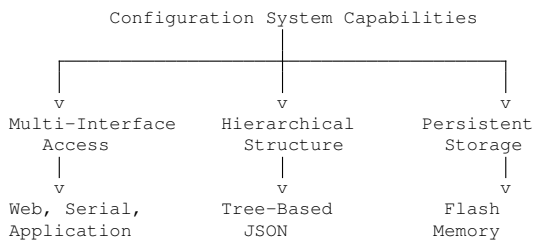
// Save to flash
SaveConfigToStorage();

```

10.5.18 Summary

The NetBurner Configuration System provides a powerful, flexible solution for managing device settings:

10.5.18.1 Key Features Recap



10.5.18.2 Access Methods Summary

Interface	Port/Method	Use Case
Web UI	Port 20034	User-friendly configuration
Serial Config	Boot UART	Command-line access
Application API	Direct code	Programmatic control
Custom Web	Your design	Branded interface

10.5.18.3 Configuration Hierarchy

```

Config (Root)
├── Version (auto-updated)
├── Reboot (force reboot flag)
├── AppData (your application data)
│   ├── Custom variables
│   ├── Settings
│   └── Parameters
├── Sys (system settings)
│   ├── Boot (boot configuration)
│   │   ├── Abort key
│   │   ├── Serial config
│   │   └── Security
│   └── NetIf (network interfaces)
│       ├── Ethernet0
│       │   ├── IPv4
│       │   └── IPv6
│       └── [Other interfaces]
  
```

10.5.18.4 Best Practices Checklist

- Organize application data in Config.AppData
- Use descriptive variable names with units
- Set User and Password for production
- Validate all user input
- Cache frequently accessed values
- Provide helpful descriptions
- Test with provided examples
- Implement error handling
- Document custom configurations
- Use version control for config schemas

10.5.18.5 Getting Started

1. Learn the Basics

- Review `BasicConfigVariable` example
- Understand config object types
- Practice reading/writing values

2. Web Integration

- Study `BasicWebConfig` example
- Create simple web interface
- Test configuration updates

3. Advanced Features

- Explore `CustomWebConfig` for UI
- Implement validation with `CustomRangeLimit`
- Create custom config objects

4. Production Deployment

- Set security credentials
- Disable unnecessary interfaces
- Implement change logging
- Test recovery procedures

The NetBurner Configuration System simplifies device management while providing the flexibility needed for complex embedded applications.

10.6 Custom Branding

10.6.1 Custom Branding

10.6.2 Overview

NetBurner is an OEM (Original Equipment Manufacturer) that enables complete customization of our products to align with your brand identity. Whether you need simple logo changes or comprehensive white-label solutions, NetBurner provides the flexibility to present your brand to your customers seamlessly.

10.6.3 Branding Capabilities Options

Custom branding options include:

- Configuration Server web page logo
- Custom web page user interface
- Physical label customization using NetBurner label template
- NetBurner application utilities branding
- Private cloud server to replace `discover.netburner.com`

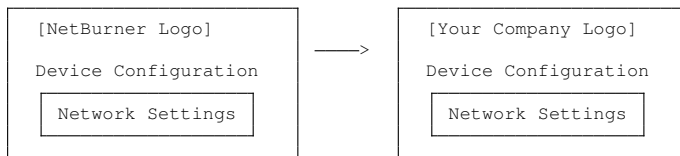
10.6.4 Customization Options

10.6.4.1 1. Configuration Server Logo Replacement

Replace the NetBurner logo on configuration screens with your company branding.

Default Configuration Page

Custom Branded Page



Implementation:**

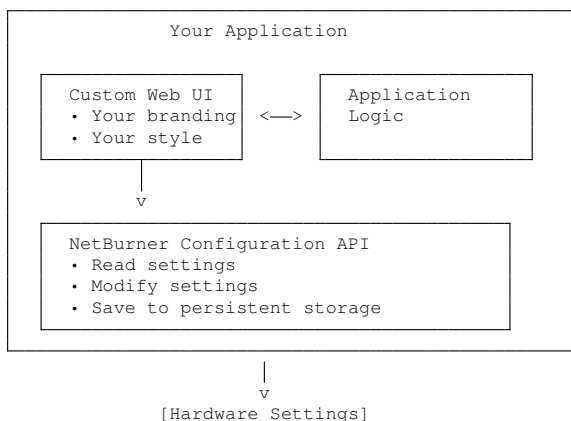
- Reference the CustomWebConfigSystem example
- Replace logo image file
- Update HTML references
- Rebuild and deploy

Location:** See example in NNDK examples directory

10.6.4.2 2. Custom Configuration Interface

Create a fully branded configuration interface integrated into your application.

Application Architecture



Key Points:**

- The default configuration server is **optional**
- Your application has **full access** to all configuration settings
- Settings can be modified via:
 - Custom web interface
 - Application code directly
 - REST APIs
 - Command protocols

Example Location:**

<NNDK_ROOT>/examples/web/

Typically /nburn/examples/web/

Capabilities:**

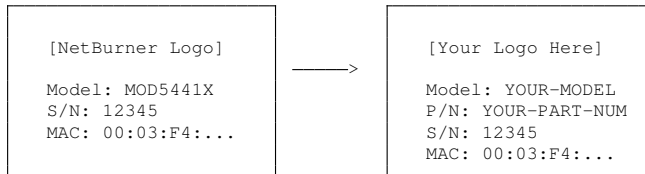
- Display current settings
- Modify any configuration parameter
- Save changes to persistent storage
- Apply changes at runtime

10.6.4.3 3. Physical Device Labeling

Order NetBurner hardware without NetBurner branding.

Standard Label

Custom Label



Options:**

1. **No Label:** Order devices without any NetBurner label
2. **Custom Label:** Use NetBurner-provided templates to create labels with:
 - Your company logo
 - Your part numbers
 - Your branding elements
 - Required regulatory markings

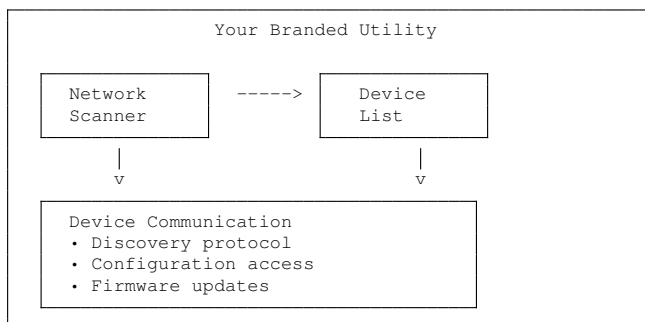
Process:**

1. Contact NetBurner sales for label templates
2. Design your custom label
3. Submit for approval
4. Order devices with custom labels

10.6.4.4 4. Discovery Utility Customization

Rebrand NetBurner PC utilities to match your company identity.

Discovery Tool Workflow



Available Tools:**

Utility	Function	Customization Level
localdiscover	Find devices on local network	Full source available
find	Device discovery tool	Full source available
Additional tools	Various utilities	Source provided

Source Location:**

<NNDK_ROOT>/pctools/

Customization Capabilities:**

- Change application name and branding
- Modify user interface

- Add custom functionality
- Integrate with your systems
- Deploy with your software packages

Build Process:**

1. Locate source in `pctools` directory
2. Modify branding elements
3. Update strings and resources
4. Compile for target platform
5. Distribute with your product

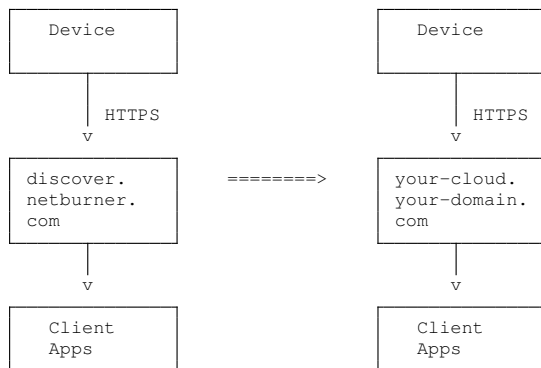
10.6.4.5 5. Custom Cloud Server Infrastructure

Difficulty:** Expert

Replace `discover.netburner.com` with your own cloud infrastructure.

Default Architecture

Custom Architecture



Requirements:**

Infrastructure:**

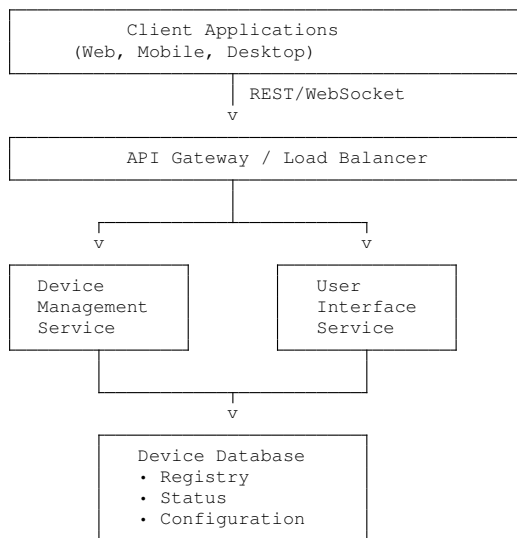
- Cloud hosting platform (AWS, Azure, GCP, etc.)
- SSL/TLS certificates
- Domain name and DNS configuration
- Load balancing (for production)
- Database for device registry
- API endpoints for device communication

Protocol Implementation:**

- Device discovery protocol
- Secure authentication
- Registration and management
- Status monitoring
- Firmware update distribution
- Client application APIs

Architecture Components:**

Cloud Server Stack



Security Considerations:**

- Device authentication tokens
- Encrypted communications
- Access control and permissions
- Audit logging
- Compliance requirements

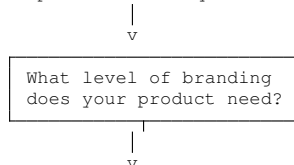
Warning:** This is an advanced customization requiring:

- Strong backend development skills
- Cloud infrastructure expertise
- Security best practices knowledge
- DevOps capabilities
- Ongoing maintenance commitment

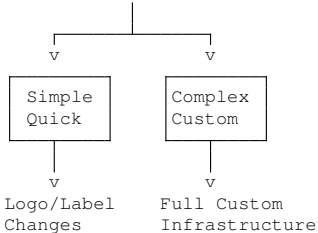
10.6.5 Implementation Strategy

10.6.5.1 Planning Your Customization

Step 1: Assess Requirements



Step 2: Choose Approach



10.6.5.2 Recommended Path

Phase 1 - Quick Wins (Week 1)**

1. Change configuration server logo
2. Order devices with custom labels
3. Update application name and strings

Phase 2 - Custom Interface (Week 2-3)**

1. Design custom web UI
2. Implement configuration interface
3. Add your branding elements
4. Test thoroughly

Phase 3 - Utilities (Week 4)**

1. Customize discovery tools
2. Update PC utilities
3. Rebrand documentation
4. Package for distribution

Phase 4 - Advanced (Month 2+)**

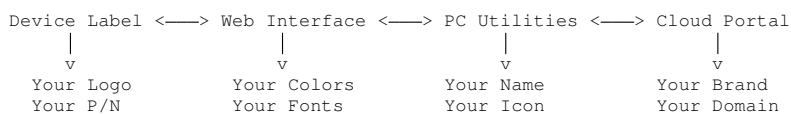
1. Plan cloud infrastructure (if needed)
2. Implement custom services
3. Migrate devices
4. Monitor and maintain

10.6.6 Best Practices

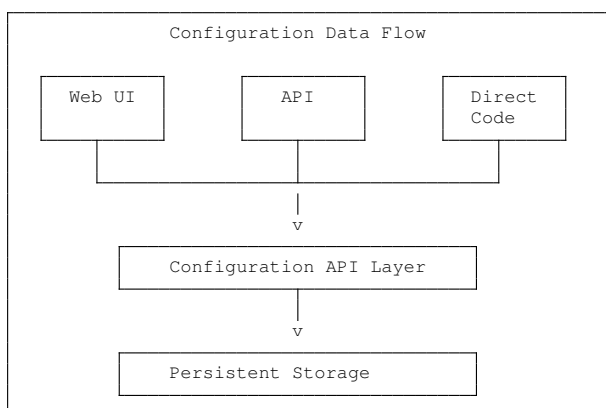
10.6.6.1 Visual Consistency

Ensure brand consistency across all touchpoints:

Brand Touchpoints



10.6.6.2 Configuration Access Patterns



10.6.6.3 Testing Checklist

- ☐ Logo displays correctly on all screens
 - ☐ All text reflects your branding
 - ☐ Custom utilities function properly
 - ☐ Configuration changes persist across reboots
 - ☐ Discovery tools find your devices
 - ☐ Documentation references your brand
 - ☐ Physical labels are accurate
 - ☐ Cloud services (if applicable) are operational
-

10.6.7 Resources and Support

10.6.7.1 Example Code Locations

Customization Type	Location	Description
Configuration Logo	CustomWebConfigSystem	Logo replacement example
Custom Web Interface	<NNDK_ROOT>/examples/web/	Web-based configuration
PC Utilities Source	<NNDK_ROOT>/pctools/	Discovery and management tools

10.6.7.2 Getting Started

For Simple Customizations:**

1. Review the CustomWebConfigSystem example
2. Prepare your logo files (PNG, SVG formats)
3. Follow example documentation
4. Test on development device

For Complex Customizations:**

1. Contact NetBurner technical support
2. Discuss your requirements
3. Review available options
4. Plan implementation timeline

10.6.7.3 Support Channels

- **Technical Documentation:** NetBurner NNDK documentation
 - **Example Code:** Examples directory in NNDK
 - **Technical Support:** NetBurner support team
 - **Community Forums:** NetBurner developer community
-

10.6.8 Security Considerations

When implementing custom branding, maintain security:

Security Layers

Your Branded Interface	< Authentication
Configuration Access	< Authorization
Device Communication	< Encryption
Cloud Services (if any)	< Secure Channels

Key Principles:**

- Maintain authentication mechanisms
- Preserve encryption protocols
- Don't expose sensitive configuration
- Validate all user inputs
- Log security-relevant events
- Keep firmware update security intact

Quick Decision Matrix:**

Your Needs	Recommended Approach
Simple logo change	> Configuration logo replacement
Custom look and feel	> Custom web interface
Complete white label	> All customization options
Enterprise deployment	> Custom cloud infrastructure
Time-to-market priority	> Start with logo, iterate
Full control requirement	> Complete customization path

Start with simple customizations and expand as your product evolves and requirements grow.

10.7 IPv4/IPv6 Dual Stack Guide

10.7.1 Overview

NetBurner release 2.8.0 introduced a fundamental change to IP address handling. The `IPADDR` type evolved from a simple 32-bit integer to a C++ object capable of representing both IPv4 and IPv6 addresses. This guide covers the migration path and best practices for dual-stack networking.

10.7.1.1 Key Changes in 2.8.0+

Before (IPv4 only):**

`IPADDR` = 32-bit unsigned integer

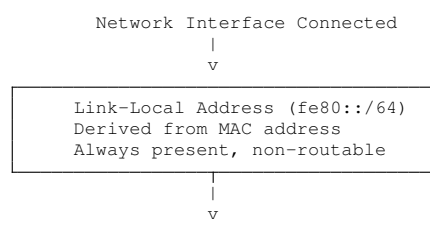
After (Dual Stack):**

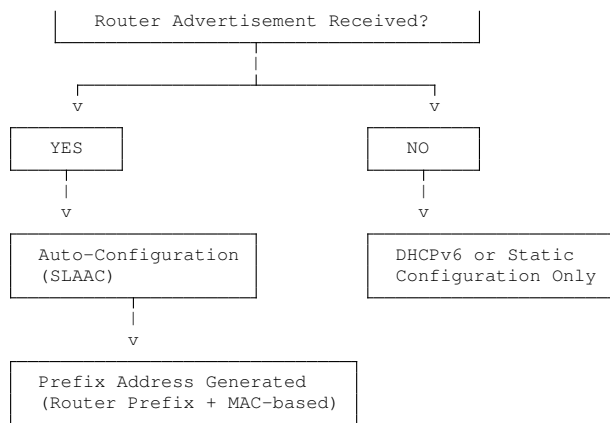
`IPADDR` = C++ object (32-bit for IPv4, 128-bit for IPv6)

10.7.2 IPv6 Address Assignment

Unlike IPv4's typical single-address-per-interface model, IPv6 interfaces inherently support multiple addresses simultaneously. Understanding how these addresses are assigned is crucial for dual-stack development.

10.7.2.1 Address Assignment Flow





10.7.2.2 Link-Local Addresses

Every IPv6 interface automatically generates a link-local address (fe80::/64 prefix) derived from its MAC address.

Address Generation:**

```

MAC Address:      00:03:f4:01:23:47
                   |
                   v
Link-Local:       fe80::203:f4ff:fe01:2347
  
```

Characteristics:**

- Non-routable (local network segment only)
- Used for network negotiation and neighbor discovery
- Always present on IPv6-enabled interfaces
- Deterministic (same for a given MAC address)

10.7.2.3 Router Auto-Configuration (SLAAC)

Routers advertise network prefixes that devices use to auto-configure global IPv6 addresses.

Address Construction:**

IPv6 Address (128 bits)	
Router Prefix (64 bits) From Router Advertisement	Host Portion (64 bits) From MAC (EUI-64)

Example:

```

Router Prefix:    2001:db8:1:2::
MAC-based Suffix: 203:f4ff:fe01:2347
                   |
                   v
Full Address:     2001:db8:1:2:203:f4ff:fe01:2347
  
```

Multiple Router Scenario:**

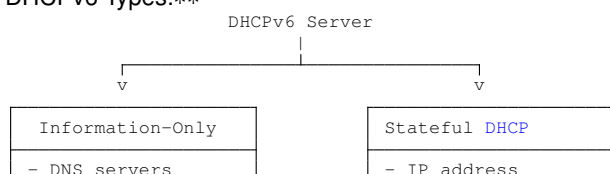
```

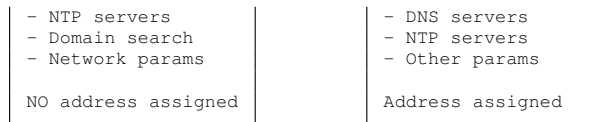
Device Interface
|
|-- Router A Prefix > Address A
|-- Router B Prefix > Address B
|-- Router C Prefix > Address C
+-- Link-Local      > fe80::...
  
```

10.7.2.4 DHCPv6 Configuration

DHCPv6 servers provide address assignment and network configuration, similar to DHCPv4 but with enhanced capabilities.

DHCPv6 Types:**





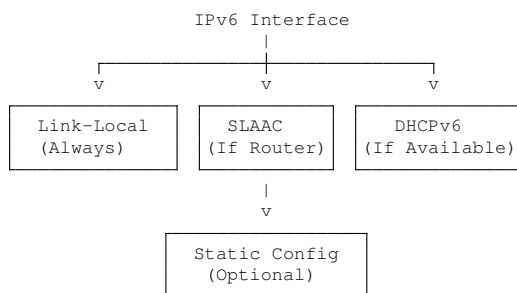
Automatic DHCPv6 Activation:**

- Router advertisements can signal DHCPv6 server presence
- NetBurner automatically initiates DHCPv6 client when signaled
- Manual DHCPv6 client start available for networks without routers

10.7.2.5 Static IP Addresses

Manual address assignment remains available when automatic methods are insufficient or undesired.

10.7.2.6 Complete Address Assignment Hierarchy



10.7.3 IPADDR Class Migration

10.7.3.1 Storage Considerations

Critical Change:** When compiled for IPv6, IPADDR grows from 32 bits to 128 bits.



Impact on Persistent Storage:**

Existing applications storing IPADDR in:

- UserParams structures
- File systems
- Flash memory
- Configuration files

Will encounter deserialization issues when upgrading to IPv6-enabled firmware.

Migration Strategy - Storage Key Pattern:**

```

struct Config {
    uint32_t storageKey;    // Version identifier
    IPADDR serverAddress;
    // ... other fields
};

const uint32_t CURRENT_KEY = 2; // Increment on structure changes

void LoadConfig(Config& cfg) {
    // Read from persistent storage
    ReadFromFlash(&cfg, sizeof(cfg));

    if (cfg.storageKey != CURRENT_KEY) {
        // Key mismatch - initialize with defaults
        InitializeDefaults(cfg);
        cfg.storageKey = CURRENT_KEY;
        SaveConfig(cfg);
    }
}
  
```

10.7.3.2 Null Value Checking

Old IPv4 Pattern (No Longer Valid):**

```
if (EthernetIP == 0) {
    // Configure DHCP
}
```

New Dual-Stack Pattern:**

```
if (EthernetIP.IsNull()) {
    // Configure DHCP
}
```

10.7.3.3 Null IP in Function Parameters

Old IPv4 Pattern:**

```
DNSResult = GetHostByName(serverName, &serverIp, 0, TICKS_PER_SECOND * 20);
```

New Dual-Stack Patterns:**

```
// Option 1: Using NullIP() static method
DNSResult = GetHostByName(serverName, &serverIp, IPADDR::NullIP(), TICKS_PER_SECOND * 20);
```

```
// Option 2: Using INADDR_ANY constant
```

```
DNSResult = GetHostByName(serverName, &serverIp, INADDR_ANY, TICKS_PER_SECOND * 20);
```

Note: INADDR_ANY is defined as `IPADDR::NullIP()` in system headers.

10.7.4 IPADDR Member Functions

The IPADDR class uses object-oriented syntax with member functions accessed via dot notation.

10.7.4.1 Function Types

```
IPADDR Object
|
+-- Member Functions (instance.function())
|   |
|   +-- Setting values
|   +-- Checking values
|   +-- Output functions
|
+-- Static Functions (IPADDR::function())
    |
    +-- Factory methods
    +-- Utility functions
```

10.7.4.2 Setting IP Address Values

Member Functions (Instance Methods):**

```
IPADDR myIpObject;

// Set to null
myIpObject.SetNull();

// Set from string
myIpObject.SetFromAscii("192.168.1.100");
myIpObject.SetFromAscii("2001:db8::1");

// Set from IPv4 address
IPADDR4 ipv4Addr = 0xC0A80164; // 192.168.1.100
myIpObject.SetFromIP4(ipv4Addr);
```

Static Functions (Class Methods):**

```
// Create null IP address
IPADDR nullAddr = IPADDR::NullIP();

// Create from ASCII string
IPADDR serverAddr = IPADDR::AsciiToIp("192.168.1.1");
IPADDR ipv6Addr = IPADDR::AsciiToIp("2001:db8::8a2e:370:7334");
```

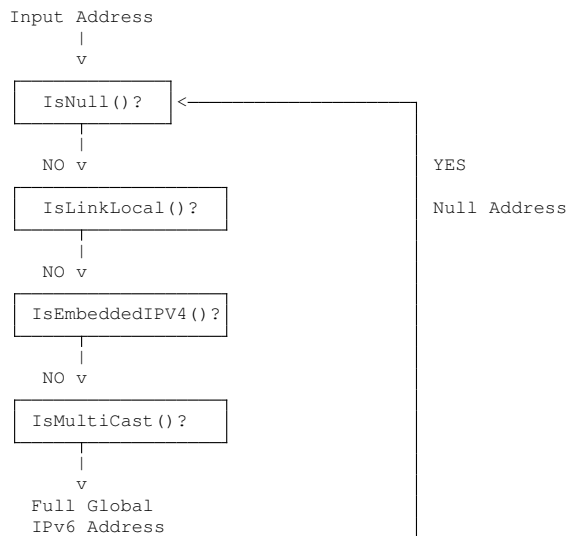
10.7.4.3 Reading and Checking IP Address Values

Boolean Checks:**

```
IPADDR addr;

if (addr.IsNull()) { /* Address is null */ }
if (addr.IsLoopBack()) { /* Is loopback (127.0.0.1 or ::1) */ }
if (addr.IsMultiCast()) { /* Is multicast address */ }
if (addr.IsLinkLocal()) { /* Is IPv6 link-local (fe80::/10) */ }
if (addr.IsEmbeddedIPv4()) { /* Is IPv4-mapped IPv6 (::FFFF:x.x.x.x) */ }
```

Address Type Detection:**

**Multicast MAC Conversion:****

```

IPADDR mcastAddr = IPADDR::AsciiToIp("ff02::1");
if (mcastAddr.IsMultiCast()) {
    MACADDR mac = mcastAddr.McastMac();
    // Use MAC address for multicast operations
}

```

10.7.4.4 Output Functions**Console Output:****

```
IPADDR addr = IPADDR::AsciiToIp("192.168.1.100");
```

```

// Print to stdout (debug serial port)
addr.printf();
// Output: 192.168.1.100

```

File Descriptor Output:**

```
int tcpSocket = /* ... established TCP connection ... */;
```

```

// Print to TCP socket
serverAddr.fprintf(tcpSocket);

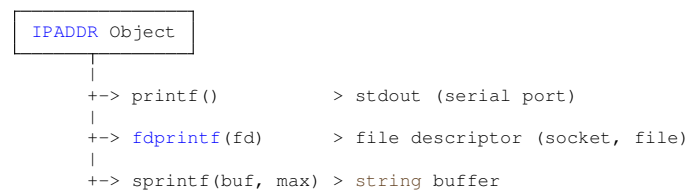
```

String Buffer Output:**

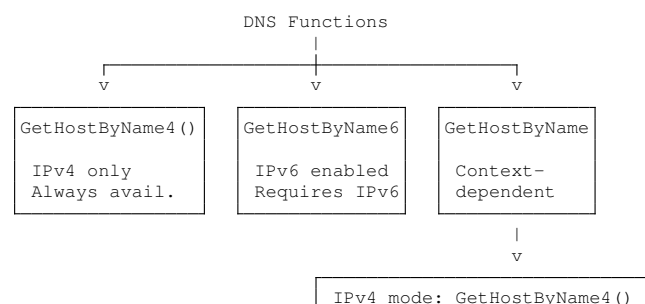
```

char buffer[128];
int written = addr.sprintf(buffer, sizeof(buffer));
printf("Address: %s (%d chars)\n", buffer, written);

```

Output Method Comparison:****10.7.5 Domain Name Service (DNS)**

DNS function signatures vary between IPv4 and IPv6 modes, but can be written to work identically in both.

10.7.5.1 DNS Function Variants

IPv6 mode: `GetHostByName6()`

10.7.5.2 Universal DNS Code Pattern

Works in both IPv4 and IPv6 modes:**

```
IPADDR resultIP;
int result = GetHostByName(
    "thename.com",           // Hostname to resolve
    &resultIP,               // Result storage
    IPADDR::NullIP(),       // Use system DNS server
    10 * TICKS_PER_SECOND   // Timeout
);

if (result == DNS_OK) {
    printf("Resolved to: ");
    resultIP.printf();
    printf("\n");
}
```

10.7.5.3 DNS Record Types

DNS Query
|
v

Record Type Selection	
DNS_A	- IPv4 address (A record)
DNS_AAAA	- IPv6 address (AAAA)
DNS_CNAME	- Canonical name
DNS_MX	- Mail exchange
DNS_MB	- Mailbox resource
DNS_MG	- Mail group resource

10.7.5.4 IPv6 DNS Priority Control

GetHostByName6 () supports dual-stack resolution with configurable priority:

```
int GetHostByName(
    const char *name,           // Name to resolve
    IPADDR *pResultIP,         // Result storage
    const IPADDR &dnsServerIP, // DNS server (or NullIP)
    WORD timeout,              // Timeout in ticks
    WORD TYPE1 = DNS_A,        // Primary type (default: IPv4)
    WORD TYPE2 = DNS_AAAA       // Secondary type (default: IPv6)
);
```

Resolution Flow:**

```

graph TD
    A[DNS Query: "example.com"] --> B[v]
    B --> C[Try TYPE1]
    C --> D[v]
    C --> E[v]
    D --> F[Success]
    E --> G[Failure]
    G --> H[v]
    H --> I[Try TYPE2]
    I --> J[v]
    I --> K[v]
    J --> L[Success]
    K --> M[Failure]
    L --> N[+]
    M --> O[v]
    N --> P[Return Result]
    O --> Q[Return Error]

```

Default Behavior (IPv4 first):**

```
// Tries IPv4 (DNS_A), falls back to IPv6 (DNS_AAAA)
GetHostByName("example.com", &result, IPADDR::NullIP(), 10 * TICKS_PER_SECOND);
```

IPv6-First Resolution:**

```
// Tries IPv6 first, falls back to IPv4
```

```
GetHostByName(
    "example.com",
    &result,
    IPADDR::NullIP(),
    10 * TICKS_PER_SECOND,
    DNS_AAAA, // Try IPv6 first
    DNS_A      // Fall back to IPv4
);
```


Replace with: - <code>IsNull()</code> - Key versioning

Replace with: - <code>NullIP()</code> - <code>INADDR_ANY</code>

10.7.7.2 Storage Migration

Step 1: Add Version Key

<pre>uint32_t version; IPADDR addresses...</pre>

|
v

Step 2: Check on Load

<pre>if (version != cur) Initialize()</pre>

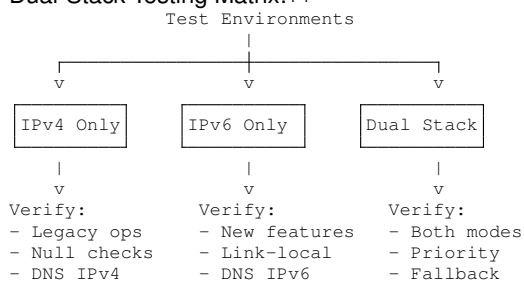
|
v

Step 3: Save with Key

<pre>config.version = cur SaveToFlash(config)</pre>

10.7.7.3 Testing Strategy

Dual-Stack Testing Matrix:**



10.7.8 Best Practices

10.7.8.1 Portable Code Patterns

Use `GetHostByName()` without version suffix:**

```
// Portable across IPv4/IPv6
GetHostByName(hostname, &result, IPADDR::NullIP(), timeout);
```

Always use `IsNull()` for comparisons:**

```
// Portable null checking
if (addr.IsNull()) {
    // Configure or error
}
```

Use `IPADDR::NullIP()` for null parameters:**

```
// Portable null parameter
Connect(host, port, IPADDR::NullIP());
```

10.7.8.2 Storage Design

Version-aware configuration structures:**

```
struct NetworkConfig {
    uint32_t configVersion; // First field
    IPADDR staticIP;
    IPADDR gateway;
    IPADDR dnsServer;
    // ... other fields
};
```

Always validate on load:**

```
void LoadConfig(NetworkConfig& cfg) {
    ReadFromStorage(&cfg, sizeof(cfg));

    if (cfg.configVersion != CURRENT_CONFIG_VERSION) {
        InitDefaults(cfg);
        SaveConfig(cfg);
    }
}
```

10.7.8.3 DNS Resolution

Prefer IPv6 when available:**

```
// IPv6-first resolution with IPv4 fallback
GetHostByName(
    hostname,
    &result,
    IPADDR::NullIP(),
    timeout,
    DNS_AAAA, // Try IPv6 first
    DNS_A     // IPv4 fallback
);
Specify timeout appropriately:**
// Allow time for both TYPE1 and TYPE2 queries
WORD timeout = 20 * TICKS_PER_SECOND; // Not 10 seconds
```

10.7.9 Summary

10.7.9.1 Key Takeaways

IPADDR is now an object, not a 32-bit integer
Use IsNull() instead of == 0
Use NullIP() instead of 0 in parameters
IPv6 interfaces have multiple addresses
Storage size changes: 32-bit > 128-bit
Use version keys for persistent storage
DNS supports dual-stack with priority control
Link-local addresses need scope ID on BSD/macOS

10.7.9.2 Migration Path

```
Current IPv4 Code
|
v
Audit for patterns
|
v
Update null checks
|
v
Add storage versioning
|
v
Test in IPv4 mode
|
v
Test in IPv6 mode
|
v
Test dual-stack
|
v
Deploy gradually
```

For detailed API documentation, consult the [IPADDR6](#) class reference in the NetBurner API documentation.

10.8 EFFS Programming Guide

10.8.1 NetBurner EFFS Programming Guide

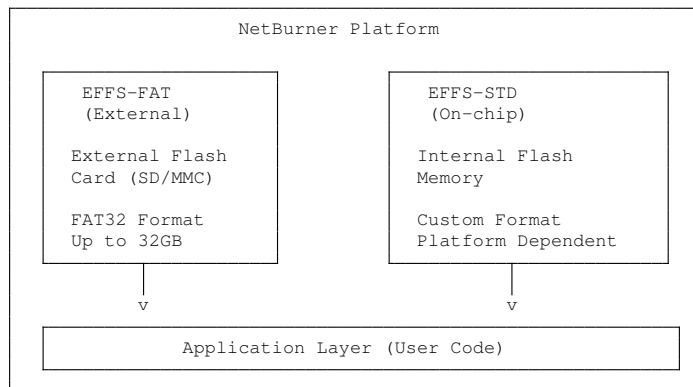
10.8.2 Overview

This guide covers the NetBurner implementation of the HCC Embedded Flash File System (EFFS) and NetBurner flash card hardware interface. EFFS provides file system capabilities for both external flash cards (EFFS-FAT) and on-chip flash memory (EFFS-STD).

10.8.2.1 Topics Covered

- Example programs and supported platforms
- EFFS-FAT file system for external flash cards
- EFFS-STD file system for on-chip flash memory
- Hardware interface design for external flash cards
- Debug port configuration

10.8.3 System Architecture



10.8.4 Supported Platforms

10.8.4.1 Platform Capabilities Matrix

Platform	External Flash Card	On-board Flash Card
MODM7AE70	SPI	N/A
MOD54415	SPI	SPI
MOD54417	SPI	SPI
SB800EX	SDIO	-
NANO54415	SPI	SDIO

Note: All platforms support EFFS-STD on-chip flash file system

10.8.4.2 Interface Types

SPI Mode**: Serial Peripheral Interface

- Single data channel operation
 - Lower data transfer rates
 - Simpler hardware implementation
- SDIO Mode**: Secure Digital Input/Output
- Four parallel data channels
 - Higher data transfer rates
 - More complex hardware design

10.8.5 Example Programs

10.8.5.1 EFFS-FAT Examples

Located at: `\nburn\examples\EFFS\Fat`

Available examples demonstrate:

- Simple file storage operations
- FTP server integration
- HTTP server with file system
- HTTP with variable tags
- Multi-task file system access
- RAM drive implementation
- Application updates from flash

10.8.5.2 EFFS-FAT Security Examples

Advanced security implementations:

`HttpsDualCert**`

- Dual certificate support (compiled-in and SD card-based)
- Dynamic certificate loading from EFFS-FAT
- Web server SSL/TLS configuration
`SSL_pop3**`
- Secure email retrieval via SSL
- Email storage to SD/MMC flash card
- POP3 protocol implementation
`SendMailAttach**`
- File attachment from EFFS-FAT
- Secure email transmission with SSL/TLS
- SMTP integration

10.8.5.3 EFFS-STD Examples

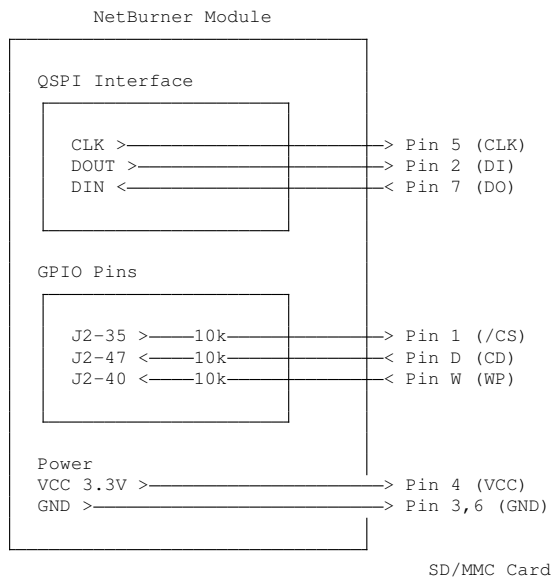
Located at: `\nburn\examples\EFFS\Std`

`HTTP Example**`

- Web page serving from STD file system
- Application-level page serving
- File system override capability
`Security Examples** (\nburn\examples\SSL)`
`HttpsUploadCert**`
- Certificate and key upload via web interface
- SSL/TLS credential storage in EFFS-STD
`SslVerifyPeerEfs**`
- CA List upload and storage
- Peer verification support
- Certificate authority management

10.8.6 Flash Card Hardware Interface

10.8.6.1 SD/MMC Hardware Connection Diagram



10.8.6.2 Signal Connection Table

SD/MMC Pin	Signal Name	NetBurner Connection
1	/CS (Chip Select)	J2-35 (10k pull-up)
2	DI (Data In)	J2-28 (QSPI DOUT)
3	GND	Ground
4	VCC	VCC 3.3V
5	CLK	J2-J5 (QSPI CLK)
6	GND	Ground
7	DO (Data Out)	J2-J7 (QSPI DIN)
8	NC	No Connection
9	NC	No Connection
D	CD (Card Detect)	J2-47 (10k pull-up)
W	WP (Write Protect)	J2-40 (10k pull-up)
Com	Common	Ground

Note: GPIO pins (CS, CD, WP) are configurable in mmc_m7.cpp or mmc_mcf.cpp depending on platform

10.8.6.3 Hardware Specifications

SD/MMC Connector**: Mouser Electronics Part # 688-SCDA1A0901

Card Compatibility**:

- Must support native SPI mode transfers
- Standard SD/MMC: Up to 2GB

- SDHC: Up to 32GB

SPI Interface Requirements**:

- EFFS normally requires exclusive use of SPI interface
- Sharing SPI with other peripherals is not officially supported

- Exclusive use recommended for optimal performance

10.8.6.4 GPIO Pin Configuration

Configure interface control pins in platform-specific files:

MODM7AE70**: \nburn\platform\<platform>\mmc_m7.cpp Other Platforms**: \nburn\platform\<platform>\mmc_mcf.cpp

Key functions to modify:

```
void MMC_BaseInit(int CurrentDrive) // Configure SD/MMC interface pins
int get_cd(int currentDrive) // Get Card Detect state
int get_wp(void) // Get Write Protect state
void spi_cs_lo(void) // Set SPI chip select low
void spi_cs_hi(void) // Set SPI chip select high
```

10.8.6.5 Interrupt-Driven SPI (ColdFire Platforms Only)

For MOD5441X, SB800EX, and NANO54415 platforms:

Enable interrupt-driven SPI for improved system performance at the cost of slightly lower SD/MMC performance.

This is beneficial when network or user tasks need priority during file system access.

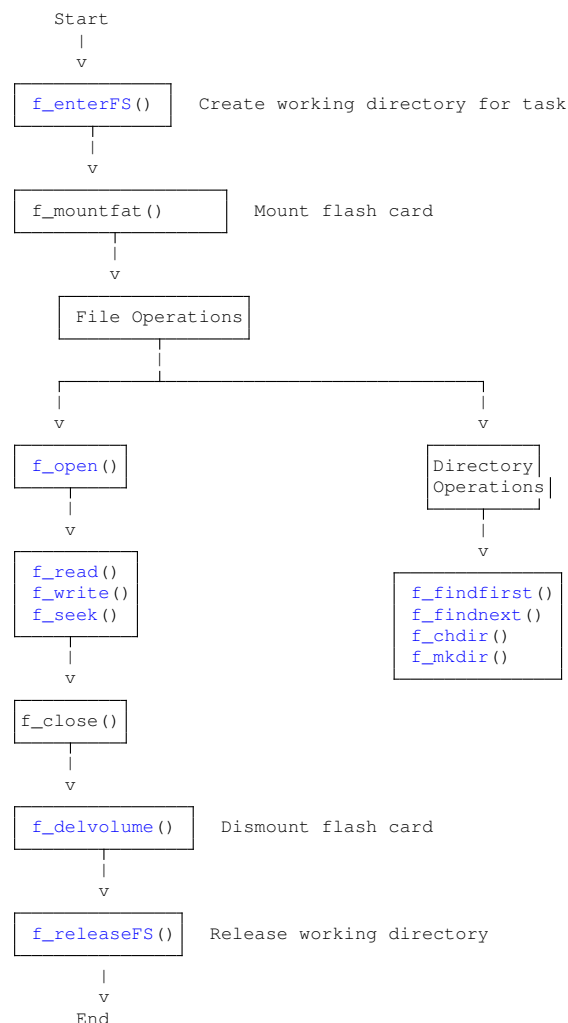
Configuration**: Uncomment in \nburn\platform\<platform>\mmc_mcf.cpp:

```
#define SD_IRQ_SPI
```

Then rebuild the platform system directory.

10.8.7 EFFS-FAT File System Operation

10.8.7.1 Typical Operation Flow



10.8.7.2 Common EFFS-FAT Function Calls

10.8.7.2.1 Working Directory Management

```
int f_enterFS(void);           // Create working directory for current task
void f_releaseFS(void);       // Release working directory
```

10.8.7.2.2 Mount/Dismount Operations

```
int f_mountfat(MMC_DRV_NUM, mmc_initfunc, F_MMC_DRIVE0); // Mount flash card
int f_delvolume(int drivenum);                          // Dismount flash card
```

10.8.7.2.3 File Operations

```
F_FILE *f_open(const char *filename, const char *mode); // Open file
int f_close(F_FILE *filehandle);                       // Close file
```

10.8.7.2.4 Read/Write Operations

```
int f_getfreespace(int drivenum, F_SPACE *pspace);
long f_write(const void *buf, long size, long size_st, F_FILE *filehandle);
long f_read(void *buf, long size, long size_st, F_FILE *filehandle);
long f_seek(F_FILE *filehandle, long offset, long whence);
int f_eof(F_FILE *filehandle);
int f_rewind(F_FILE *filehandle);
int f_delete(const char *filename);
```

10.8.7.2.5 Directory Operations

```
int f_findfirst(const char *filename, F_FIND *find);
int f_findnext(F_FIND *find);
int f_chdir(const char *dirname);
int f_mkdir(const char *dirname);
```

10.8.7.2.6 Time/Date Functions

```
int f_settime(const char *filename, unsigned short ctime, unsigned short cdate);
int f_gettime(const char *filename, unsigned short *ptime, unsigned short *pdate);
```

10.8.7.3 File Time and Date Stamps

EFFS supports file time and date stamps obtained through:

- Network Time Protocol (NTP) server
- Real-Time Clock (RTC)
- Manual configuration

Note**: If no time/date is set, files will be stamped with January 1, 1980.

10.8.7.4 File System Utilities

All NetBurner EFFS examples include `FileSystemUtils.cpp` providing:

- Easy-to-use initialization interface
- Status checking functions
- Testing utilities
- Read/write helpers

Location**:

- FAT utilities: `<nburn_install>\examples\_common\EFFS\FAT`
- STD utilities: `<nburn_install>\examples\_common\EFFS\STD`

Card Type Selection**: Edit `cardtype.h` to select between CF or SD/MMC cards.

10.8.8 EFFS-FAT Project Configuration

10.8.8.1 NBEclipse Project Setup

10.8.8.1.1 Step 1: Add EFFS-FAT Library

1. Right-click project in Project Explorer
2. Navigate to: C/C++ Build > Settings > GNU C/C++ Linker > Libraries
3. Add "FatFile" to Libraries (-l) list

10.8.8.1.2 Step 2: Import Source Files

Import files from: <nburn_install>\examples_common\EFFS\FAT

10.8.8.2 Makefile Configuration

Add the following lines to your project makefile:

```
XTRALIB = $(NNDK_ROOT)/platform/$(PLATFORM)/original/lib/libFatFFile.a
include $(NNDK_ROOT)/examples/_common/EFFS/FAT/common.mak
```

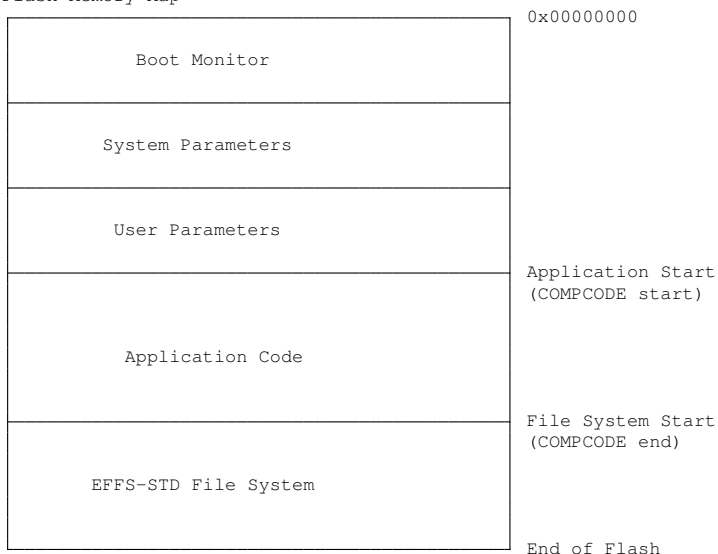
10.8.9 EFFS-STD File System (On-Chip Flash)

10.8.9.1 EFFS-FAT vs EFFS-STD Comparison

Feature	EFFS-FAT	EFFS-STD
Storage	External Flash Card (SD/MMC)	On-chip Flash Memory
Format	FAT32	Custom
Size	Up to 32GB	Platform Dependent
Functions	f_* prefix (e.g., f_open())	fs_* prefix (e.g., fs_open())
Library	libFatFile.a	libStdFFile.a
Removable	Yes	No

10.8.9.2 Flash Memory Layout

Flash Memory Map



10.8.9.3 Memory Allocation Considerations

When using EFFS-STD, you must:

1. Determine application code size (use compressed size from build output)
2. Allocate flash sectors for file system
3. Ensure sufficient room for application growth
4. Configure COMPCODE flags to define memory boundaries

Flash memory is divided into sectors (typically 4KB or 64KB). The EFFS-STD file system requires allocation of specific sectors. Consult your platform's flash memory datasheet for architecture details.

10.8.10 EFFS-STD Project Configuration

10.8.10.1 NBEclipse Configuration

10.8.10.1.1 Step 1: Import Source Files

Import from: `<nburn_install>\examples\_common\EFFS\STD`

10.8.10.1.2 Step 2: Add EFFS-STD Library

1. Right-click project
2. Navigate to: `C/C++ Build > Settings > GNU C/C++ Linker > Libraries`
3. Add "StdFFFile" to Libraries section (no .a extension)

10.8.10.1.3 Step 3: Platform-Specific Configuration

10.8.10.1.3.1 MODM7AE70, SBE70LC Platforms

Enable "Locate Application at Fixed Address in Flash":

1. Right-click project > Properties
2. Navigate to: `C/C++ Build > Settings > NB Flashpack > General`
3. Check "Locate Application at Fixed Address in Flash"

Purpose**: Disables background flash erase to preserve file system sectors

10.8.10.1.3.2 NANO54415, SB800EX Platforms

Modify CompCode flags to reserve file system space:

1. Right-click project
2. Navigate to: `C/C++ Build > Settings > NBCompCode > General`
3. Change memory address range:

Original: `COMPCODEFLAGS = 0x04000 0x800000`
 Modified: `COMPCODEFLAGS = 0x04000 0x700000`

Parameters**:

- First address: Application start
- Second address: File system start

10.8.10.1.3.3 MOD54415, MOD54417 Platforms

Modify CompCode flags:

1. Right-click project
2. Navigate to: `C/C++ Build > Settings > NBCompCode > General`
3. Change memory address range:

Original: `COMPCODEFLAGS = 0xC0040000 0xC2000000`
 Modified: `COMPCODEFLAGS = 0xC0040000 0xC1F00000`

10.8.11 EFFS-STD Makefile Configuration

10.8.11.1 Step 1: Add EFFS-STD Library

```
XTRALIB = $(NNDK_ROOT)/platform/$(PLATFORM)/original/lib/libStdFFile.a
DBXTRALIB = $(NNDK_ROOT)/platform/$(PLATFORM)/original/lib/libStdFFile.a
```

10.8.11.2 Step 2: Platform-Specific Configuration

10.8.11.2.1 MODM7AE70, SBE70LC

Enable fixed application address:

```
EXTRAPACKARGS += -cflag C:3
```

10.8.11.2.2 NANO54415, SB800EX

Modify memory address range:

```
# Original
COMPCODEFLAGS = 0x04000 0x800000
```

```
# Modified for file system
COMPCODEFLAGS = 0x04000 0x700000
```

10.8.11.2.3 MOD54415, MOD54417

Modify memory address range:

```
# Original
COMPCODEFLAGS = 0xC0040000 0xC2000000
```

```
# Modified for file system
COMPCODEFLAGS = 0xC0040000 0xC1F00000
```

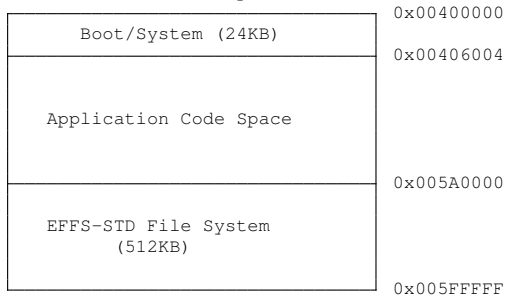
10.8.12 Flash Memory Address Configuration

10.8.12.1 Platform Memory Map

Platform	Total Size	Start Address	End Address	Configuration Example
MODM7AE70	2MB	0x00400000	0x005FFFFF	512K Flash File System: Application: 0x00406004 File System: 0x005A0000 COMPCODEFLAGS = 0x00406004 0x005A0000 #define FLASH_SIZE (2*1024*1024) #define FS_SIZE (256*512)

10.8.12.2 Memory Region Diagram

MODM7AE70 Flash Memory (2MB Total)



10.8.12.3 Configuration File Reference

For SAME70Q21 platform configuration details, refer to: [\nburn\examples_common\EFFS\STD\src\flash←Chip\SAME70Q21.h](#)

10.8.13 Debug Port Configuration

10.8.13.1 Overview

The debug port is an RS-232 serial port used for interactive communication with your NetBurner device during development and testing.

10.8.13.2 Default Configuration

By default, the debug port has the following mappings:

Standard I/O Mappings	
stdout	> Debug Port (console output)
stdin	> Debug Port (console input)
stderr	> Debug Port (error output)

10.8.13.3 Supported Functions

Standard C/C++ I/O functions work with the debug port:

```
printf()    // Formatted output to debug port
scanf()     // Formatted input from debug port
gets()      // Read string from debug port
puts()      // Write string to debug port
getchar()   // Read single character
putchar()   // Write single character
```

10.8.13.4 Custom Configuration

The debug port is fully configurable:

Option 1*: Disable and use as general-purpose UART

- Frees port for custom communication protocols
- Suitable for production deployments
- Option 2*: Reassign stdio descriptors
- Route stdout/stdin/stderr to network interfaces
- Use telnet or SSH for remote debugging
- Implement custom logging mechanisms
- Option 3*: Multiple debug interfaces
- Combine serial and network debug output
- Create redundant logging paths
- Support simultaneous local and remote debugging

10.8.14 API Reference Documentation

10.8.14.1 EFFS-FAT API Reference

Complete function documentation located at: `\nburn\docs\NetBurner\EFFS`

10.8.14.2 EFFS-STD API Reference

Detailed function documentation available in NetBurner documentation under `@ref groupEFFSSTD`

10.8.14.3 Key Function Mapping

EFFS-FAT (External)	EFFS-STD (On-chip)
<code>f_enterFS()</code> <code>f_releaseFS()</code> <code>f_open()</code> <code>f_close()</code> <code>f_read()</code> <code>f_write()</code> <code>f_seek()</code> <code>f_delete()</code> <code>f_chdir()</code> <code>f_mkdir()</code>	<code>fs_enterFS()</code> <code>fs_releaseFS()</code> <code>fs_open()</code> <code>fs_close()</code> <code>fs_read()</code> <code>fs_write()</code> <code>fs_seek()</code> <code>fs_delete()</code> <code>fs_chdir()</code> <code>fs_mkdir()</code>

10.8.15 Best Practices

10.8.15.1 General Guidelines

1. **Always check return values** from file system functions
2. **Close files properly** to prevent data loss
3. **Handle card removal gracefully** in EFFS-FAT applications
4. **Reserve adequate flash space** for EFFS-STD file systems
5. **Test with actual hardware** before production deployment

10.8.15.2 EFFS-FAT Specific

- Verify card is inserted before mounting
- Implement card detect monitoring in applications
- Handle write-protect status appropriately
- Use proper file modes for read/write operations
- Dismount volumes before card removal when possible

10.8.15.3 EFFS-STD Specific

- Calculate application size carefully
- Leave room for application growth
- Understand flash sector architecture
- Consider wear leveling implications
- Don't exceed allocated file system space

10.8.15.4 Performance Optimization

- Use interrupt-driven SPI when system responsiveness is critical
- Batch file operations when possible
- Minimize file open/close cycles
- Use appropriate buffer sizes for read/write operations
- Consider RAM drive for frequently accessed temporary files

10.8.16 Troubleshooting

10.8.16.1 Common EFFS-FAT Issues

Card Not Detected**

- Check physical connections
- Verify power supply (3.3V)
- Confirm card detect pin configuration
- Test with known-good card

Mount Failures**

- Ensure card is formatted FAT32
- Verify card capacity limits (2GB for standard, 32GB for SDHC)
- Check SPI interface configuration
- Confirm correct initialization sequence

Read/Write Errors**

- Check write-protect status
- Verify sufficient free space
- Ensure proper file handles
- Confirm correct file modes

10.8.16.2 Common EFFS-STD Issues

Insufficient Space**

- Recalculate application size
- Adjust COMPCODE flags
- Reduce file system allocation if needed
- Verify compressed vs uncompressed size

Flash Erase Conflicts**

- Ensure fixed address mode enabled (MODM7AE70/SBE70LC)
- Check memory map configuration
- Verify sector boundaries
- Confirm no overlap with application space

File System Corruption**

- Avoid power interruption during writes
- Implement proper shutdown sequences
- Check flash wear level
- Verify sector allocation

10.8.17 Additional Resources

10.8.17.1 Documentation References

- **EFFS API Reference:** `\nburn\docs\NetBurner\EFFS`
- **Platform Data Sheets:** Consult for flash memory specifications
- **Hardware Schematics:** Development board design references
- **Example Applications:** `\nburn\examples\EFFS`

10.8.17.2 Support Files

- `FileSystemUtils.cpp`: Common utilities for all examples
- `cardtype.h`: Flash card type selection
- Platform-specific `mmc_*.cpp`: Hardware interface configuration
- Flash chip headers: Platform memory configuration

10.8.17.3 Example Locations

```

\nburn\examples\
├── EFFS\
│   ├── FAT\           > EFFS-FAT examples
│   ├── STD\          > EFFS-STD examples
│   └── SSL\           > Security examples with EFFS
├── common\
│   └── EFFS\
│       ├── FAT\       > Shared FAT utilities
│       └── STD\       > Shared STD utilities

```

10.8.18 Summary

The NetBurner EFFS provides robust file system capabilities for embedded applications:

- **EFFS-FAT:** External flash card storage with FAT32 compatibility
- **EFFS-STD:** On-chip flash storage with custom format
- **Easy Integration:** Well-documented APIs and extensive examples
- **Flexible Hardware:** Multiple platform support with various interfaces
- **Security Ready:** Integration with NetBurner security libraries

Choose EFFS-FAT for removable storage and large capacity requirements. Choose EFFS-STD for integrated, non-volatile storage in resource-constrained applications.

Both file systems provide reliable, production-ready storage solutions for NetBurner embedded systems.

10.9 File Descriptors

10.9.1 File Descriptors

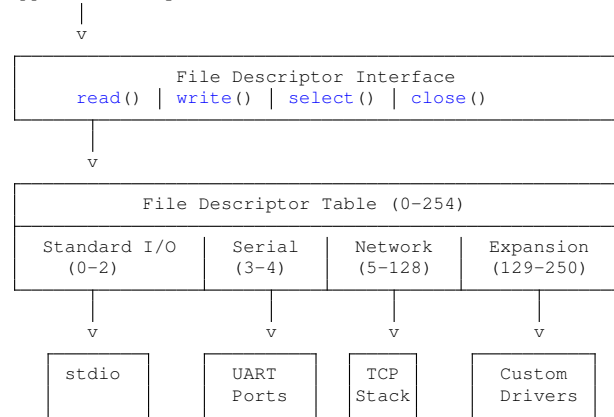
10.9.2 Overview

The NetBurner environment provides a unified I/O abstraction layer through file descriptors (FDs), seamlessly integrating the RTOS, TCP/IP stack, and hardware peripherals into a standard POSIX-like interface. A file descriptor serves as a handle to various I/O resources including network sockets, serial ports, system peripherals, and custom I/O devices.

This unified approach enables consistent I/O operations across different resource types using standard functions like `read()`, `write()`, `select()`, and `close()`.

10.9.3 File Descriptor Architecture

Application Layer



10.9.4 File Descriptor Allocation

NetBurner provides a maximum of **255 file descriptors** (0-254), organized into specific ranges:

10.9.4.1 Allocation Map

File Descriptor Space			
0-2	3-4	5-128	129-250
Standard I/O	Serial Ports	TCP Sockets	Expansion Pool
3 FDs	2 FDs	124 FDs	122 FDs
Fixed System	Fixed UART 0-1	32 TCP Connections	Available for Custom Use

10.9.4.2 Detailed Breakdown

Range	Count	Purpose	Description
0-2	3	Standard I/O	stdin (0), stdout (1), stderr (2)
3-4	2	Serial Ports	UART 0 (fd=3), UART 1 (fd=4)
5-128	124	TCP Sockets	Network connections (32 sockets)
129-250	122	Expansion	Custom drivers, additional UARTs, buffers

10.9.4.3 Expansion Pool Usage

The expansion range (129-250) can accommodate:

Expansion File Descriptors (129-250)

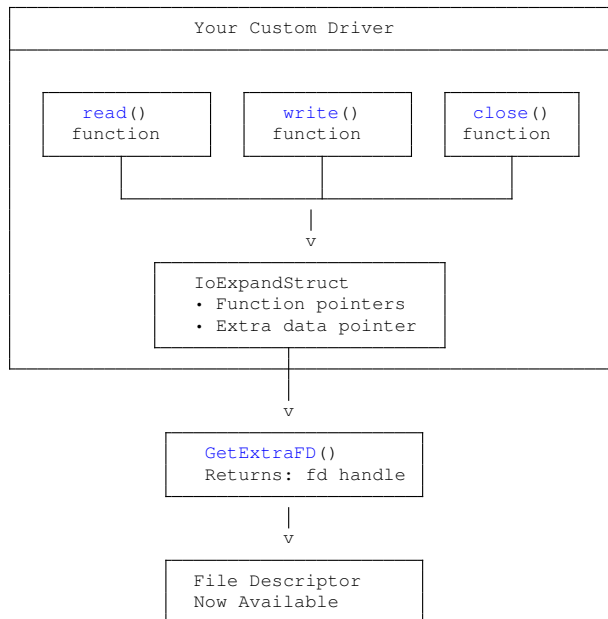
External UARTs SPI-to-Serial	Additional TCP Sockets Beyond the 32 default	Custom Buffers Ring Buffers
I2C Devices I2C bus access	Custom Protocols Proprietary interfaces	Virtual Devices Memory streams

10.9.5 Creating Custom I/O Drivers

10.9.5.1 Driver Architecture

Custom file descriptors enable integration of any I/O device into the standard file descriptor framework.

Custom Driver Implementation Flow



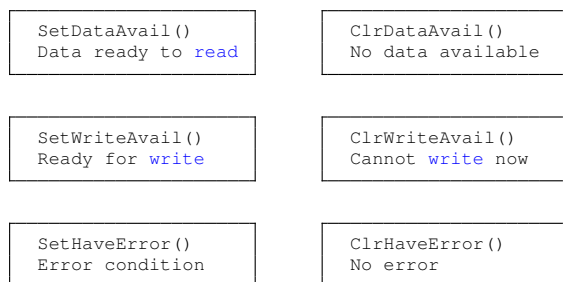
10.9.5.2 Core Header File

```
#include <iointernal.h> // Located in: <NNDK_ROOT>/include/
```

10.9.5.3 State Management Functions

Functions to control the state of your file descriptor:

State Control Functions



Purpose:** These functions update the state of your file descriptor for use with I/O functions such as `select()`, `read()`, and `write()`.

10.9.6 IoExpandStruct Definition

The core structure for custom file descriptor implementation:

```

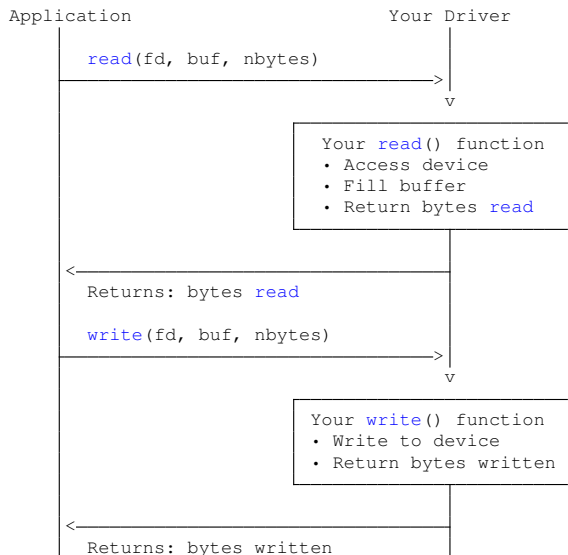
struct IoExpandStruct
{
    int (*read)(int fd, char *buf, int nbytes); // Read function
    int (*write)(int fd, const char *buf, int nbytes); // Write function
    int (*close)(int fd); // Close function
    void *extra; // Custom data pointer
};

```

10.9.6.1 Structure Members

Member	Type	Purpose
read	Function pointer	Implements read operations for your device
write	Function pointer	Implements write operations for your device
close	Function pointer	Cleanup and release resources
extra	void*	Optional pointer to device-specific data

10.9.6.2 Data Flow



10.9.7 File Descriptor Management API

10.9.7.1 Core Functions

```

// Allocate a new file descriptor
int GetExtraFD(void *extra_data, struct IoExpandStruct *pFuncs);

// Retrieve custom data associated with a file descriptor
void* GetExtraData(int fd);

// Release a file descriptor back to the pool
void FreeExtraFD(int fd);

```

10.9.7.2 Function Details

10.9.7.2.1 GetExtraFD()

Purpose:** Allocate a file descriptor for your custom I/O device.

Parameters:**

- `extra_data`: Optional pointer to device-specific data
- `pFuncs`: Pointer to `IoExpandStruct` containing your function implementations

Returns:** File descriptor handle (int), or -1 on failure

Usage Example:**

```

IoExpandStruct myDeviceFuncs = {
    .read = myReadFunction,
    .write = myWriteFunction,
    .close = myCloseFunction,
    .extra = &myDeviceData
};

int fd = GetExtraFD(&myDeviceData, &myDeviceFuncs);
if (fd < 0) {
    printf("Failed to allocate file descriptor\n");
}

```

10.9.7.2.2 GetExtraData()

Purpose:** Retrieve the custom data pointer associated with a file descriptor.

Parameters:**

- fd: File descriptor handle

Returns:** Pointer to extra data (void*), or NULL if not set

Usage:**

```
MyDeviceStruct *device = (MyDeviceStruct*)GetExtraData(fd);
if (device != NULL) {
    // Access device-specific data
    device->status = READY;
}
```

10.9.7.2.3 FreeExtraFd()

Purpose:** Release a file descriptor back to the available pool.

Parameters:**

- fd: File descriptor to release

Important:** Always call this when your device is closed to prevent descriptor leaks.

```
void myCloseFunction(int fd) {
    // Cleanup device resources
    MyDeviceStruct *device = (MyDeviceStruct*)GetExtraData(fd);
    if (device) {
        // Device-specific cleanup
        device->shutdown();
    }

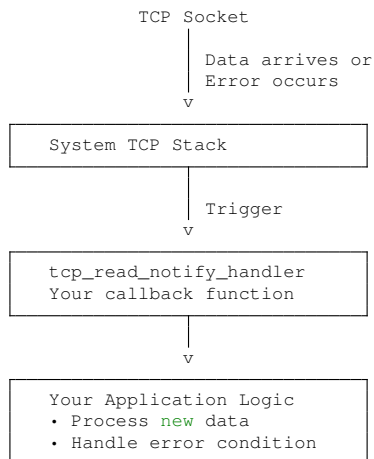
    // Release the file descriptor
    FreeExtraFd(fd);
}
```

10.9.8 TCP Socket Notifications

10.9.8.1 Callback System

NetBurner provides a callback mechanism for TCP socket events:

TCP Event Flow



10.9.8.2 Callback Definition

```
// Callback function type
typedef void (tcp_read_notify_handler)(int tcp_fd);

// Registration function
void RegisterTCPReadNotify(int tcp_fd, tcp_read_notify_handler *newhandler);
```

10.9.8.3 Callback Parameters

When your callback is invoked:

Condition	tcp_fd Value	Meaning
Data available	fd >= 0	Socket has new data to read
Error state	fd < 0	TCP connection has entered error state

10.9.8.4 Implementation Example

```
// Your callback handler
void myTcpNotifyHandler(int tcp_fd) {
    if (tcp_fd >= 0) {
        // New data available
        char buffer[256];
        int bytesRead = read(tcp_fd, buffer, sizeof(buffer));
        if (bytesRead > 0) {
            // Process received data
            processIncomingData(buffer, bytesRead);
        }
    } else {
        // Error condition
        printf("TCP socket error detected\n");
        handleTcpError();
    }
}

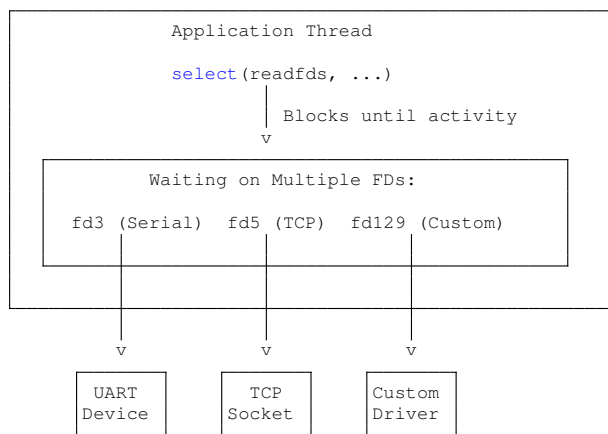
// Register the callback
void setupTcpSocket(int tcp_fd) {
    RegisterTCPReadNotify(tcp_fd, myTcpNotifyHandler);
}
```

10.9.9 Using select() with File Descriptors

10.9.9.1 Multiplexing Multiple I/O Sources

One of the most powerful features of the file descriptor system is the ability to wait on multiple I/O sources simultaneously using `select()`.

`select()` Multiplexing



10.9.9.2 select() Usage Pattern

```
#include <iosys.h>

void monitorMultipleSources() {
    int serial_fd = 3;      // UART 0
    int tcp_fd = 5;         // TCP socket
    int custom_fd = 129;    // Custom device

    fd_set readfds;

    while (1) {
        // Initialize the set
        FD_ZERO(&readfds);
        FD_SET(serial_fd, &readfds);
        FD_SET(tcp_fd, &readfds);
        FD_SET(custom_fd, &readfds);

        // Find the highest fd number
        int max_fd = custom_fd;

        // Wait for activity on any descriptor
    }
}
```

```

int result = select(max_fd + 1, &readfds, NULL, NULL, NULL);

if (result > 0) {
    // Check which descriptor has activity
    if (FD_ISSET(serial_fd, &readfds)) {
        handleSerialData(serial_fd);
    }

    if (FD_ISSET(tcp_fd, &readfds)) {
        handleTcpData(tcp_fd);
    }

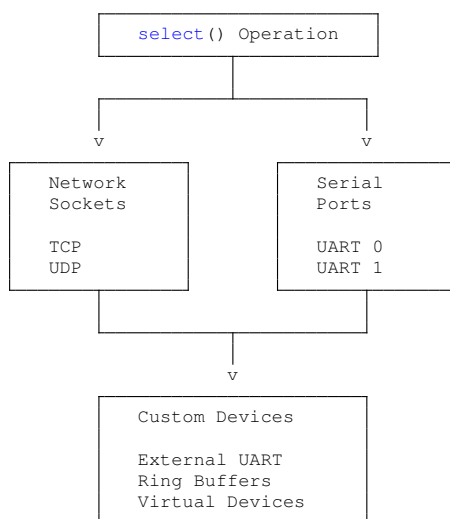
    if (FD_ISSET(custom_fd, &readfds)) {
        handleCustomDevice(custom_fd);
    }
}
}
}

```

10.9.9.3 Mixing Device Types

You can combine any file descriptor types in a single `select()` call:

Mixed Device Selection



10.9.10 Complete Implementation Example

10.9.10.1 Custom Ring Buffer Device

Here's a complete example of implementing a ring buffer as a file descriptor:

```

#include <iointernal.h>
#include <nbrtos.h>

// Device-specific data structure
typedef struct {
    char buffer[1024];
    int read_pos;
    int write_pos;
    int count;
    OS_CRIT critical_section;
} RingBufferDevice;

// Read function implementation
int ringBufferRead(int fd, char *buf, int nbytes) {
    RingBufferDevice *dev = (RingBufferDevice*)GetExtraData(fd);
    if (!dev) return -1;

    OSCritEnter(&dev->critical_section, CRIT_FLAG_NONE);

    int bytes_to_read = (nbytes < dev->count) ? nbytes : dev->count;
    int bytes_read = 0;

    for (int i = 0; i < bytes_to_read; i++) {
        buf[i] = dev->buffer[dev->read_pos];
        dev->read_pos = (dev->read_pos + 1) % 1024;
        dev->count--;
        bytes_read++;
    }
}

```

```

    if (dev->count == 0) {
        ClrDataAvail(fd); // No more data available
    }

    OSCritLeave(&dev->critical_section);
    return bytes_read;
}

// Write function implementation
int ringBufferWrite(int fd, const char *buf, int nbytes) {
    RingBufferDevice *dev = (RingBufferDevice*)GetExtraData(fd);
    if (!dev) return -1;

    OSCritEnter(&dev->critical_section, CRIT_FLAG_NONE);

    int space_available = 1024 - dev->count;
    int bytes_to_write = (nbytes < space_available) ? nbytes : space_available;
    int bytes_written = 0;

    for (int i = 0; i < bytes_to_write; i++) {
        dev->buffer[dev->write_pos] = buf[i];
        dev->write_pos = (dev->write_pos + 1) % 1024;
        dev->count++;
        bytes_written++;
    }

    if (dev->count > 0) {
        SetDataAvail(fd); // Data available for reading
    }

    if (dev->count < 1024) {
        SetWriteAvail(fd); // Space available for writing
    } else {
        ClrWriteAvail(fd); // Buffer full
    }

    OSCritLeave(&dev->critical_section);
    return bytes_written;
}

// Close function implementation
int ringBufferClose(int fd) {
    RingBufferDevice *dev = (RingBufferDevice*)GetExtraData(fd);
    if (dev) {
        // Clean up resources
        delete dev;
    }
    FreeExtraFd(fd);
    return 0;
}

// Create ring buffer file descriptor
int createRingBufferFd() {
    RingBufferDevice *device = new RingBufferDevice;
    device->read_pos = 0;
    device->write_pos = 0;
    device->count = 0;
    OSCritInit(&device->critical_section);

    IoExpandStruct funcs = {
        .read = ringBufferRead,
        .write = ringBufferWrite,
        .close = ringBufferClose,
        .extra = device
    };

    int fd = GetExtraFD(device, &funcs);
    if (fd >= 0) {
        SetWriteAvail(fd); // Initially ready for write
    }

    return fd;
}

```

10.9.10.2 Usage of Custom Device

```

void useCustomRingBuffer() {
    // Create the custom file descriptor
    int ring_fd = createRingBufferFd();

    if (ring_fd < 0) {
        printf("Failed to create ring buffer\n");
        return;
    }
}

```

```

// Write data
const char *message = "Hello, Ring Buffer!";
write(ring_fd, message, strlen(message));

// Read data
char readBuffer[256];
int bytesRead = read(ring_fd, readBuffer, sizeof(readBuffer));
readBuffer[bytesRead] = '\0';

printf("Read from ring buffer: %s\n", readBuffer);

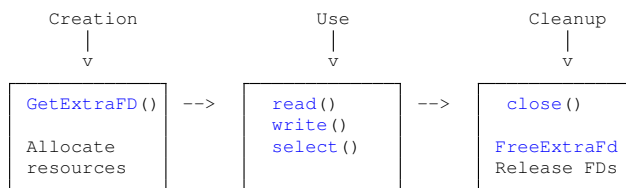
// Close when done
close(ring_fd);
}

```

10.9.11 Best Practices

10.9.11.1 Resource Management

File Descriptor Lifecycle



10.9.11.2 Guidelines

1. Always Release File Descriptors

- Call `FreeExtraFd()` in your close function
- Prevent descriptor exhaustion

2. Thread Safety

- Use critical sections for shared data
- Protect state changes with `OSCritEnter/Leave`

3. State Management

- Update availability flags appropriately
- Use `SetDataAvail()` when data arrives
- Use `ClrDataAvail()` when buffer empties

4. Error Handling

- Return appropriate error codes
- Use `SetHaveError()` for error conditions
- Check return values from all functions

5. Resource Initialization

- Initialize all structure members
- Verify allocations succeed
- Set initial state flags correctly

Problem	Cause	Solution

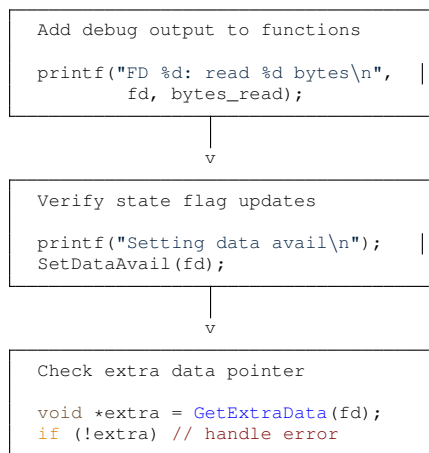
10.9.12 Troubleshooting

10.9.12.1 Common Issues

Problem	Cause	Solution
<code>select()</code> never returns	State flags not set	Call <code>SetDataAvail()</code> when data ready
File descriptor allocation fails	Pool exhausted	Release unused FDs with <code>FreeExtraFd()</code>
Data corruption	Missing critical sections	Protect shared data with <code>OSCritEnter/Leave</code>
Memory leaks	Not freeing resources	Implement proper cleanup in close function

10.9.12.2 Debugging Tips

Debug Flow



10.9.13 Summary

10.9.13.1 Key Concepts

File Descriptor System:**

- Unified I/O abstraction for all device types
- POSIX-like interface for consistent access

- 255 total descriptors with defined ranges

Custom Driver Implementation:**

- Define `read()`, `write()`, and `close()` functions
- Use `IoExpandStruct` to register functions
- Manage state with `Set/Clr` functions

Multiplexing:**

- Use `select()` to wait on multiple sources
- Mix network, serial, and custom devices
- Efficient event-driven programming

Best Practices:**

- Always release file descriptors
- Implement thread-safe operations
- Update state flags correctly
- Handle errors gracefully

10.9.13.2 Quick Reference

```
// Essential includes
#include <iointernal.h>

// Core functions
int fd = GetExtraFD(extra_data, &funcs);
void *data = GetExtraData(fd);
FreeExtraFd(fd);

// State management
SetDataAvail(fd);    ClrDataAvail(fd);
SetWriteAvail(fd);   ClrWriteAvail(fd);
SetHaveError(fd);    ClrHaveError(fd);

// TCP callbacks
RegisterTCPReadNotify(tcp_fd, handler);
```

The file descriptor system provides a powerful and flexible foundation for integrating diverse I/O resources into your NetBurner applications.

10.10 HTML Processing

10.10.1 HTML Processing

10.10.2 Introduction

The NetBurner Web Server automatically processes all content in your project's `html/` folder. Whether you're using HTML examples, the application wizard, or creating a project from scratch, the web server provides powerful tools for serving both static and dynamic content.

10.10.2.1 Quick Start

1. Create an `html/` folder in your project (if it doesn't exist)
2. Add your web content (HTML, CSS, JavaScript, images, etc.)
3. Call `StartHttp()` or `StartHttps()` in your application
4. Build your project - content is automatically compiled and linked

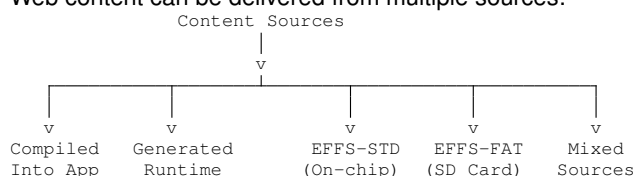
The NetBurner build system automatically compiles web content into your application image, eliminating the need for external file storage (though file system delivery is also supported).

10.10.2.2 Web Server Initialization

```
StartHttp();           // Default port 80
StartHttp(2000);        // Custom HTTP port
StartHttps();           // Default port 443 (HTTPS only)
StartHttps(527);        // Custom HTTPS port
StartHttps(443, 80);    // Both HTTPS and HTTP on standard ports
```

10.10.2.3 Content Sources

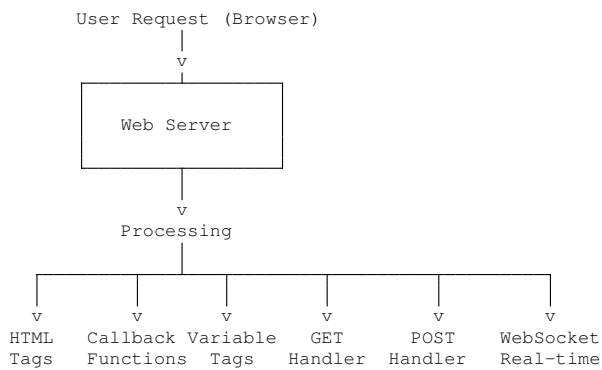
Web content can be delivered from multiple sources:



10.10.3 Dynamic Content Overview

Dynamic content is generated at runtime on your NetBurner device, enabling real-time updates, interactive controls, and data visualization.

10.10.3.1 Dynamic Content Methods



Available methods:

1. **HTML Tags** - Embed dynamic content directly in HTML
2. **Function Callbacks** - Execute C/C++ functions during page rendering
3. **Variable Tags** - Insert application variable values in real-time
4. **JavaScript** - Client-side dynamic behavior
5. **GET Handlers** - Intercept and customize page requests
6. **POST Handlers** - Process form submissions
7. **WebSockets** - Two-way real-time communication

10.10.4 HTML Tags

HTML tags enable real-time dynamic content by embedding directives in your HTML source code. When the web server streams content to the browser, it processes these tags and takes appropriate action.

10.10.4.1 Tag Overview

Tag	Purpose	Use Case
CPPCALL	Execute C++ function without parameters	Display computed data, trigger actions
VARIABLE ()	Execute C++ function with parameters	Parameterized callbacks, complex operations
VARIABLE	Insert application variable value	Display sensor readings, system state
CONFIGVALUE	Display configuration variable	Show system settings
CONFIGINPUT	Create input field with config value	Edit system configuration
CONFIGTABLE	Generate configuration table	Display hierarchical settings

10.10.4.2 Tag Syntax

Tags are embedded within HTML comments to avoid interfering with standard HTML processing:

```

<!--CPPCALL functionName -->
<!--VARIABLE expression -->
<!--VARIABLE functionName(fd, parameters) -->

```

Important*: Spacing matters!

- No space between <!-- and tag name
- Space required before -->

```

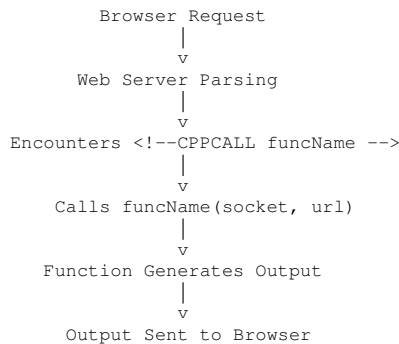
Correct: <!--CPPCALL myFunction -->
Incorrect: <!-- CPPCALL myFunction-->
Incorrect: <!--CPPCALL myFunction-->

```

10.10.5 The CPPCALL Tag

The CPPCALL tag executes a C++ function when encountered during page rendering. The function receives the socket file descriptor and URL, giving it full control over response generation.

10.10.5.1 Function Flow



10.10.5.2 Function Signature

```

void YourFunctionName(int sock, PCSTR url)
{
    // Your code here
    // Write to sock to send data to browser
}
  
```

10.10.5.3 Example: Display Client Information

HTML Code:**

```

<html>
<body>
    <h1>Connection Information</h1>
    Client Address: <!--CPPCALL webShowClientInfo -->
</body>
</html>
  
```

C++ Implementation:**

```

void webShowClientInfo(int sock, PCSTR url)
{
    IPADDR localIP = GetSocketLocalAddr(sock);
    uint32_t remotePort = GetSocketRemotePort(sock);

    localIP.fprintf(sock); // Write IP address
    fprintf(sock, ":%d<br>\r\n", remotePort); // Write port number
}
  
```

Result:**

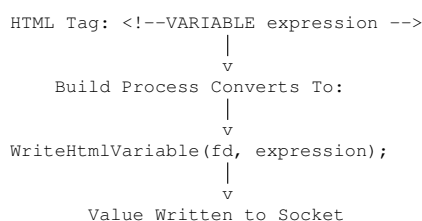
```
Client Address: 192.168.1.100:54321
```

10.10.6 The VARIABLE Tag

The VARIABLE tag has two distinct uses:

1. **Direct Variable Display** - Insert variable or expression value
2. **Parameterized Function Call** - Execute function with custom parameters

10.10.6.1 Use Case 1: Display Variables



Simple Variable Display:**

```
<!-- Display system uptime in ticks -->
```



```
System Ticks: <!--VARIABLE TimeTick -->

<!-- Display uptime in seconds using expression -->
Uptime: <!--VARIABLE TimeTick/TICKS_PER_SECOND --> seconds
```

10.10.6.2 Supported Variable Types

The following types are automatically supported (defined in [htmlfiles.h](#)):

```
void WriteHtmlVariable(int fd, char c);
void WriteHtmlVariable(int fd, int i);
void WriteHtmlVariable(int fd, short i);
void WriteHtmlVariable(int fd, long i);
void WriteHtmlVariable(int fd, uint8_t b);
void WriteHtmlVariable(int fd, uint16_t w);
void WriteHtmlVariable(int fd, unsigned long dw);
void WriteHtmlVariable(int fd, const char *);
void WriteHtmlVariable(int fd, MACADDR ma);
void WriteHtmlVariable(int fd, IPADDR ip);
```

10.10.6.3 Use Case 2: Parameterized Function Calls

When you need to pass parameters to a function (beyond the fixed CPPCALL signature), use `VARIABLE()`:

```
HTML: <!--VARIABLE MyFunc(fd, param1, param2) -->
      |
      v
Converted To: WriteHtmlVariable(fd, MyFunc(fd, param1, param2));
      |
      v
      MyFunc Executes and Returns Result
      |
      v
      Result Written to Browser
```

Header Declaration:**

```
const char* DisplaySensorData(int fd, int sensorId);
```

HTML Usage:**

```
Sensor 1: <!--VARIABLE DisplaySensorData(fd, 1) --><br>
Sensor 2: <!--VARIABLE DisplaySensorData(fd, 2) --><br>
```

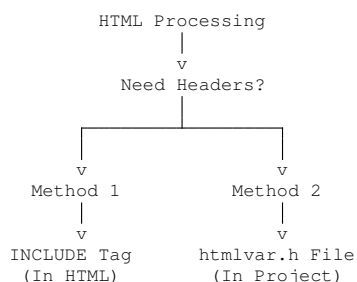
Implementation:**

```
const char* DisplaySensorData(int fd, int sensorId)
{
    int reading = ReadSensor(sensorId);
    fdprintf(fd, "Sensor %d: %d units", sensorId, reading);
    return ""; // Empty string - output already written to fd
}
```

10.10.7 The INCLUDE Tag and htmlvar.h

For the build system to resolve variable and function references in HTML tags, it needs access to appropriate headers.

10.10.7.1 Two Methods for Include Resolution



10.10.7.2 Method 1: INCLUDE Tag in HTML

Place the include directive directly in your HTML file:

```
<html>
<body>
<!--INCLUDE myIncludeFile.h -->
Value = <!--VARIABLE MyVar --><br>
</body>
</html>
```

10.10.7.3 Method 2: htmlvar.h Header File

Create a file named `htmlvar.h` in your project. The build system automatically searches for this file:

```
#ifndef HTMLVARS_H_
#define HTMLVARS_H_

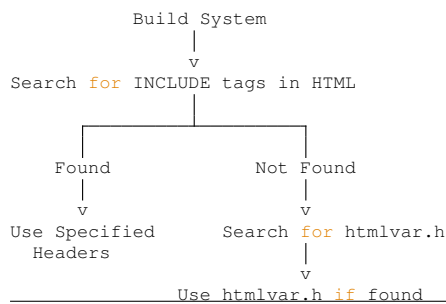
#include <nbrtos.h>      // For TimeTick, TICKS_PER_SECOND

extern int  gIntVal;
extern char gStrVal[];

// Function with parameters for VARIABLE() tag
const char* DisplayWithParam(int fd, int value);

#endif
```

10.10.7.4 Include Resolution Flow



10.10.8 Extending VARIABLE Tag Support

You can extend the `VARIABLE` tag to support custom types by overloading `WriteHtmlVariable()`.

10.10.8.1 Custom Type Example

Define Custom Structure:**

```
struct SensorData {
    int temperature;
    int humidity;
    char location[80];
    uint32_t timestamp;
};
```

SensorData sensor1;

Add to `htmlvar.h`:**

```
void WriteHtmlVariable(int fd, SensorData data);
```

Use in HTML:**

```
Sensor Status: <!--VARIABLE sensor1 -->
```

Implement the Function:**

```
void WriteHtmlVariable(int fd, SensorData data)
{
    fprintf(fd, "<div class='sensor'>");
    fprintf(fd, "  Location: %s<br>", data.location);
    fprintf(fd, "  Temperature: %dC<br>", data.temperature);
    fprintf(fd, "  Humidity: %d%%<br>", data.humidity);
    fprintf(fd, "  Timestamp: %lu", data.timestamp);
    fprintf(fd, "</div>");
}
```

10.10.8.2 System Example: MAC Address Display

Here's how the system implements MAC address display:

```
void WriteHtmlVariable(int fd, MACADDR ma)
{
    PBYTE lpb = (PBYTE)&ma;
    for (int i = 0; i < 5; i++)
    {
        fd_writehex(fd, lpb[i]);
        write(fd, ":", 1);
    }
    fd_writehex(fd, lpb[5]);
}
```

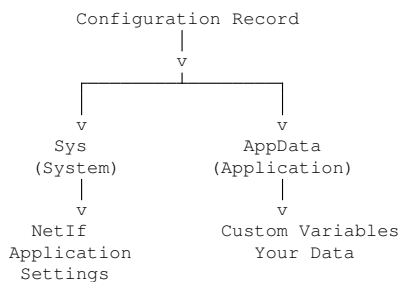
10.10.9 Configuration System Tags

The CONFIG tags simplify displaying and modifying system configuration and application variables from the Configuration Record.

10.10.9.1 Configuration Tag Types

Tag	Function	Creates
CONFIGVALUE	Display value	Read-only text
CONFIGINPUT	Display + edit	HTML input field
CONFIGTABLE	Display hierarchy	HTML table

10.10.9.2 Configuration Tree Structure



10.10.9.3 Example: Display Configuration Values

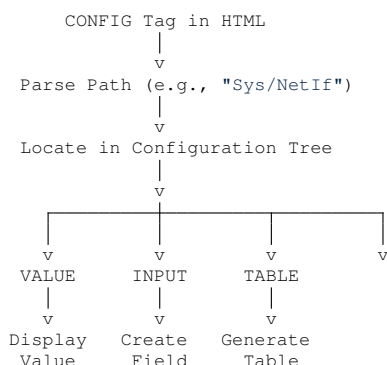
```

<!-- Display system hostname -->
Hostname: <!--CONFIGVALUE Sys/Application/HostName -->

<!-- Create editable input for application variable -->
Device Name: <!--CONFIGINPUT AppData/DeviceName -->

<!-- Display entire configuration section as table -->
<!--CONFIGTABLE Sys/NetIf -->
  
```

10.10.9.4 Configuration Access Flow



10.10.10 HTTP GET Request Handler

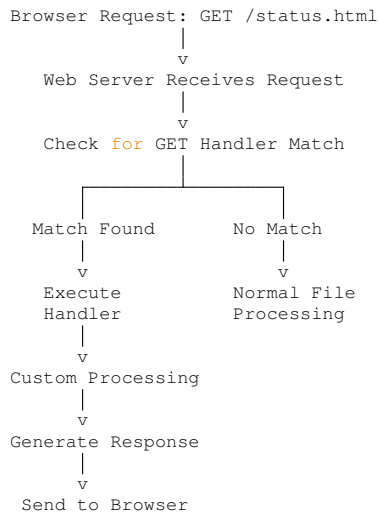
GET handlers intercept browser requests before normal file serving, enabling URL encoding, custom processing, and dynamic routing.

10.10.10.1 When to Use GET Handlers

- URL-encoded parameters: `http://device/status?sensor=1`
- Dynamic page generation based on request
- Stateful applications (session management via URL)

- API endpoints
- Custom routing logic

10.10.10.2 GET Handler Flow



10.10.10.3 Creating a GET Handler

Step 1: Define Callback Function**

```

int callbackGetOddEven(int sock, HTTP_Request &request)
{
    // Send HTML header
    SendHTMLHeader(sock);

    // Custom processing
    printf("GET request from: %I\r\n", request.client_IPaddr);

    // Conditional response based on system time
    if ((Secs % 2) == 0)
        SendFileFragment("even.html", sock);
    else
        SendFileFragment("odd.html", sock);

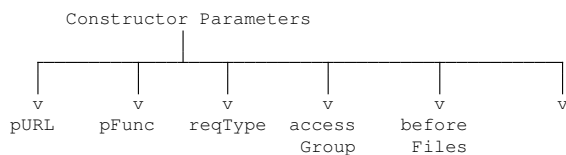
    return 1; // Indicates request was handled
}
  
```

Step 2: Register Handler**

```

CallBackFunctionPageHandler getHandler(
    "OddEven.html",      // URL to intercept
    callbackGetOddEven,  // Callback function
    tGet,                // Request type
    0,                  // Access group (0 = no password)
    true                 // beforeFiles: true = always called
);
  
```

10.10.10.4 CallBackFunctionPageHandler Parameters



Parameter	Description	Example
pURL	URL mask to match (supports wildcards)	"status.html", "api/*"
pFunction	Pointer to callback function	myCallback
reqType	Request type (tGet, tPost, etc.)	tGet
accessGroup	Security group (0 = no restriction)	0
beforeFiles	When to call: true = always, false = only if file not found	true

10.10.10.5 URL Encoding Example

```
// Handler for: http://device/sensor?id=5&type=temp
int sensorHandler(int sock, HTTP_Request &request)
{
    SendHTMLHeader(sock);

    // Parse URL parameters
    const char* id = GetRequestParameter(request, "id");
    const char* type = GetRequestParameter(request, "type");

    fprintf(sock, "<html><body>");
    fprintf(sock, "Sensor ID: %s<br>", id);
    fprintf(sock, "Type: %s<br>", type);

    // Read and display sensor
    int value = ReadSensor(atoi(id));
    fprintf(sock, "Value: %d</body></html>", value);

    return 1;
}

CallbackFunctionPageHandler sensorGet("sensor*", sensorHandler, tGet, 0, true);
```

10.10.11 HTTP Form POST Handler

POST handlers process form submissions, enabling user input, configuration updates, and interactive web applications.

10.10.11.1 Creating a POST Handler

```
HTML Form Creation
    |
    v
POST Handler Callback
    |
    v
Handler Declaration
    |
    v
Processing Complete
```

10.10.11.2 Step 1: Create HTML Form

```
<html>
<title>Configuration Form</title>
<body>
    <h1>Device Configuration</h1>

    <form action="updateconfig" method="post">
        Device Name: <input type="text" name="DeviceName" value="Default"><br>
        IP Address:  <input type="text" name="IPAddress" value="192.168.1.100"><br>
        Port:       <input type="text" name="Port" value="80"><br>
        Enable:     <input type="checkbox" name="Enable" checked><br>
        <br>
        <input type="submit" value="Update Configuration">
    </form>
</body>
</html>
```

10.10.11.3 Step 2: Create Callback Function

```
void configPostCallback(int sock, PostEvents event,
                        const char* pName, const char* pValue)
{
    static char deviceName[80];
    static char ipAddress[20];
    static int port;
    static bool enabled;
    static bool error = false;

    switch(event)
    {
        case eStartingPost:
            // Initialize before processing variables
            deviceName[0] = '\0';
            ipAddress[0] = '\0';
            port = 0;
            enabled = false;
            error = false;
    }
```

```

        break;

    case eVariable:
        // Process each form variable
        if (strcmp("DeviceName", pName) == 0)
        {
            strncpy(deviceName, pValue, 79);
            printf("Device Name: %s\r\n", deviceName);
        }
        else if (strcmp("IPAddress", pName) == 0)
        {
            strncpy(ipAddress, pValue, 19);
            printf("IP Address: %s\r\n", ipAddress);
        }
        else if (strcmp("Port", pName) == 0)
        {
            port = atoi(pValue);
            printf("Port: %d\r\n", port);
        }
        else if (strcmp("Enable", pName) == 0)
        {
            enabled = (strcmp(pValue, "on") == 0);
            printf("Enabled: %s\r\n", enabled ? "true" : "false");
        }
        else
        {
            printf("Unknown variable: %s\r\n", pName);
            error = true;
        }
        break;

    case eFile:
        // Handle file upload (if applicable)
        break;

    case eEndOfPost:
        // Processing complete - send response
        if (error)
        {
            RedirectResponse(sock, "error.html");
        }
        else
        {
            // Apply configuration
            ApplyConfiguration(deviceName, ipAddress, port, enabled);
            RedirectResponse(sock, "success.html");
        }
        break;
}
}

```

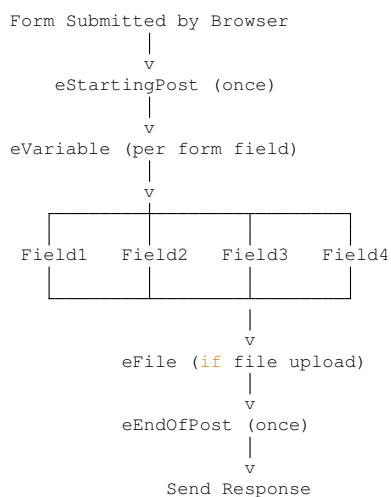
10.10.11.4 Step 3: Register Handler

```

HtmlPostVariableListCallback postHandler(
    "updateconfig*",          // URL mask to match
    configPostCallback        // Callback function
);

```

10.10.11.5 POST Event Flow



10.10.11.6 POST Events

Event	When Called	Purpose
eStartingPost	Once at start	Initialize variables, clear state
eVariable	Once per form field	Process each input value
eFile	If file uploaded	Handle file data
eEndOfPost	Once at end	Finalize, validate, send response

10.10.11.7 Multiple Form Handling

You can use wildcards to handle multiple forms with one callback:

```
void multiFormHandler(int sock, PostEvents event,
                     const char* pName, const char* pValue)
{
    switch(event)
    {
        case eStartingPost:
            // Determine which form based on URL
            break;

            // ... handle events
    }
}

// Handles form1.html, form2.html, form3.html, etc.
HtmlPostVariableListCallback multiPost("form*", multiFormHandler);
```

10.10.12 Complete Examples

10.10.12.1 Example 1: System Status Page with Mixed Tags

```
<!DOCTYPE html>
<html>
<head>
    <title>System Status</title>
    <!--INCLUDE htmlvar.h -->
</head>
<body>
    <h1>NetBurner System Status</h1>

    <h2>System Information</h2>
    <table border="1">
        <tr>
            <td>Uptime (seconds):</td>
            <td><!--VARIABLE Secs --></td>
        </tr>
        <tr>
            <td>Uptime (ticks):</td>
            <td><!--VARIABLE TimeTick --></td>
        </tr>
        <tr>
            <td>MAC Address:</td>
            <td><!--VARIABLE GetMacAddress() --></td>
        </tr>
        <tr>
            <td>Client Info:</td>
            <td><!--CPPCALL webShowClientInfo --></td>
        </tr>
    </table>

    <h2>Configuration</h2>
    Hostname: <!--CONFIGVALUE Sys/Application/HostName --><br>

    <h2>Sensor Readings</h2>
    Temperature: <!--VARIABLE ReadTemperature(fd) -->C<br>
    Pressure: <!--VARIABLE ReadPressure(fd) --> kPa<br>
</body>
</html>
```

10.10.12.2 Example 2: Interactive Control Panel

HTML Form:**

```
<html>
<head><title>Control Panel</title></head>
<body>
    <h1>Device Control Panel</h1>

    <form action="control" method="post">
```

```

<h2>Output Control</h2>
Output 1: <input type="checkbox" name="Out1"><br>
Output 2: <input type="checkbox" name="Out2"><br>
Output 3: <input type="checkbox" name="Out3"><br>

<h2>PWM Settings</h2>
Duty Cycle: <input type="range" name="Duty" min="0" max="100" value="50">
<span id="duty-value">50</span><br>

<br>
<input type="submit" value="Apply Settings">
</form>

<script>
// Update duty cycle display
document.querySelector('input[name="Duty"]').oninput = function() {
    document.getElementById('duty-value').textContent = this.value;
};
</script>
</body>
</html>

```

POST Handler:**

```

void controlPostCallback(int sock, PostEvents event,
                        const char* pName, const char* pValue)
{
    static bool out1 = false, out2 = false, out3 = false;
    static int dutyCycle = 0;

    switch(event)
    {
        case eStartingPost:
            out1 = out2 = out3 = false;
            dutyCycle = 0;
            break;

        case eVariable:
            if (strcmp("Out1", pName) == 0)
                out1 = (strcmp(pValue, "on") == 0);
            else if (strcmp("Out2", pName) == 0)
                out2 = (strcmp(pValue, "on") == 0);
            else if (strcmp("Out3", pName) == 0)
                out3 = (strcmp(pValue, "on") == 0);
            else if (strcmp("Duty", pName) == 0)
                dutyCycle = atoi(pValue);
            break;

        case eEndOfPost:
            // Apply settings to hardware
            SetOutput(1, out1);
            SetOutput(2, out2);
            SetOutput(3, out3);
            SetPWMDutyCycle(dutyCycle);

            printf("Outputs: %d %d %d, Duty: %d%\r\n",
                out1, out2, out3, dutyCycle);

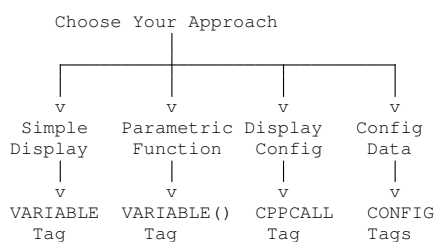
            RedirectResponse(sock, "control.html");
            break;
    }
}

```

```
HtmlPostVariableListCallback controlPost("control*", controlPostCallback);
```

10.10.13 Best Practices

10.10.13.1 Tag Usage Guidelines



Use VARIABLE when:**

- Displaying simple variable values
- Showing calculated expressions

- Output is straightforward text/numbers
Use `VARIABLE()` when:**
- Need to pass parameters to function
- Require conditional formatting
- Complex data processing needed
Use `CPPCALL` when:**
- Need full control over output
- Complex HTML generation
- Multiple operations required
Use `CONFIG` tags when:**
- Working with Configuration Record
- Simplifying configuration UI
- Displaying system settings

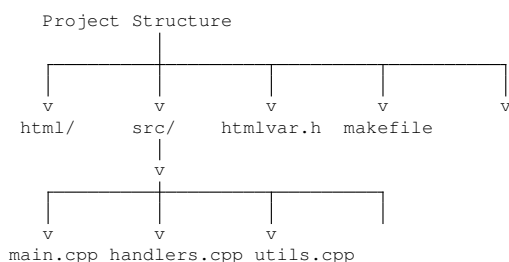
10.10.13.2 Performance Considerations

1. **Minimize Function Calls:** Each tag invokes processing overhead
2. **Buffer Output:** Use efficient I/O operations
3. **Cache Values:** Don't recalculate on every request
4. **Use Appropriate Tags:** `CONFIG` tags are optimized for configuration data

10.10.13.3 Security Guidelines

1. **Validate Input:** Always validate POST data
2. **Sanitize Output:** Prevent injection attacks
3. **Use Access Groups:** Restrict sensitive pages
4. **Implement Timeouts:** Prevent resource exhaustion

10.10.13.4 Code Organization



Recommended:**

- Keep HTML files in `html/` folder
- Centralize handlers in dedicated source files
- Use `htmlvar.h` for all HTML-exposed declarations
- Document callback functions clearly
- Group related handlers together

10.10.14 Troubleshooting

10.10.14.1 Common Issues and Solutions

Issue: CPPCALL function not found**

Symptom: Linker error or runtime error

Solution: Verify function signature exactly matches:
`void FunctionName(int sock, PCSTR url)`

Issue: VARIABLE not displaying**

Symptom: Blank output where variable should appear

Solution:

1. Check `htmlvar.h` includes proper headers
2. Verify variable is `extern` declared
3. Ensure `WriteHtmlVariable()` overload exists for type

Issue: POST variables not received**

Symptom: `eVariable` not called for form fields

Solution:

1. Verify form action matches handler URL mask
2. Check `method="post"` in form tag
3. Ensure input fields have name attributes

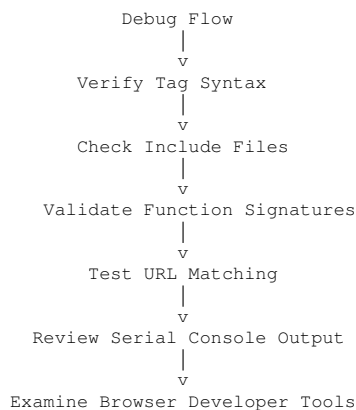
Issue: Spacing in tags**

Symptom: Tag not processed, appears as comment

Solution: Verify spacing:

Correct: `<!--CPPCALL func -->`
 Incorrect: `<!-- CPPCALL func -->`
 Incorrect: `<!--CPPCALL func-->`

10.10.14.2 Debug Checklist



1. Enable serial console debugging with `printf()`
2. Check web browser developer console for errors
3. Verify HTML is syntactically correct
4. Test with minimal examples first
5. Use `RedirectResponse()` for POST debugging
6. Monitor socket operations with diagnostic output

10.10.15 Additional Resources

10.10.15.1 Related Documentation

- **Web Server Examples:** `\nburn\examples\web`
- **Configuration System:** NetBurner Configuration Guide
- **WebSockets:** WebSocket Programming Guide
- **Security:** Authentication and Access Control

10.10.15.2 Key Header Files

```
#include <http.h>           // HTTP server functions
#include <htmlfiles.h>      // WriteHtmlVariable() overloads
#include <nbrtos.h>         // Timing, system functions
#include <netinterface.h>   // Network interface utilities
```

10.10.15.3 Quick Reference

Tag Syntax:**

```
<!--CPPCALL functionName -->
<!--VARIABLE expression -->
<!--VARIABLE function(fd, params) -->
<!--CONFIGVALUE path/to/variable -->
<!--CONFIGINPUT path/to/variable -->
```

Function Signatures:**

```
// CPPCALL
void FunctionName(int sock, PCSTR url);

// VARIABLE()
const char* FunctionName(int fd, int param);

// GET Handler
int HandlerName(int sock, HTTP_Request &request);

// POST Handler
void HandlerName(int sock, PostEvents event,
                 const char* pName, const char* pValue);
```

Declaration Templates:**

```
// GET Handler
CallbackFunctionPageHandler name("url", callback, tGet, 0, true);

// POST Handler
HtmlPostVariableListCallback name("url*", callback);
```

10.10.16 Summary

NetBurner's HTML processing system provides a comprehensive toolkit for building dynamic web applications on embedded devices:

Core Capabilities:**

- Seamless integration of C/C++ code with HTML
- Real-time variable display and function execution
- Form processing and user input handling
- Custom request routing and URL encoding
- Configuration system integration

Development Workflow:**

```
Design HTML Interface
    |
    v
Add Dynamic Tags
    |
    v
Implement Callbacks
    |
    v
Register Handlers
    |
    v
Build and Deploy
    |
    v
Test and Iterate
```

Key Takeaways:**

- Start with simple VARIABLE and CPPCALL tags
- Progress to GET/POST handlers for complex interactions
- Use CONFIG tags for configuration management
- Follow best practices for security and performance

- Leverage examples and documentation

The NetBurner HTML processing system eliminates the complexity of traditional CGI or server-side scripting while providing powerful dynamic content capabilities optimized for embedded systems.

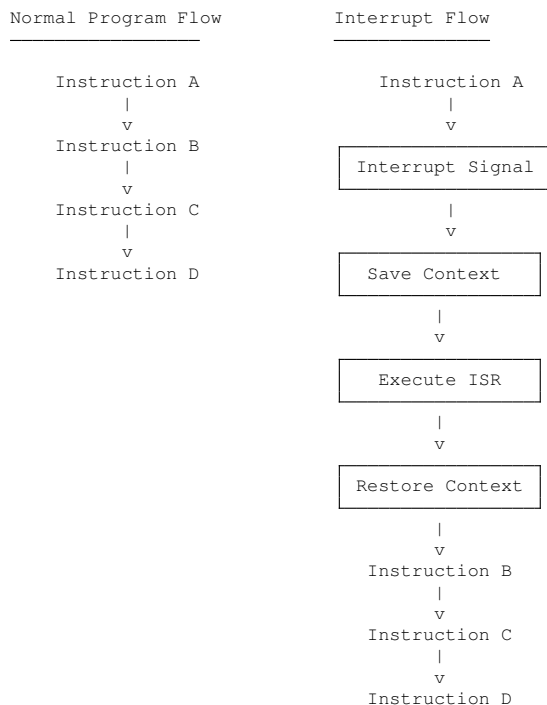
10.11 Interrupt Handling

10.11.1 interrupt_handling Interrupt Handling in NetBurner Applications

10.11.2 interrupt_overview Overview

An **interrupt** is a hardware or software signal that temporarily suspends normal program execution to handle a time-critical event. When an interrupt occurs, the processor:

1. Saves the current execution context
2. Executes the Interrupt Service Routine (ISR)
3. Restores the context and resumes normal execution



10.11.2.1 interrupt_sources Common Interrupt Sources

NetBurner devices handle interrupts from various sources:

- **Hardware Timers** - Periodic or one-shot timing events
- **Serial Ports** - UART data reception/transmission
- **Ethernet Controllers** - Network packet events
- **GPIO Pins** - External hardware signals
- **Software Traps** - Programmatically triggered interrupts

10.11.2.2 interrupt_platform_variations Platform Variations

Interrupt handling implementation varies by processor architecture:

Platform	Complexity	Key Characteristics
ARM (Cortex-M7)	Simple	Direct function calls, minimal setup
ColdFire (MCF5441x)	Complex	Requires entry/exit macros, controller setup

NetBurner provides abstraction layers to simplify interrupt configuration across all platforms.

10.11.3 core_interrupt_functions Core Interrupt Functions

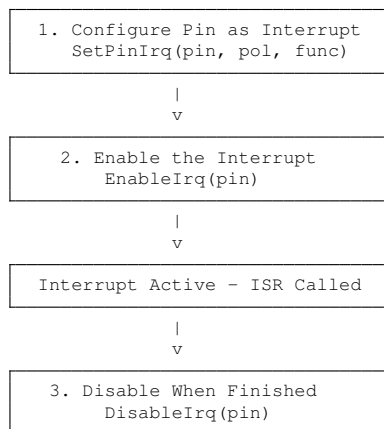
The following functions work across **all NetBurner platforms** and provide a consistent interface for interrupt management.

10.11.3.1 interrupt_header_requirements Header Requirements

```
#include <pins.h>           // For PinIO objects
#include <system.h>         // For interrupt functions
```

10.11.3.2 interrupt_function_overview Function Overview

Interrupt Configuration Workflow



10.11.3.3 interrupt_setpinirq SetPinIrq() - Configure Interrupt Pin

Function Signature:**

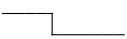
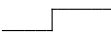
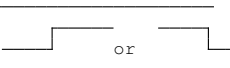
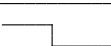
```
bool SetPinIrq(PinIO pin, int polarity, PinIrq_t func);
```

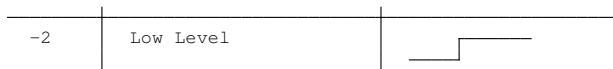
Parameters:**

Parameter	Type	Description
pin	PinIO	Pin object to configure as interrupt source
polarity	int	Trigger condition (see polarity table below)
func	PinIrq_t	Function pointer to the ISR

Polarity Values:**

Signal Polarity Configuration

Value	Trigger Type	Signal Diagram
2	High Level	
1	Rising Edge (Pos)	
0	Any Edge (Both)	
-1	Falling Edge (Neg)	



Returns:** true if configuration successful, false otherwise.

Example:**

```
// ISR function
void MyButtonISR() {
    // Handle button press
    printf("Button pressed!\r\n");
}

// Configure pin 23 for falling edge interrupt
PinIO buttonPin(23);
buttonPin.function(PINMODE_GPIO);
SetPinIrq(buttonPin, -1, MyButtonISR);
```

10.11.3.4 interrupt_enableirq EnableIrq() - Activate Interrupt

Function Signature:**

```
void EnableIrq(PinIO pin);
```

Enables interrupt processing for the specified pin.

Example:**

```
EnableIrq(buttonPin); // Start receiving interrupts
```

10.11.3.5 interrupt_disableirq DisableIrq() - Deactivate Interrupt

Function Signature:**

```
void DisableIrq(PinIO pin);
```

Disables interrupt processing for the specified pin.

Example:**

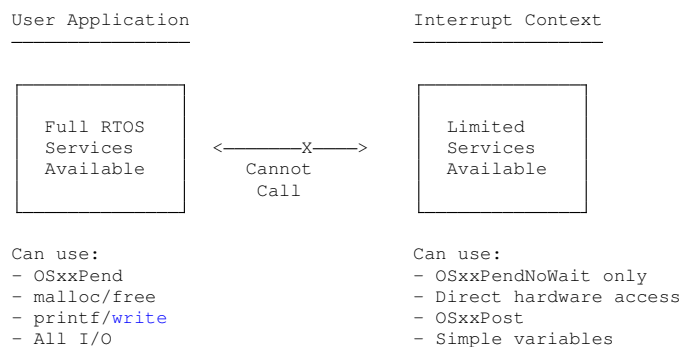
```
DisableIrq(buttonPin); // Stop receiving interrupts
```

10.11.4 isr_restrictions ISR Restrictions

10.11.4.1 isr_critical_constraints Critical Constraints

The code within an ISR executes in an **interrupt context**, which has strict limitations. The following operations are **PROHIBITED** in an ISR:

ISR Execution Context



10.11.4.2 isr_prohibited_rtos Prohibited RTOS Functions

All blocking/waiting functions are forbidden:**

```
// NEVER call these in an ISR:
OSMboxPend() // Mailbox wait
OSSemPend() // Semaphore wait
OSQPend() // Queue wait
OSCritEnter() // Critical section entry
OSChangePrio() // Priority change
OSTaskDelete() // Task deletion
OSLock() // Scheduler lock
OSUnlock() // Scheduler unlock
OSTaskCreate() // Task creation
OSTimeDly() // Time delay
```

Allowed RTOS operations (non-blocking only):**

```
// Safe to call in ISR:
OSMboxPendNoWait() // Non-blocking mailbox check
OSSemPendNoWait() // Non-blocking semaphore check
```

```

OSQPendNoWait()    // Non-blocking queue check
OSMboxPost()       // Post to mailbox
OSSemPost()        // Post to semaphore
OSQPost()          // Post to queue

```

10.11.4.3 isr_prohibited_io Prohibited I/O Functions

All I/O operations that may block are forbidden:**

```

// NEVER call these in an ISR:
write()           // File/socket write
writeall()        // Complete write
read()            // File/socket read
printf()          // Formatted output
fprintf()         // File formatted output
fdprintf()        // Descriptor formatted output
printf()          // Integer formatted output
scanf()           // Formatted input
gets()            // Line input
puts()            // String output

```

10.11.4.4 isr_prohibited_memory Prohibited Memory Functions

Dynamic memory operations are forbidden:**

```

// NEVER call these in an ISR:
malloc()          // Allocate memory
free()            // Free memory
new               // C++ allocation
delete            // C++ deallocation

```

10.11.4.5 isr_recommended_pattern Recommended ISR Pattern

```

// Correct ISR Implementation
volatile bool eventFlag = false;
volatile uint32_t eventData = 0;

// ISR: Minimal processing, set flag
void MyISR() {
    // Read hardware register
    eventData = ReadHardwareRegister();

    // Set flag for main loop
    eventFlag = true;

    // Optional: Signal task via semaphore
    OSSemPost(&processingTask);
}

// Main task: Handle event
void ProcessingTask(void* pd) {
    while (1) {
        OSSemPend(&processingTask, 0);

        if (eventFlag) {
            eventFlag = false;

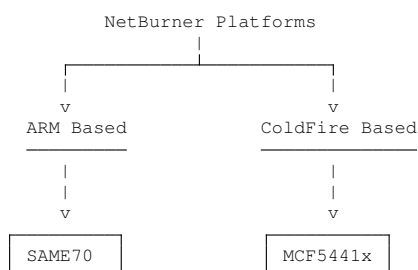
            // Now safe to use all functions
            ProcessEvent(eventData);
            printf("Event processed: %lu\r\n", eventData);
        }
    }
}

```

10.11.5 platform_specific_implementations Platform-Specific Implementations

Different processor architectures require different interrupt setup procedures.

Platform Interrupt Architecture



 v	 v
Simple Config	Complex Config
- Direct ISR	- INTERRUPT macro
- Port priority	- SetIntc() setup
- PinIO based	- Vector tables

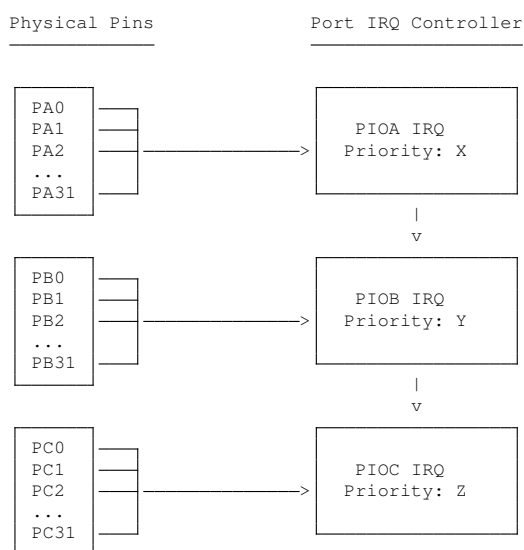
10.11.6 same70_platforms SAME70 Platforms (MODM7AE70 and SBE70LC)

10.11.6.1 same70_architecture Architecture Overview

The SAME70 (Cortex-M7) features:

- **Any GPIO pin** can be used as an interrupt source
- **Port-based priority** - all pins on same port share priority
- **7 Priority Levels** - 1 (highest) to 7 (lowest)

SAME70 Interrupt Structure



10.11.6.2 same70_port_priority Port Priority Configuration

10.11.6.2.1 SetPortIrqPriority() - By PinIO Object

Function Signature:**

```
bool SetPortIrqPriority(PinIO pin, int priority);
```

Parameters:**

Parameter	Type	Description
pin	PinIO	Any pin on the target port
priority	int	Priority level: 1 (highest) to 7 (lowest)

Important:** Setting priority for any pin affects the **entire port**.

Returns:** true if successful, false otherwise.

Example:**

```
PinIO myPin(23); // Pin on PIOB
SetPortIrqPriority(myPin, 3); // All PIOB interrupts now priority 3
```

10.11.6.2.2 SetPortIrqPriority() - By Port Number

Function Signature:**

```
bool SetPortIrqPriority(int portNum, int priority);
```

Parameters:**

Parameter	Type	Description
portNum	int	Port number: PIOA=0, PIOB=1, PIOC=2, PIOD=3, PIOE=4
priority	int	Priority level: 1 (highest) to 7 (lowest)

Example:**

```
SetPortIrqPriority(1, 3); // Set PIOB to priority 3
```

10.11.6.3 same70_port_mapping Port Priority Mapping

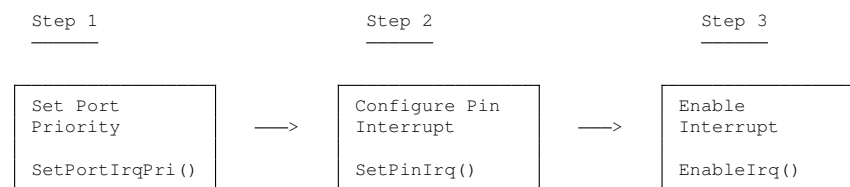
SAME70 Port Assignments

Port Number	Port Name	Example Usage
0	PIOA	High priority sensors
1	PIOB	Medium priority inputs
2	PIOC	Communication signals
3	PIOD	General I/O
4	PIOE	Low priority monitoring

10.11.6.4 same70_workflow Configuration Workflow

Critical:** Always set port priority **before** configuring pin interrupts.

SAME70 Interrupt Setup Sequence



10.11.6.5 same70_example Complete SAME70 Example

```
#include <pins.h>
#include <system.h>
#include <nbrtos.h>

// ISR for external sensor
void SensorISR() {
    // Minimal processing in ISR
    OSemPost(&sensorEvent);
}

void ConfigureSensorInterrupt() {
    // Pin 42 is on PIOB (port 1)
    PinIO sensorPin(42);
    sensorPin.function(PINMODE_GPIO);

    // Step 1: Set port priority (all PIOB pins)
    SetPortIrqPriority(1, 2); // High priority (2)

    // Step 2: Configure pin interrupt
    SetPinIrq(sensorPin, 1, SensorISR); // Rising edge trigger

    // Step 3: Enable interrupt
    EnableIrq(sensorPin);

    printf("Sensor interrupt configured on pin 42\r\n");
}
```

10.11.7 mcf5441x_platforms MCF5441x Platforms (MOD54415, MOD54417, NANO54415 and SB800EX)

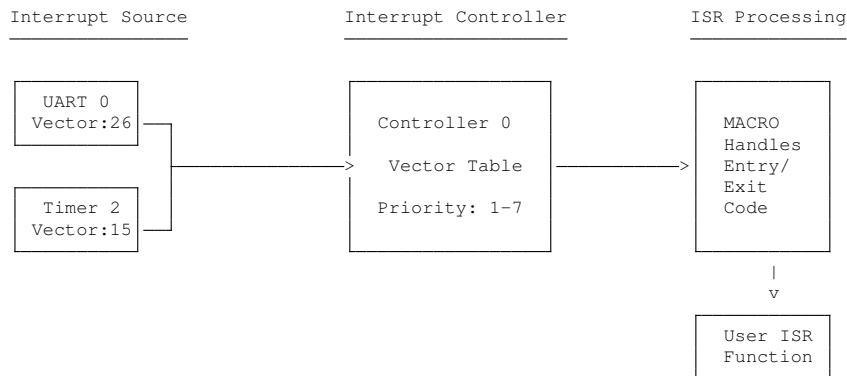
10.11.7.1 mcf5441x_architecture Architecture Overview

The MCF5441x (ColdFire) processor requires more complex setup:

- **INTERRUPT() macro** - Handles ISR entry/exit code
- **SetIntc() function** - Configures interrupt controller hardware

- **7 Priority Levels** - 1 to 7, with 7 being highest (unmaskable)

ColdFire Interrupt Flow



10.11.7.2 mcf5441x_interrupt_macro INTERRUPT() Macro

The INTERRUPT() macro simplifies ISR implementation by handling:

- Saving processor registers
- Setting status register
- RTOS notification
- Restoring registers on exit

10.11.7.2.1 Header Requirements

```
#include <nbrtos.h>      // RTOS functions
#include <cfinter.h>     // ColdFire interrupt support
```

10.11.7.2.2 Macro Syntax

```
INTERRUPT(FunctionName, SR_Value)
{
    // ISR code here
}
```

10.11.7.2.3 Parameters

Parameter	Description
FunctionName	ISR function name (used with SetIntc())
SR_Value	Status Register value - controls interrupt masking

10.11.7.2.4 Status Register Values

The Status Register (SR) controls which interrupts can preempt the current ISR:

ColdFire Status Register Interrupt Masking

SR Value	Mask Level	Allows Interrupts At Level	Blocks
0x2000	0	All (1-7)	None
0x2100	1	2 and above	Level 1
0x2200	2	3 and above	Levels 1-2
0x2300	3	4 and above	Levels 1-3
0x2400	4	5 and above	Levels 1-4
0x2500	5	6 and above	Levels 1-5
0x2600	6	7 only	Levels 1-6
0x2700	7	None (masks all maskable)	All < 7

Note: Level 7 interrupts are unmaskable - they always execute
 Bit 29 is supervisor bit (always set = 0x2000 base)

10.11.7.2.5 Interrupt Priority Hierarchy

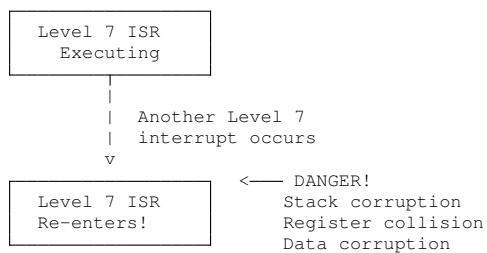
Interrupt Priority Levels

Level 7	—————	Highest (Unmaskable) Critical system events Can interrupt anything
^		
Level 6	—————	Very High Priority Time-critical operations
^		
Level 5	—————	High Priority Important peripherals
^		
Level 4	—————	Medium-High Priority Communication interfaces
^		
Level 3	—————	Medium Priority General peripherals
^		
Level 2	—————	Low-Medium Priority Non-critical I/O
^		
Level 1	—————	Low Priority Background tasks
^		
Level 0	—————	Disabled

10.11.7.2.6 Level 7 Warning

CRITICAL:** Level 7 interrupts are **unmaskable** and can interrupt themselves!

Level 7 Interrupt Danger



Mitigation Strategy:**

```

// Ensure ISR completes before next Level 7 can occur
__interrupt(CriticalISR, 0x2700)
{
    // 1. Acknowledge interrupt source immediately
    ClearInterruptFlag();

    // 2. Minimal processing only
    volatile uint32_t data = ReadHardwareRegister();

    // 3. Set flag for main loop processing
    criticalEventFlag = true;

    // Exit quickly - no delays!
}
  
```

10.11.7.3 mcf5441x_setintc SetIntc() Function

Configures the interrupt controller to route hardware interrupts to ISR functions.

10.11.7.3.1 Function Declaration

Important:** SetIntc() is defined in platform HAL files and must be declared as extern:

```
extern void SetIntc(int intcnum, long func, int vector, int level);
```

Location:** \nburn\platform\MOD5441x\hal.cpp

10.11.7.3.2 Parameters

Parameter	Type	Description
intcnum	int	Interrupt controller number (0 or 1 for MCF5441x)

Parameter	Type	Description
func	long	Pointer to ISR function (from INTERRUPT macro)
vector	int	Interrupt vector number (0-255, device-specific)
level	int	Interrupt priority level (0=disabled, 1-7)

10.11.7.3.3 Vector Numbers

The MCF5441x has 256 exception vectors:

- **0-63:** Core exceptions (reserved)
- **64-255:** Peripheral interrupt sources (192 device-specific vectors)

Common MCF5441x Interrupt Sources

Source	Vector	Typical Use
UART 0	26	Serial communication
UART 1	27	Serial communication
PIT Timer 2	15	Periodic timer
1-Wire	63	1-Wire bus interface
Ethernet	varies	Network events

Reference:** See Chapter 17, Table 17.2.9.1 in NXP MCF5441x Reference Manual

10.11.7.3.4 Interrupt Controller Selection

MCF5441x Interrupt Controllers

Controller 0 (intcnum = 0)	Controller 1 (intcnum = 1)
- UART 0 (Vector 26) - PIT Timers - Ethernet 1-Wire (Vector 63) - Core peripherals	- Additional peripherals - Extended I/O - Communication interfaces - Platform-specific devices

10.11.7.4 mcf5441x_example Complete MCF5441x Example: 1-Wire Interface

```
#include <nbrtos.h>
#include <cfinter.h>
#include <sim5441x.h>

// Declare external SetIntc function
extern void SetIntc(int intc, long func, int vector, int level);

// State tracking
volatile enum { OK, BUSY, ERROR } ow_state = OK;

// Actual ISR implementation
void OWMasterIsr()
{
    // Read 1-Wire status
    uint8_t status = siml.ow.sr;

    // Clear interrupt flag
    siml.ow.sr = status;

    // Update state
    if (status & OW_SR_RBF) {
        // Receive buffer full
        ow_state = OK;
    }
}

// INTERRUPT macro wrapper - sets SR to mask interrupts below level 6
INTERRUPT(ow_master_int_routine, 0x2600)
{
    OWMasterIsr();
}

// Initialization function
void OWInit()
{
    // Configure 1-Wire hardware registers
    siml.ow.cr = 0;           // Control register
    siml.ow.div = 0x7C;       // Divide peripheral clock to 1 MHz
}
```

```

siml.ow.cmd = 0;           // Command register
siml.ow.ier = OW_IER_ERBF; // Enable receive buffer full interrupt

// Configure interrupt controller
// Controller 0, ISR function, Vector 63, Level 3
SetIntc(0, (long)&ow_master_int_routine, 63, 3);

// Initialize state
ow_state = OK;

printf("1-Wire interrupt configured: Vector 63, Level 3\r\n");
}

// Usage in UserMain()
void UserMain(void *pd)
{
    init();

    // Initialize 1-Wire with interrupt support
    OWInit();

    while (1) {
        // Application code
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

10.11.7.5 mcf5441x_workflow MCF5441x Configuration Workflow

ColdFire Interrupt Setup Process

Step 1: Implement ISR Logic

```

void MyIsr() {
    // Handle interrupt
}

```

|
v

Step 2: Wrap with INTERRUPT Macro

```

INTERRUPT(my_isr, 0x2500) {
    MyIsr();
}

```

|
v

Step 3: Configure Hardware

```

// Set registers
peripheral.control = ...;
peripheral.enable = ...;

```

|
v

Step 4: Configure Controller

```

SetIntc(0,
(long)&my_isr,
vector_num,
priority_level);

```

|
v

Step 5: Interrupt Active

```

ISR Called on Events

```

10.11.7.6 mcf5441x_best_practices Best Practices for MCF5441x

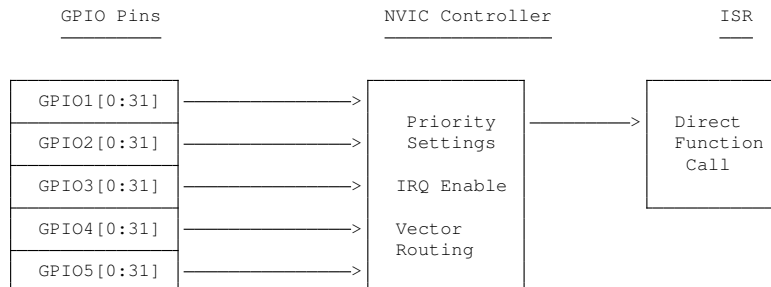
1. **Always use INTERRUPT() macro** - Never write bare ISR functions
2. **Match SR value to needs** - Don't block higher priority interrupts unnecessarily
3. **Initialize hardware first** - Set up peripheral registers before SetIntc()
4. **Use correct vector numbers** - Consult reference manual for device-specific vectors
5. **Avoid Level 7 unless critical** - Use lower levels when possible

10.11.8 somrt1061_platform SOMRT1061 Platform

10.11.8.1 somrt1061_overview Overview

The SOMRT1061 platform uses the i.MX RT1061 processor (ARM Cortex-M7). Interrupt configuration follows the standard NetBurner approach with some platform-specific considerations.

i.MX RT1061 Interrupt Architecture



10.11.8.2 somrt1061_configuration Configuration Approach

The SOMRT1061 uses the **standard NetBurner interrupt functions**:

```
#include <pins.h>
#include <system.h>

// ISR function
void MyRT1061ISR() {
    // Handle interrupt
}

// Configuration
void ConfigureInterrupt() {
    PinIO myPin(23);
    myPin.function(PINMODE_GPIO);

    // Standard functions work directly
    SetPinIrq(myPin, 1, MyRT1061ISR); // Rising edge
    EnableIrq(myPin);
}
```

10.11.8.3 somrt1061_notes Platform-Specific Notes

1. **No port-based priority** - Each GPIO can have individual priority
2. **Direct ISR calls** - No INTERRUPT() macro needed
3. **Fast interrupt response** - Cortex-M7 architecture provides low latency
4. **Standard PinIO support** - All GPIO pins accessible via PinIO objects

10.11.8.4 somrt1061_example SOMRT1061 Example

```
#include <init.h>
#include <nbrtos.h>
#include <pins.h>
#include <system.h>

const char *AppName = "RT1061 Interrupt Example";

// Semaphore for signaling
OS_SEM buttonSem;

// Simple ISR - post semaphore
void ButtonPressISR() {
    OSSemPost(&buttonSem);
}

// Processing task
void ButtonTask(void *pd) {
    while (1) {
        OSSemPend(&buttonSem, 0);
        printf("Button pressed at %lu seconds\r\n", Secs);
    }
}
```

```

}

void UserMain(void *pd) {
    init();

    // Initialize semaphore
    OSemInit(&buttonSem, 0);

    // Configure button on pin 23
    PinIO buttonPin(23);
    buttonPin.function(PINMODE_GPIO);

    // Setup interrupt for falling edge (button press)
    SetPinIrq(buttonPin, -1, ButtonPressISR);
    EnableIrq(buttonPin);

    // Create processing task
    OSTaskCreate(ButtonTask,
        NULL,
        (void *)&ButtonTaskStack[USER_TASK_STK_SIZE],
        (void *)&ButtonTaskStack,
        MAIN_PRIO - 1);

    printf("%s: Interrupt configured on pin 23\r\n", AppName);

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

10.11.9 interrupt_summary Summary and Best Practices

10.11.9.1 interrupt_cross_platform Cross-Platform Compatibility

For maximum portability across NetBurner platforms:

```

// Always use these standard functions:
SetPinIrq(pin, polarity, isr_func);    // Configure
EnableIrq(pin);                       // Enable
DisableIrq(pin);                      // Disable

```

10.11.9.2 interrupt_platform_selection Platform Selection Guide

Choose Your Approach Based on Platform

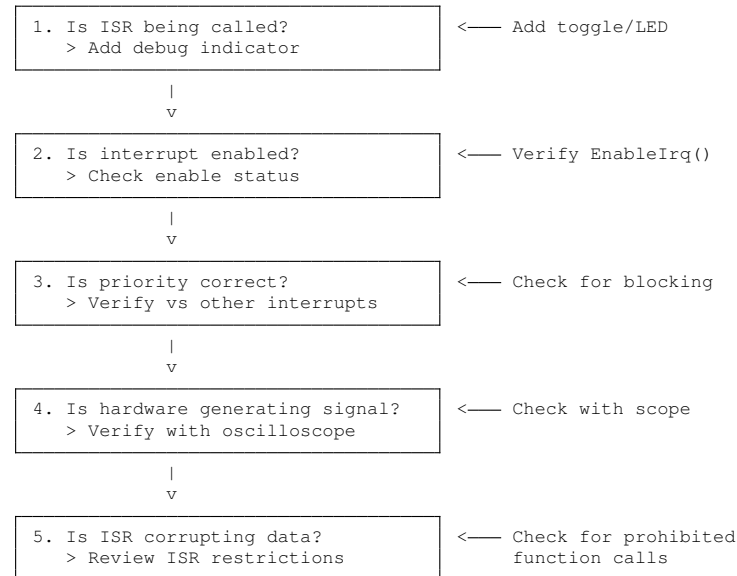
Platform	Complexity	Key Functions
SAME70 (MODM7AE70)	Simple	SetPinIrq() SetPortIrqPriority() EnableIrq() / DisableIrq()
MCF5441x (MOD54415, etc)	Complex	INTERRUPT() macro SetIntc() Manual register setup
SOMRT1061 (i.MX RT1061)	Simple	SetPinIrq() EnableIrq() / DisableIrq()

10.11.9.3 interrupt_general_best_practices General Best Practices

1. **Keep ISRs short** - Minimal processing, set flags, post semaphores
2. **Use volatile for shared variables** - Prevent compiler optimization issues
3. **Never block in ISRs** - No waiting, no delays, no blocking I/O
4. **Signal tasks for processing** - Use semaphores/mailboxes to defer work
5. **Test interrupt rates** - Ensure ISR completes before next interrupt
6. **Document vector/priority choices** - Make interrupt configuration clear
7. **Initialize hardware before interrupts** - Configure peripherals, then enable IRQ

10.11.9.4 interrupt_debugging Debugging Interrupts

Interrupt Debugging Checklist



10.11.9.5 interrupt_error_prevention Error Prevention

```
// WRONG - Blocking in ISR
void BadISR() {
    OSTimeDly(TICKS_PER_SECOND); // NEVER!
    printf("Interrupt!\r\n");      // NEVER!
}

// CORRECT - Signal and exit
void GoodISR() {
    interruptFlag = true;
    OSSemPost(&processSem);      // Non-blocking post
}
```

10.11.10 interrupt_resources Additional Resources

For more detailed information:

- **Platform Hardware Manuals:** \nburn\docs\nxp\
- **ColdFire Interrupt Header:** \nburn\arch\coldfire\include\cfinter.h
- **NetBurner API Documentation:** SDK API reference
- **Example Projects:** NetBurner SDK examples directory

10.12 JSON Lexer

10.12.1 JSON Lexer Programming Guide

10.12.2 Overview

The JSON Lexer provides a powerful interface for parsing and accessing JSON data in NetBurner applications. This guide covers version 3.3.9 and later, which includes significant improvements over earlier versions.

10.12.2.1 Core Components

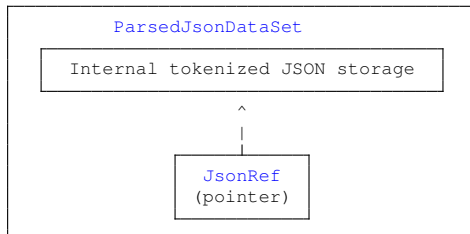
The JSON Lexer interface is defined in `json_lexer.h` and provides two primary object types:

10.12.2.1.1 ParsedJsonDataSet

Represents a complete JSON object that has been parsed and stored internally. This object contains the entire tokenized JSON structure in an optimized format using NetBurner poolBuffers.

10.12.2.1.2 JsonRef

A reference object that acts as a pointer to a location within a [ParsedJsonDataSet](#). It simplifies navigation through complex JSON hierarchies by providing a clean, intuitive interface for accessing nested data.



10.12.2.2 Version Notes

The interface introduced in version 3.3.9 supersedes the previous API. While legacy code continues to work, all new development should use the JsonRef-based interface described in this guide for improved clarity and ease of use.

10.12.3 Loading JSON Data

There are three primary methods to populate a [ParsedJsonDataSet](#) object:

10.12.3.1 Method 1: From Text Blob

Load JSON directly from a text string or character array.

```
// Example from: \nburn\examples\JsonLexer\ParseTest
const char* jdata = "{\"key\": \"value\"}";
size_t jlen = strlen(jdata);
ParsedJsonDataSet pjd((const char *)jdata, jlen, true);
```

10.12.3.2 Method 2: From Web Server

Retrieve JSON from an external HTTP endpoint.

```
// Examples in:
// \nburn\examples\JSON\GetJsonFromServer
// \nburn\examples\JSON\PostAndGetJsonExternalServer
// \nburn\examples\WebClient\EarthQuake

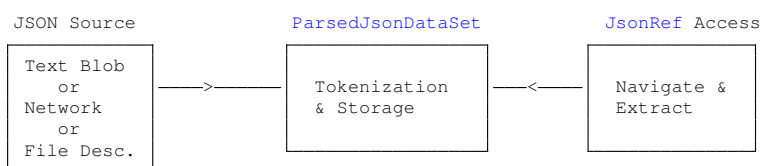
ParsedJsonDataSet pjds;
// Use web client functions to fetch data into pjds
```

10.12.3.3 Method 3: From File Descriptor

Read JSON from any file descriptor source (web socket, serial port, etc.).

```
ParsedJsonDataSet pjds;           // Declare the object
pjds.ReadFrom(int fd);           // Read from file descriptor
```

10.12.4 Data Flow Diagram



10.12.5 Accessing JSON Data

10.12.5.1 Example JSON Structure

Consider the following JSON data:

```
{
  "Anumber": 1234,
  "AString": "ImaString",
  "AnObject": {
    "Item1": 1,
    "Item2": "TheItem2"
  },
  "AnArray": [1, 2, 3, 4]
}
```

10.12.5.2 Basic Access Patterns

10.12.5.2.1 Simple Values

```
// Extract integer
int i = pjds("Anumber");

// Extract strings (two methods)
NBString s = pjds("AString");           // NBString object
const char* pStr = pjds("AString");     // Raw pointer (valid until dataset destroyed)
```

10.12.5.2.2 Nested Objects

Use [JsonRef](#) to simplify access to nested structures:

```
// Create a reference to the nested object
JsonRef jr = pjds("AnObject");

// Access nested values through the reference
int i1 = jr("Item1");
NBString s2 = jr("Item2");
```

10.12.5.2.3 Array Access

Use array indexing to access elements:

```
// Iterate through array elements
for(int i = 0; i < 4; i++) {
    printf("Item[%d] = %d\n", i, pjds("AnArray")[i]);
}
```

10.12.6 Access Operator Summary

The [ParsedJsonDataSet](#) and [JsonRef](#) objects use the following notation:

- ("name") - Find element by name
- [index] - Access array element by index

10.12.6.1 Navigation Example

```
pjds
├── ("Anumber") → int value
├── ("AString") → string value
├── ("AnObject") → JsonRef
│   ├── ("Item1") → int value
│   └── ("Item2") → string value
└── ("AnArray")
    ├── [0] → value
    ├── [1] → value
    ├── [2] → value
    └── [3] → value
```

10.12.7 Type Conversion Operators

The [ParsedJsonDataSet](#) supports automatic type conversion through operator overloading:

```
operator bool()      const { return PermissiveCurrentBool(); }
operator float()     const { return (float)CurrentNumber(); }
operator double()    const { return CurrentNumber(); }
operator uint8_t()   const { return (uint8_t)CurrentNumber(); }
operator int()       const { return (int)CurrentNumber(); }
operator uint16_t()  const { return (uint16_t)CurrentNumber(); }
operator uint32_t()  const { return (uint32_t)CurrentNumber(); }
operator int8_t()    const { return (int8_t)CurrentNumber(); }
operator int16_t()   const { return (int16_t)CurrentNumber(); }
operator int32_t()   const { return (uint32_t)CurrentNumber(); }
operator time_t()    const { return (time_t)CurrentNumber(); }
operator const char*() const { return CurrentString(); }
operator NBString()  const { return (NBString)CurrentString(); }
```

10.12.8 Validation Functions

10.12.8.1 IsValid() Method

Verify that a [JsonRef](#) points to valid data:

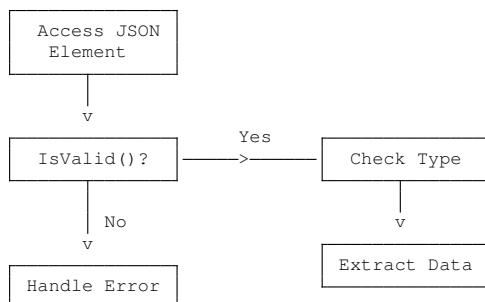
```
pjds("AnArray")[3].IsValid(); // returns true (valid element)
pjds("AnArray")[4].IsValid(); // returns false (out of bounds)
```

10.12.8.2 Type Checking Functions

Determine the type of JSON data:

```
IsNumber() // Check if element is a number
IsObject() // Check if element is an object
IsString() // Check if element is a string
IsBool()   // Check if element is boolean
IsNull()   // Check if element is null
IsArray()  // Check if element is an array
```

10.12.8.3 Validation Flow



10.12.9 Example References

Complete working examples demonstrating these concepts:

- **Basic Parsing:** `\nburn\examples\JsonLexer\ParseTest`
- **Complex Parsing:** `\nburn\examples\webclient\Earthquake`

The Earthquake example demonstrates parsing a large, complex JSON structure from a live web service and extracting specific fields efficiently.

10.12.9.1 Earthquake Example Application

10.12.9.1.1 Overview

This example retrieves earthquake data from the USGS API and parses the complex JSON response to display recent seismic activity.

10.12.9.2 Source Code

```
#include <init.h>
#include <stdio.h>
#include <ctype.h>
#include <buffers.h>
#include <json_lexer.h>
#include <webclient/http_funcs.h>
#include <nbttime.h>

const char* url = "https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/4.5_day.geojson";
const char* AppName = "EarthQuake";

void UserMain(void* pd)
{
    init(); // Initialize network stack
    EnableSystemDiagnostics(); // Enable diagnostics
    WaitForActiveNetwork(TICKS_PER_SECOND * 5); // Wait for DHCP address

    printf("Application started\n");

    bool bTimeSet = SetTimeNTPFromPool();

    if(bTimeSet) {
        printf("Time Set\r\n");
    }

    ParsedJsonDataSet JsonResult;
    bool result = DoGet(url, JsonResult);

    if (result)
    {
        // Get earthquake count from metadata
        int32_t nQuakes = JsonResult("metadata")("count");
        printf("In the last Day we have had %ld Earthquakes > 4.5\r\n", nQuakes);

        // Iterate through earthquake events
        for (int i = 0; i < nQuakes; i++)
        {
            // Create a reference to properties for this earthquake
            JsonRef OneQuakeProperties = JsonResult("features")[i]("properties");

            if (OneQuakeProperties.Valid())
            {
                double magnitude = OneQuakeProperties("mag");
                NBString place_name = OneQuakeProperties("place");
                printf("Magnitude :%2.1f at %s", magnitude, place_name.c_str());

                if (bTimeSet)
                {
                    time_t when = OneQuakeProperties("time");
                    when /= 1000;
                    time_t now = time(NULL);
                    time_t delta = (now - when);
                    int sec = delta % 60;
                    int mins = (delta / 60) % 60;
                    int hour = (delta / 3600);
                    printf(" %02d:%02d:%02d ago", hour, mins, sec);
                }
                printf("\r\n");
            }
            else
            {
                printf("Parse error failed to find array element %d\r\n", i);
            }
        }
    }
    else
    {
        printf("Failed to contact server\r\n");
    }

    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND * 1);
    }
}
```

10.12.9.3 JSON Structure Navigation

The earthquake data follows this hierarchy:

```
root
├── ("metadata")
│   └── ("count") → number of earthquakes
```

```

└─> ("features") ──> array
    └─> [i] ──> array element
        └─> ("properties")
            └─> ("mag") ──> magnitude
            └─> ("place") ──> location string
            └─> ("time") ──> timestamp

```

10.12.9.4 Data Extraction Flow

```

DoGet(url)
  |
  v
ParsedJsonDataSet
└─> metadata
    └─> count ──────────> Loop Control
└─> features[i]
    └─> properties
        └─> mag ─────────> Display
        └─> place ──────> Display
        └─> time ───────> Calculate & Display

```

10.12.9.5 Example Output

```

Application started
Time Set
In the last Day we have had 17 Earthquakes > 4.5
Magnitude :4.5 at Fiji region 01:44:12 ago
Magnitude :4.6 at 128 km W of San Juan, Peru 10:06:35 ago
Magnitude :4.5 at 7 km NW of Papayal, Peru 10:32:50 ago
Magnitude :4.9 at 56 km SSE of Sand Point, Alaska 10:47:22 ago
Magnitude :4.9 at 22 km SSE of Padong, Philippines 11:05:28 ago
Magnitude :5.1 at 70 km WNW of San Antonio de los Cobres, Argentina 12:25:58 ago
Magnitude :5.3 at 116 km SSE of Kushiro, Japan 13:12:09 ago
Magnitude :4.6 at 112 km SSE of Kushiro, Japan 13:13:00 ago
Magnitude :4.8 at 129 km WNW of Pangai, Tonga 13:49:43 ago
Magnitude :4.9 at 256 km ESE of Ust'-Kamchatsk Staryy, Russia 20:15:25 ago
Magnitude :4.6 at 120 km NNE of Tobelo, Indonesia 20:23:53 ago
Magnitude :4.9 at 57 km ESE of Koseda, Japan 21:28:56 ago
Magnitude :4.8 at 24 km NE of La Paz, Philippines 22:38:44 ago
Magnitude :4.5 at northern Qinghai, China 23:10:02 ago
Magnitude :4.6 at 8 km S of Padong, Philippines 23:13:24 ago
Magnitude :4.5 at Fiji region 23:19:04 ago
Magnitude :5.1 at 108 km SSE of Hihifo, Tonga 23:39:10 ago

```

10.12.10 USGS JSON Response Structure

10.12.10.1 Response Schema

The USGS earthquake API returns data in GeoJSON format with the following structure:

```

{
  "type": "FeatureCollection",
  "metadata": { ... },
  "features": [ ... ],
  "bbox": [ ... ]
}

```

10.12.10.2 Metadata Section

```

"metadata": {
  "generated": <timestamp>,
  "url": <string>,
  "title": <string>,
  "status": <number>,
  "api": <version>,
  "count": <number>
}

```

10.12.10.3 Features Array

Each earthquake event is represented as a feature:

```

"features": [
  {
    "type": "Feature",
    "properties": {

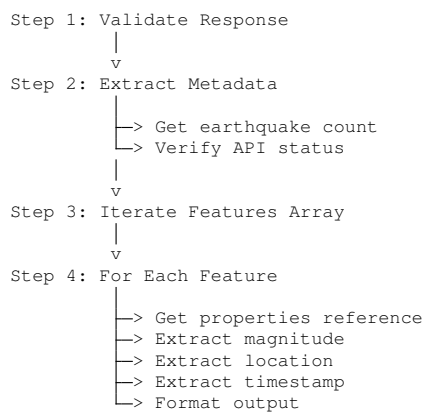
```

```

        "mag": <number>,
        "place": <string>,
        "time": <timestamp>,
        "updated": <timestamp>,
        "url": <string>,
        "status": <string>,
        "tsunami": <0|1>,
        ...
    },
    "geometry": {
        "type": "Point",
        "coordinates": [<lon>, <lat>, <depth>]
    },
    "id": <string>
},
...
]

```

10.12.10.4 Parsing Strategy



10.12.11 Complete JSON Sample

The following shows a condensed version of the actual JSON returned by the USGS API. To view the complete response structure when running the example, uncomment this line in the source code:

```
// JsonResult.PrintObject(true);
```

10.12.11.1 Sample Structure

```

{
  "type": "FeatureCollection",
  "metadata": {
    "generated": 1666815488000,
    "url": "https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/4.5_day.geojson",
    "title": "USGS Magnitude 4.5+ Earthquakes, Past Day",
    "status": 200,
    "api": "1.10.3",
    "count": 17
  },
  "features": [
    {
      "type": "Feature",
      "properties": {
        "mag": 4.5,
        "place": "Fiji region",
        "time": 1666809240464,
        "updated": 1666814388040,
        "url": "https://earthquake.usgs.gov/earthquakes/eventpage/us7000ikie",
        "status": "reviewed",
        "tsunami": 0,
        "type": "earthquake"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [-178.3577, -20.3848, 551.981]
      },
      "id": "us7000ikie"
    },
    ...
  ],
  "bbox": [-178.3577, -24.0942, 10, 166.3788, 55.5213, 590.211]
}

```

10.12.12 Best Practices

10.12.12.1 Memory Management

- [JsonRef](#) objects are lightweight references, not copies
- `const char*` strings are valid only while the [ParsedJsonDataSet](#) exists
- Use [NBString](#) for persistent storage

10.12.12.2 Error Handling

Always validate before accessing data:

```
JsonRef element = pjds("SomeKey");
if (element.IsValid()) {
    // Safe to access
    int value = element;
} else {
    // Handle missing or invalid data
}
```

10.12.12.3 Performance Considerations

Best Practice: Cache [JsonRef](#) Objects

Good:

```
JsonRef props = pjds("features")[i]("properties");
double mag = props("mag");
NBString place = props("place");
```

Avoid:

```
double mag = pjds("features")[i]("properties")("mag");
NBString place = pjds("features")[i]("properties")("place");
```

10.12.12.4 Type Safety

Use type checking before conversion:

```
if (element.IsNumber()) {
    double value = element;
}
if (element.IsString()) {
    NBString text = element;
}
```

10.12.13 Advanced Usage

10.12.13.1 Nested Array Navigation

```
// Access deeply nested structures
JsonRef deepElement = pjds("level1")("level2")[index]("level3");
```

10.12.13.2 Dynamic Key Access

```
// Build key names at runtime
NBString keyName = "dynamic_" + NBString(i);
int value = pjds(keyName.c_str());
```

10.12.13.3 Array Iteration Patterns

```
// Safe iteration with validation
JsonRef array = pjds("myArray");
if (array.IsArray()) {
    for (int i = 0; array[i].IsValid(); i++) {
        // Process array[i]
    }
}
```

10.12.14 Summary

The JSON Lexer provides a powerful, intuitive interface for working with JSON data in NetBurner applications:

- [ParsedJsonDataSet](#) handles parsing and storage
- [JsonRef](#) simplifies navigation of complex structures
- Automatic type conversion reduces boilerplate code
- Validation functions ensure safe data access
- Performance optimizations through internal poolBuffer management

For complete working examples, refer to the NetBurner example applications in the `\nburn\examples` directory.

10.13 NetBurner RTOS

10.13.1 Introduction

The NetBurner Real-Time Operating System (NBRTOS) is a full-featured preemptive multitasking RTOS supporting:

- Multiple tasks with 255 priority levels
- Semaphores for resource protection
- Mailboxes for message passing
- FIFOs for flexible queuing
- Message queues for structured communication
- Critical sections for mutual exclusion

The RTOS comes pre-configured as part of the NetBurner development package. Your application begins execution in the `UserMain()` task.

10.13.2 Priority Levels

To conserve system resources, the default number of tasks is 64, with 64 being the Idle task that will run when all other higher priority tasks are blocked. You can increase the number of tasks up to 255. There are a number of system tasks running at priorities defined in [constants.h](#). The recommended `UserMain()` priority is `MAIN_PRIO`, which is a value of 50. When creating additional tasks:

- Add or subtract from `MAIN_PRIO` for readability. For example, `MAIN_PRIO-1` is a higher priority task, which `MAIN_PRIO+1` is a lower priority task.

- Use the helper task function: `OSGetNextPrio()` to choose. For example, `OSGetNextPrio(← OSNextPrio::Above)`

```
enum class OSNextPrio {
    Maximum = -2, // Highest priority currently unused, 0->OS_LO_PRIO
    Above = -1, // Lowest priority that is higher than the current task, OSTaskID()->0
    Below = 0, // Highest priority that is lower than the current task,
    OSTaskID()->OS_LO_PRIO
    Next = Below, // Highest priority that is lower than the current task,
    OSTaskID()->OS_LO_PRIO
    Minimum = 1 // Lowest priority currently unused, OS_LO_PRIO->0
};
```

Task Priority Range:

Priority Range: 1 - 255 (Default maximum is 64).

Lower Number = Higher Priority 1 <--- Highest Priority 2 .


```

.   Application Tasks
.
64 <--- Lowest Priority (Idle)
.   (Default Maximum)
.
.
.
.
254
255 <--- Lowest Priority

```

Important**: Each priority level can only be used by ONE task at a time. Always check return values when creating tasks or changing priorities.

10.13.3 What is a Preemptive RTOS?

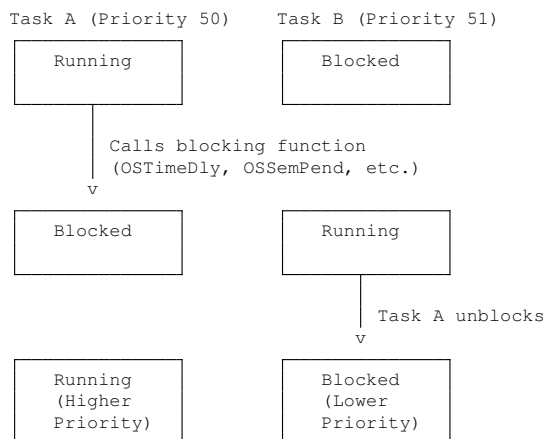
10.13.4 Core Principle

The highest priority task ready to run will ALWAYS run.**

This is fundamentally different from time-sliced operating systems like Windows or Unix. In NBRRTOS:

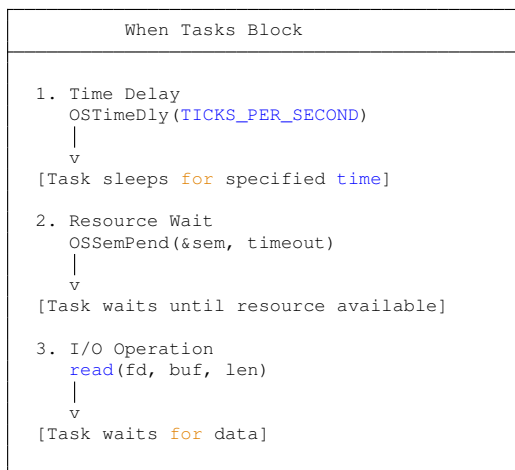
- No round-robin scheduling
- No time slices
- Priority-based execution only

10.13.5 Task Execution Flow



10.13.6 Blocking Scenarios

A higher priority task MUST block for lower priority tasks to execute:



10.13.7 Function Categories

10.13.7.1 RTOS Blocking Functions

Functions that pend on resources or create time delays:

- `OSTimeDly()` - Time delay
- `OSSemPend()` - Semaphore wait
- `OSMboxPend()` - Mailbox wait
- `OSQPend()` - Queue wait
- `OSFlagPendAny()` - Flag wait (any)
- `OSFlagPendAll()` - Flag wait (all)

10.13.7.2 I/O Functions That Block

Functions that perform read operations or pend on file descriptors:

- `select()` - Wait for I/O readiness
- `read()` - Read data (blocks until data available)
- `write()` - Write data (blocks until space available)
- `gets()` - Get string from input
- `getchar()` - Get character
- `fgets()` - Get string from file

10.13.7.3 Network Functions That Block

Functions that wait for connections or data:

- `accept()` - Wait for incoming connection
- `UDPPacket()` - Wait to receive UDP packet (when configured for receive)

10.13.7.4 Functions That Unblock Tasks

Functions that post to pending resources:

- `OSMboxPost()` - Post message to mailbox
- `OSQPost()` - Post message to queue
- `OSSemPost()` - Post to semaphore

10.13.8 System Task Priorities

System tasks use predefined priority levels defined in `nbrtos/include/constants.h`. The exact tasks depend on:

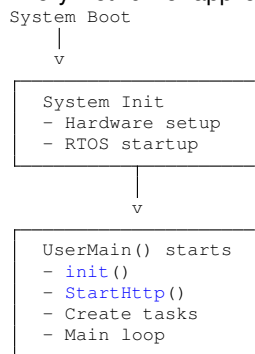
- Your platform (MOD5441X, SOMRT1061, etc.)
- Enabled system features (HTTP server, network stack, etc.)

Example*: Calling `StartHTTP()` creates a system task for web server request handling.

10.13.9 Task Creation

10.13.10 Entry Point: UserMain()

Every NetBurner application begins in the `UserMain()` task, which is created automatically by the system.



10.13.11 Task Creation Methods

10.13.11.1 Method 1: OSTaskCreatewName() - Full Control

Provides complete control over all task parameters with status return value.

Parameters**:

1. Task function name
2. Optional parameter to pass (NULL if unused)
3. Top of task stack
4. Bottom of task stack

5. Task priority
6. Task name (string)

Returns**: Status code (OS_NO_ERR on success)

10.13.11.2 Method 2: OSSimpleTaskCreatewName() - Simplified

Automatically allocates stack space, requires only:

1. Task function name
2. Task priority
3. Task name (string)

Note**: Does not return status code

10.13.12 Priority Selection Helper

Use `OSGetNextPrio()` to automatically find an available priority, but ensure:

- Not higher than critical system tasks (e.g., `ETHER_TASK_PRIO`)
- Not too low for your real-time requirements
- Refer to [constants.h](#) for system priority definitions

10.13.13 Complete Example

```

/*
    Task Creation Example
    Demonstrates both OSTaskCreateName() and OSSimpleTaskCreateName()
*/
#include <init.h>
#include <stdlib.h>
#include <nbrtos.h>
#include <system.h>
#include <utils.h>

const char *AppName = "Task Creation Example";

// Stack allocation for full task creation method
uint32_t TaskAllParamsStack[USER_TASK_STK_SIZE];

/*
    Task created with OSTaskCreateName()
    Demonstrates passing parameters and full control
*/
void TaskAllParams(void *pd)
{
    uint32_t loopCount = 0;
    uint32_t delayTime = (uint32_t)pd; // Cast parameter

    printf("TaskAllParams delay: %ld seconds\r\n", delayTime);

    while (1)
    {
        printf("TaskAllParams iteration: %ld\r\n", loopCount);
        loopCount++;
        OSTimeDly(TICKS_PER_SECOND * delayTime);
    }
}

/*
    Task created with OSSimpleTaskCreateName()
    Simplified creation with automatic stack allocation
*/
void TaskSimple(void *pd)
{
    uint32_t loopCount = 0;
    uint32_t delayTime = 6;

    printf("TaskSimple delay: %ld seconds\r\n", delayTime);

    while (1)
    {
        printf("TaskSimple iteration: %ld\r\n", loopCount);
        loopCount++;
        OSTimeDly(TICKS_PER_SECOND * delayTime);
    }
}

/*
    Main entry point
*/
void UserMain(void *pd)
{
    uint32_t delayTime = 3;
    int returnCode;

    init(); // Initialize network stack
    WaitForActiveNetwork(TICKS_PER_SECOND * 5); // Wait for DHCP

    // Create task with full parameter control
    printf("Creating TaskAllParams...\n");
    returnCode = OSTaskCreateName(
        TaskAllParams, // Task function
        (void *)delayTime, // Parameter to pass
        &TaskAllParamsStack[USER_TASK_STK_SIZE], // Stack top
        TaskAllParamsStack, // Stack bottom
        MAIN_PRIO - 1, // Priority
        "TaskAllParams" // Name
    );

    if (returnCode == OS_NO_ERR)
        printf("Success\r\n");
    else
        printf("*** Error: %d\r\n", returnCode);

    // Create task with simplified method
    printf("Creating TaskSimple\r\n");
    OSSimpleTaskCreateName(TaskSimple, MAIN_PRIO - 2, "TaskSimple");

    while (1)

```

```

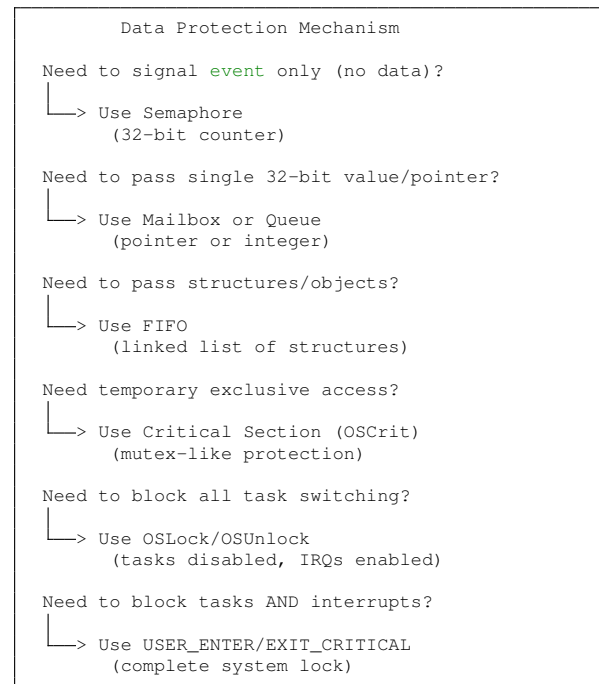
{
    OSTimeDly(TICKS_PER_SECOND);
}

```

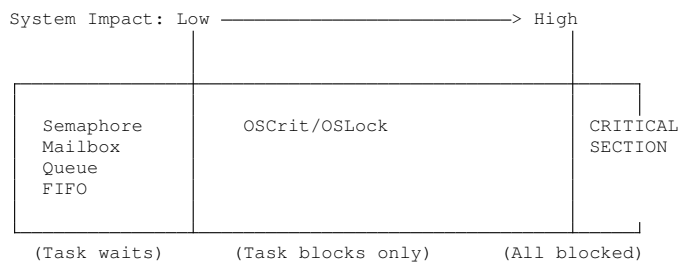
10.13.14 Protecting Shared Data

10.13.15 Mechanism Selection Guide

Choose the appropriate protection mechanism based on your needs:



10.13.16 Protection Mechanisms (Ordered by System Impact)



10.13.17 Detailed Mechanism Comparison

Mechanism	Purpose	Blocks	IRQ Impact	Best For
OSSemPend()/Post()	Signal/protect resource	Task only	None	Event signaling, simple resource protection
OSMboxPend()/Post()	Pass pointer message	Task only	None	Passing single message/pointer between tasks
OSQPend()/Post()	Message queue (FIFO)	Task only	None	Multiple messages, bounded queue
OSFifoPend()/Post()	Linked list queue	Task only	None	Unbounded message passing, complex structures
OSCritEnter()/Exit()	Counted mutex	Task only	None	Multi-task resource protection

Mechanism	Purpose	Blocks	IRQ Impact	Best For
OSLock()/Unlock()	Disable task switch	All tasks	None	Short critical sections
USER_ENTER/EXIT↔ _CRITICAL()	Disable all	Tasks + IRQs	All disabled	Absolute protection (use sparingly)

10.13.18 Important Considerations

10.13.18.1 Queue vs. FIFO

Queue**:

- Fixed size (declared at compile time)
- Bounded memory usage
- Predictable behavior
- Good for embedded systems

FIFO**:

- Dynamic size (linked list)
- Memory allocated by application
- Flexible but requires memory management
- Avoid dynamic allocation in embedded systems

Recommendation**: Use static memory pools for FIFO structures to avoid fragmentation.

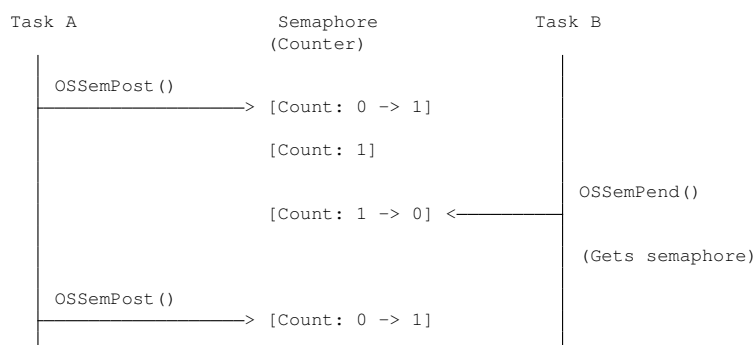
10.13.19 Semaphores

10.13.20 Concept

A semaphore is a protected counter used for:

- Controlling access to shared resources
- Signaling event occurrence
- Task synchronization

10.13.21 Operation Flow



10.13.22 Usage Pattern

```

// 1. Declare semaphore
OS_SEM MySemaphore;

// 2. Initialize semaphore (typically in UserMain)
OS_SemInit(&MySemaphore, 0); // Initial value: 0

// 3. Post to semaphore (signal/release)
  
```

```

OSSemPost (&MySemaphore);

// 4. Pend on semaphore (wait/acquire)
OSSemPend(&MySemaphore, 0); // 0 = wait forever

// 5. Pend with timeout
OSSemPend(&MySemaphore, TICKS_PER_SECOND * 5); // Wait 5 seconds

```

10.13.23 Timeout Specification

Timeout Parameter:

0	= Wait forever
TICKS_PER_SECOND	= 1 second
TICKS_PER_SECOND * 5	= 5 seconds
TICKS_PER_SECOND / 2	= 500 milliseconds

10.13.24 Resource Protection Example

```

OS_SEM SerialPortSem;

void UserMain(void *pd) {
    init();
    OSSemInit(&SerialPortSem, 1); // Initial count: 1 (available)

    // Create tasks that share serial port...
}

void TaskA(void *pd) {
    while (1) {
        OSSemPend(&SerialPortSem, 0); // Acquire

        // Protected: exclusive access to serial port
        SerialPort.Write("Task A\r\n");

        OSSemPost (&SerialPortSem); // Release

        OSTimeDly(TICKS_PER_SECOND);
    }
}

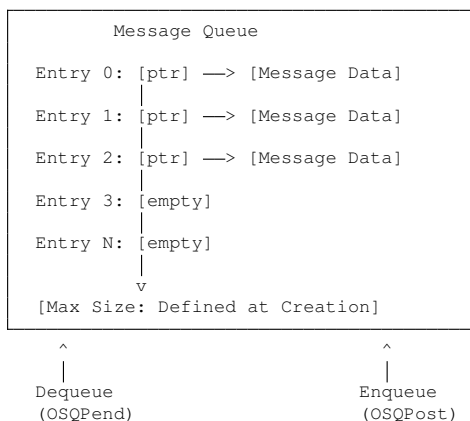
```

10.13.25 Message Queues

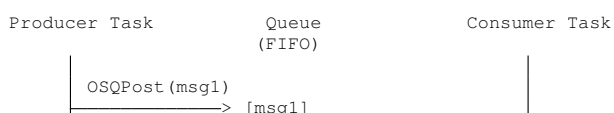
10.13.26 Concept

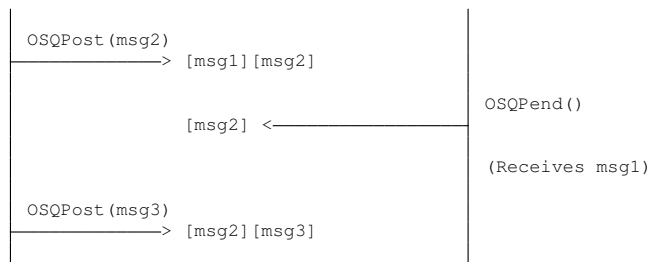
Message queues enable tasks and ISRs to send and receive pointer-sized messages. The pointers typically reference structures or objects containing actual data.

10.13.27 Queue Structure



10.13.28 Message Flow





10.13.29 Usage Example

```

// Define message structure
typedef struct {
    uint32_t sensorId;
    float temperature;
    uint32_t timestamp;
} SensorReading;

// Declare queue
OS_Q SensorQueue;
void *SensorQueueStorage[10]; // Storage for 10 messages

void UserMain(void *pd) {
    init();

    // Initialize queue with 10 entries
    OSQInit(&SensorQueue, SensorQueueStorage, 10);

    // Create producer and consumer tasks...
}

void SensorTask(void *pd) {
    SensorReading reading;

    while (1) {
        // Read sensor
        reading.sensorId = 1;
        reading.temperature = ReadSensor();
        reading.timestamp = Secs;

        // Post to queue
        OSQPost(&SensorQueue, (void *)&reading);

        OSTimeDly(TICKS_PER_SECOND);
    }
}

void ProcessingTask(void *pd) {
    while (1) {
        // Wait for message
        SensorReading *msg = (SensorReading *)OSQPend(&SensorQueue, 0);

        // Process message
        printf("Sensor %lu: %.2fC at %lu\r\n",
            msg->sensorId, msg->temperature, msg->timestamp);
    }
}

```

10.13.30 FIFOs

10.13.31 Concept

FIFOs are similar to queues but specifically designed for linked-list-based message passing. Unlike queues with fixed size, FIFOs grow dynamically within available memory.

10.13.32 Structure Requirements

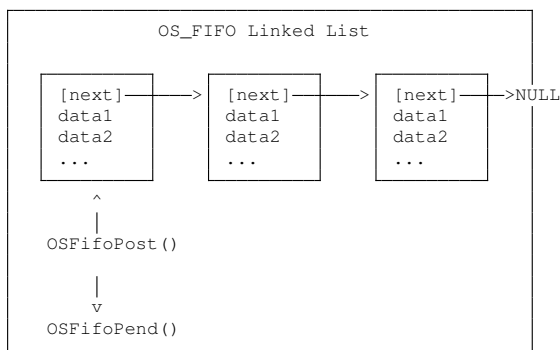
The first element of any FIFO structure MUST be a `void *` pointer for OS internal linking:

```

typedef struct MyFifoStruct {
    void *next; // REQUIRED: Must be first element
    uint32_t data1; // Your data
    float data2; // Your data
    char message[64]; // Your data
} MyFifoStruct;

```


10.13.33 FIFO Architecture



10.13.34 Memory Management Strategies

10.13.34.1 Strategy 1: Static Pool (Recommended)

```

#define FIFO_POOL_SIZE 20

// Create static pool of FIFO structures
MyFifoStruct fifoPool[FIFO_POOL_SIZE];
bool poolInUse[FIFO_POOL_SIZE] = {false};

// Allocate from pool
MyFifoStruct* AllocateFifoStruct() {
    for (int i = 0; i < FIFO_POOL_SIZE; i++) {
        if (!poolInUse[i]) {
            poolInUse[i] = true;
            return &fifoPool[i];
        }
    }
    return NULL; // Pool exhausted
}

// Return to pool
void FreeFifoStruct(MyFifoStruct *ptr) {
    int index = ptr - fifoPool;
    if (index >= 0 && index < FIFO_POOL_SIZE) {
        poolInUse[index] = false;
    }
}

```

10.13.34.2 Strategy 2: Dynamic Allocation (Not Recommended)

```

// Avoid in embedded systems due to fragmentation risk
MyFifoStruct *ptr = (MyFifoStruct *)malloc(sizeof(MyFifoStruct));
// ... use ...
free(ptr);

```

10.13.35 Queue vs. FIFO Decision

Choose Queue When:

```

Fixed message count
Predictable memory
Simple data types
Bounded behavior

```

Choose FIFO When:

```

Variable msg count
Complex structures
Need flexibility
Can manage memory

```

10.13.36 FIFO Functions

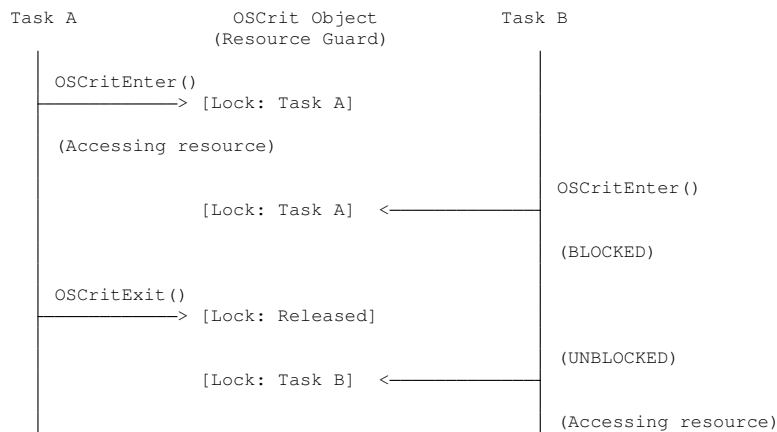
- OSFifoInit() - Initialize FIFO
- OSFifoPost() - Add to end of FIFO
- OSFifoPostFirst() - Add to front of FIFO (priority)
- OSFifoPend() - Wait for and remove first item
- OSFifoPendNoWait() - Check and remove without waiting

10.13.37 Critical Sections

10.13.38 OSCrit - Counted Critical Section

A counting mutex that restricts resource access to one task at a time.

10.13.39 Operation Flow



10.13.40 Counting Behavior

Critical Section Counter:

```

OSCritEnter()  -> Count++
OSCritExit()   -> Count--

When Count reaches 0:
-> Resource released
-> Next task can enter
  
```

10.13.41 Usage Examples

10.13.41.1 Manual Enter/Exit

```

OSCritObj linkedListGuard;

void TaskA(void *pd) {
    while (1) {
        OSCritEnter(&linkedListGuard, 0); // Enter critical section

        // Protected: manipulate linked list
        linkedList.Add(data);

        OSCritExit(&linkedListGuard);      // Exit critical section

        OSTimeDly(TICKS_PER_SECOND);
    }
}
  
```

10.13.41.2 C++ Scoped Object (Recommended)

```

OSCritObj linkedListGuard;

void TaskB(void *pd) {
    while (1) {
        {
            // Automatic enter on construction
            OSCritObj lock(&linkedListGuard);

            // Protected: manipulate linked list
            linkedList.Remove(data);

        } // Automatic exit on destruction (scope end)

        OSTimeDly(TICKS_PER_SECOND);
    }
}
  
```

10.13.42 OSCrit vs. OSLock

OSCrit vs OSLock
OSCrit (Resource Mutex): - Blocks only tasks accessing same resource - Other tasks can still run - IRQs not affected - Good for: Protecting shared data OSLock (Task Switch Disable): - Blocks ALL task switching - No tasks can preempt - IRQs still run - Good for: Brief critical operations

10.13.43 OSLock Usage Pattern

```
void QuickOperation() {
    OSLock(); // Disable task switching

    // Very brief critical operation
    // (keep this SHORT!)
    globalCounter++;

    OSUnlock(); // Enable task switching
}

// C++ scoped version
void QuickOperationCpp() {
    {
        OSLockObj lock; // Auto OSLock() on creation

        // Brief critical operation
        globalCounter++;

    } // Auto OSUnlock() on destruction
}
```

Warning:** Keep OSLock sections as short as possible to maintain system responsiveness.

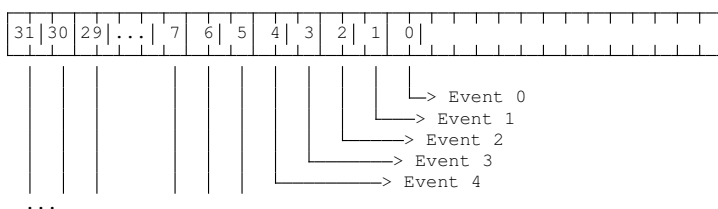
10.13.44 OS Flags

10.13.45 Concept

OSFlags enable monitoring multiple events simultaneously using a 32-bit bitmap where each bit represents a distinct flag or event.

10.13.46 Flag Bitmap Structure

32-bit Flag Register:



10.13.47 Flag Operations

Flag Manipulation:

```
OSFlagSet(flags, 0x0001)
[Bit 0 = 1]

OSFlagSet(flags, 0x0004)
[Bit 0 = 1, Bit 2 = 1]

OSFlagClear(flags, 0x0001)
```

```
[Bit 2 = 1]
```

10.13.48 Pending Modes

Wait **for** ANY flag (OR):

```
Mask: 0x0007 (bits 0, 1, 2)

OSFlagPendAny(flags, 0x0007, ...)

Unblocks when:
Bit 0 = 1 OR
Bit 1 = 1 OR
Bit 2 = 1
```

Wait for ALL flags (AND):

```
Mask: 0x0007 (bits 0, 1, 2)

OSFlagPendAll(flags, 0x0007, ...)

Unblocks when:
Bit 0 = 1 AND
Bit 1 = 1 AND
Bit 2 = 1
```

10.13.49 Flag Functions Summary

Function	Description
OSFlagCreate()	Create flag object
OSFlagSet()	Set specified bits
OSFlagClear()	Clear specified bits
OSFlagState()	Read current flag value
OSFlagPendAll()	Wait until ALL specified flags set
OSFlagPendAny()	Wait until ANY specified flag set
OSFlagPendNoWait()	Check ALL flags without waiting
OSFlagPendAnyNoWait()	Check ANY flag without waiting

10.13.50 Usage Example

```
OS_FLAGS systemFlags;

// Define flag bits
#define FLAG_SENSOR_READY 0x0001 // Bit 0
#define FLAG_NETWORK_UP 0x0002 // Bit 1
#define FLAG_DATA_AVAILABLE 0x0004 // Bit 2
#define FLAG_ERROR 0x0008 // Bit 3

void UserMain(void *pd) {
    init();
    OSFlagCreate(&systemFlags);

    // Create tasks...
}

void SensorTask(void *pd) {
    while (1) {
        // Read sensor
        if (SensorValid()) {
            OSFlagSet(&systemFlags, FLAG_SENSOR_READY);
        }
        OSTimeDly(TICKS_PER_SECOND);
    }
}

void ProcessingTask(void *pd) {
    while (1) {
        // Wait for sensor AND network
        uint32_t flags = OSFlagPendAll(
            &systemFlags,
```

```

        FLAG_SENSOR_READY | FLAG_NETWORK_UP,
        TICKS_PER_SECOND * 10
    );

    if (flags) {
        // Both conditions met, process data
        ProcessSensorData();

        // Clear flags
        OSFlagClear(&systemFlags, FLAG_SENSOR_READY);
    }
}

void MonitorTask(void *pd) {
    while (1) {
        // Check for any error condition (non-blocking)
        uint32_t flags = OSFlagPendAnyNoWait(
            &systemFlags,
            FLAG_ERROR
        );

        if (flags & FLAG_ERROR) {
            printf("Error detected!\r\n");
            OSFlagClear(&systemFlags, FLAG_ERROR);
        }

        OSTimeDly(TICKS_PER_SECOND / 10);
    }
}

```

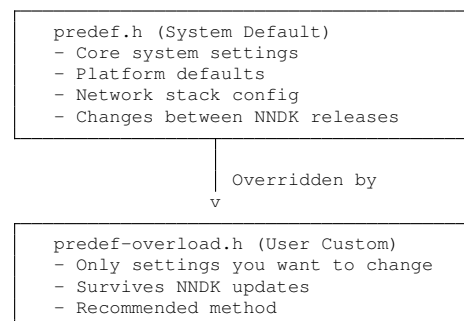
10.13.51 System Configuration

10.13.52 Predef.h and Predef-overload.h

The NetBurner system uses `predef.h` for base-level system configuration through `#define` statements.

10.13.53 Configuration Architecture

System Configuration Hierarchy:



10.13.54 Why Use Predef-overload.h?

Problems with modifying `predef.h` directly**:

1. File changes frequently between NNDK releases
2. Difficult to track your custom changes
3. Merge conflicts during updates
4. Confusing compilation errors after upgrades

Benefits of `predef-overload.h`**:

1. Only contains YOUR changes
2. Survives NNDK updates cleanly
3. Easy to review and maintain
4. No merge conflicts

10.13.55 Setup Methods

10.13.55.1 NBEclipse Setup

Refer to NBEclipse Getting Started Guide - Overload Directory section

10.13.55.2 Command Line Tools Setup

Refer to Command Line Tools - Overload section

10.13.56 Predef-overload.h Best Practices

```
/*
    predef-overload.h
    Custom system configuration overrides
*/

// Best practice: #undef before #define
// Prevents duplicate definition warnings

// Example 1: Disable IPv6
#undef IPV6
#undef IPV4ONLY
#define IPV4ONLY (1)

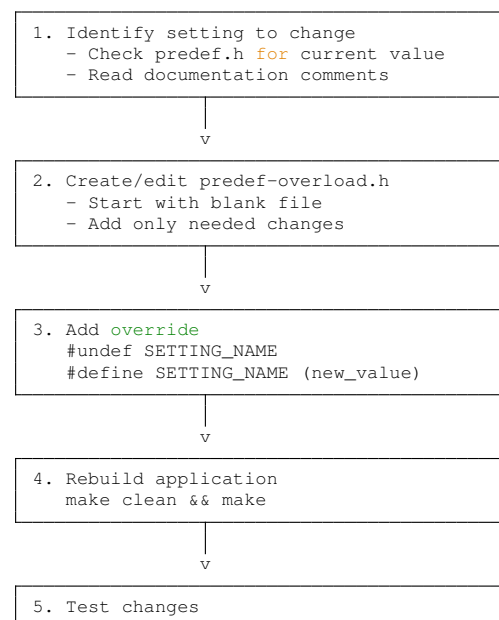
// Example 2: Custom buffer sizes
#undef TCP_BUFFER_SIZE
#define TCP_BUFFER_SIZE (8192)

// Example 3: Enable debug features
#undef DEBUG_LEVEL
#define DEBUG_LEVEL (2)

// Example 4: Stack sizes
#undef USER_TASK_STK_SIZE
#define USER_TASK_STK_SIZE (4096)
```

10.13.57 Configuration Workflow

Development Process:



10.13.58 Common Configuration Overrides

```
// Network Configuration
#undef IPV4ONLY
#define IPV4ONLY (1)           // IPv4-only mode

#undef DEFAULT_NET_TIMEOUT
```

```

#define DEFAULT_NET_TIMEOUT (30)           // 30 second timeout

// Task Configuration
#undef USER_TASK_STK_SIZE
#define USER_TASK_STK_SIZE (8192)         // 8KB stack per task

#undef MAX_USER_TASKS
#define MAX_USER_TASKS (20)                // Support 20 user tasks

// System Resources
#undef NUM_OS_FIFOS
#define NUM_OS_FIFOS (32)                  // 32 FIFO objects

#undef NUM_OS_SEMS
#define NUM_OS_SEMS (64)                   // 64 semaphores

// Debug Configuration
#undef DEBUG_BUILD
#define DEBUG_BUILD (1)                    // Enable debug features

#undef ENABLE_REMOTE_CONSOLE
#define ENABLE_REMOTE_CONSOLE (1)          // Enable remote console

```

10.13.59 Verification After Changes

```

# Clean build to ensure all changes applied
make clean

# Rebuild with new configuration
make

# Verify settings at runtime
# Check console output or use system diagnostics

```

10.13.60 Summary

10.13.61 Key Takeaways

1. RTOS Fundamentals

- Highest priority ready task always runs
- Tasks must block for lower priority tasks to execute
- 255 priority levels (1 = highest, 255 = lowest)

2. Task Creation

- Entry point: `UserMain()`
- Full control: `OSTaskCreateName()`
- Simplified: `OSSimpleTaskCreateName()`

3. Data Protection Hierarchy

Semaphore < Mailbox < Queue < FIFO < OSCrit < OSLock < Critical Section
(Least impact) —————> (Most impact)

4. Best Practices

- Use `predef-overload.h` for configuration
- Prefer static memory over dynamic allocation
- Keep critical sections short
- Check return values from task creation
- Match enter/exit calls for counted resources

5. Configuration Management

- Never modify `predef.h` directly
- Use `predef-overload.h` for customization
- Always `#undef` before `#define`
- Clean rebuild after configuration changes

10.13.62 Quick Reference

Common RTOS Calls:

```

├── Timing
│   └── OSTimeDly(TICKS_PER_SECOND)
├── Semaphores
│   ├── OSSemInit(&sem, 0)
│   ├── OSSemPend(&sem, timeout)
│   └── OSSemPost(&sem)
├── Queues
│   ├── OSQInit(&queue, storage, size)
│   ├── OSQPend(&queue, timeout)
│   └── OSQPost(&queue, msg)
├── Flags
│   ├── OSFlagCreate(&flags)
│   ├── OSFlagSet(&flags, bits)
│   ├── OSFlagPendAny(&flags, mask, timeout)
│   └── OSFlagPendAll(&flags, mask, timeout)
└── Critical Sections
    ├── OSCritEnter(&crit, timeout)
    ├── OSCritExit(&crit)
    ├── OSLock()
    └── OSUnlock()

```

10.13.63 Further Reading

Refer to the following documentation:

- [constants.h](#) - System task priorities and constants
- API reference documentation - Complete function details
- Platform-specific guides - Hardware integration
- Example applications - Working implementations

10.14 Network Protocols TCP and UDP

10.14.1 TCP vs UDP Comparison

This guide covers TCP and UDP network protocols in NetBurner applications, including practical implementation patterns and complete working examples. The first section is an overview and comparison of TCP vs UDP, followed by detailed discussions of each:

- [TCP Protocol](#)
- [UDP Protocol](#)

When designing network applications, choosing between TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) is a critical architectural decision. Each protocol serves different use cases based on reliability requirements, overhead tolerance, and application characteristics.

10.14.1.1 Protocol Characteristics

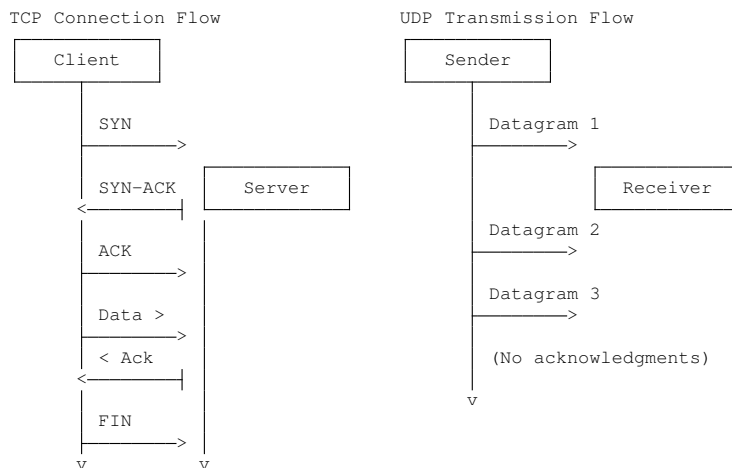
10.14.1.1.1 TCP (Transmission Control Protocol)

- **Connection-oriented:** Point-to-point stream-based protocol
- **Reliable delivery:** Guarantees packet delivery with acknowledgments
- **Ordered transmission:** Data arrives in the same sequence it was sent
- **Error detection:** Built-in error checking and retransmission
- **Stream-based:** Continuous data flow without message boundaries

10.14.1.1.2 UDP (User Datagram Protocol)

- **Connectionless:** Fire-and-forget datagram protocol
- **Best-effort delivery:** No delivery guarantees
- **Unordered:** Packets may arrive out of sequence
- **Lightweight:** Minimal overhead, no connection establishment
- **Message-based:** Each send operation creates one discrete datagram

10.14.1.2 Protocol Comparison Diagram



10.14.1.3 Real-World Analogies

TCP*: Like a telephone call

- Establish connection (dial and answer)
- Two-way conversation with confirmation ("Hello?" "Yes, I hear you")
- Orderly exchange of information
- Explicit termination ("Goodbye")

UDP*: Like sending postcards

- Write and send without confirmation
- No guarantee of delivery
- No confirmation the recipient received it
- Messages may arrive out of order

10.14.1.4 Decision Factors

10.14.1.4.1 Choose TCP When:

- **Reliability is critical:** Financial transactions, control commands, configuration data
- **Order matters:** Sequential data like file transfers or protocol messages
- **Connection state is useful:** Authentication, session management
- **Data integrity is paramount:** Critical system operations

10.14.1.4.2 Choose UDP When:

- **Speed is essential:** Real-time applications, gaming, video streaming
- **Overhead is prohibitive:** High-frequency sensor data, telemetry
- **Application handles reliability:** Custom retry logic at application layer
- **Broadcast/multicast needed:** Network discovery, status updates
- **Occasional loss is acceptable:** Streaming media, periodic updates

10.14.1.5 Common Protocol Usage

10.14.1.5.1 Applications Using TCP:

- HTTP/HTTPS - Web traffic
- FTP - File transfers
- Telnet - Remote terminal
- SMTP - Email transmission
- SSH - Secure remote access

10.14.1.5.2 Applications Using UDP:

- **DHCP** - Network configuration
- BOOTP - Bootstrap protocol
- SNMP - Network monitoring
- DNS - Name resolution
- NTP - Time synchronization
- VoIP - Voice communications

10.14.1.6 Performance Considerations

TCP Overhead Example**:

Sending 10 A/D readings **using** TCP (single connection):

```
9827312932342498273129323424982731293234 <- Stream of digits
```

Problem: Cannot identify individual readings without delimiters

UDP Datagram Example**:

Sending 10 A/D readings **using** UDP:

```
9827 3129 3234 2498 ...
```

Advantage: Each reading is a discrete packet

10.14.1.7 Key Takeaways

1. **TCP** provides reliability at the cost of overhead and connection management
2. **UDP** provides speed and simplicity at the cost of reliability
3. **SNMP** uses UDP to avoid connection overhead when monitoring many devices
4. **Custom protocols** may choose UDP and implement their own reliability layer
5. **Connection parameters:** TCP requires 4-tuple (source IP, source port, dest IP, dest port)

10.14.1.8 Additional Resources

10.14.1.8.1 NetBurner Tools

- **UDP Terminal** (Windows): Test UDP communication from PC
- **TcpServerWin.exe**: Simple TCP server for testing
- **MTTTY**: Serial terminal for debug output
- **nbdiscovery**: Network device discovery

10.14.1.8.2 Example Locations

- TCP Examples: \nburn\examples\TCP
- UDP Examples: \nburn\examples\UDP
- Web Examples: \nburn\examples\Web
- PC Tools: \nburn\pctools

10.14.1.8.3 Related Documentation

- HTTP/Web Server Implementation
- Serial Communication Protocols
- [MQTT](#) Client/Server
- SSL/TLS Secure Connections
- Network Configuration APIs

10.14.1.9 Quick Reference

10.14.1.9.1 TCP Functions

Function	Description
listen()	Create listening socket
accept()	Accept incoming connection
connect()	Initiate outbound connection
read()	Receive data
write()	Send data
close()	Close connection
select()	Multiplex multiple connections
fdprintf()	Formatted output to socket
writestring()	Send string to socket

10.14.1.9.2 UDP Class Methods

Function	Description
RegisterUDPFifo()	Register FIFO for port
UDPPacket()	Constructor (receives packet)
Validate()	Check packet validity
GetDataBuffer()	Get packet data pointer
GetDataSize()	Get data length
GetSourceAddress()	Get sender IP

Function	Description
SetSourcePort()	Set source port
SetDestinationPort()	Set destination port
AddData()	Add data to packet
Send()	Transmit packet

10.14.1.9.3 UDP Socket Functions

Function	Description
RegisterUDPFifo()	Register FIFO for port
UDPPacket()	Constructor (receives packet)
Validate()	Check packet validity
GetDataBuffer()	Get packet data pointer
GetDataSize()	Get data length
GetSourceAddress()	Get sender IP
SetSourcePort()	Set source port
SetDestinationPort()	Set destination port
AddData()	Add data to packet
Send()	Transmit packet
CreateRxUdpSocket()	Create receive socket
CreateTxUdpSocket()	Create transmit socket
recvfrom()	Receive datagram
sendto()	Send datagram

10.14.2 TCP Protocol

10.14.2.1 TCP Protocol

TCP establishes a reliable byte-stream connection between two network hosts. The listening host is the **server**, and the connecting host is the **client**. Unlike UDP, TCP transmits data as a continuous stream without automatic message boundaries.

10.14.2.2 TCP Stream Behavior

Data Transmission Without Delimiters:

```
Sender: "9827" "3129" "3234" "2498"
        [send] [send] [send] [send]
```

↓
v TCP Stream

```
Receiver sees: "98273129323424498..."
Problem: Where does one message end?
```

Solution: Application must add delimiters/framing

10.14.2.3 TCP Connection Requirements

Every TCP connection requires four parameters:

- Source IP address
- Source port number
- Destination IP address
- Destination port number

10.14.2.4 Well-Known Port Numbers

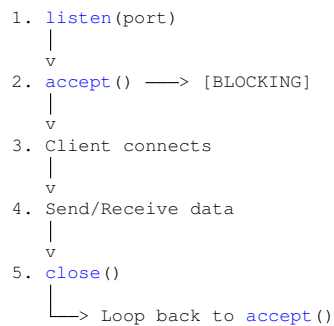
Protocol	Port	Description
FTP	21	File Transfer
Telnet	23	Remote Terminal
SMTP	25	Email Sending
DNS	53	Name Resolution
TFTP	69	Trivial FTP
HTTP	80	Web Traffic
POP3	110	Email Retrieval
NTP	123	Time Sync

10.14.2.5 TCP Server Implementation

A TCP server listens for incoming connections on a specified port, accepts connection requests, and communicates with connected clients.

10.14.2.5.1 TCP Server Flow

Server Lifecycle:



10.14.2.5.2 TCP Server Functions

Core functions for TCP server implementation:

- `listen()` - Open a listening socket on a specific port
- `accept()` - Accept incoming connection requests (blocking)
- I/O functions - `read()`, `write()`, `fdprintf()`, `writestring()`
- `close()` - Close network socket

10.14.2.5.3 Basic TCP Server Example

A NetBurner embedded TCP server application that creates a listening socket on port 23 (Telnet) and accepts client connections for communication. Location*: \nburn\examples\TCP

10.14.2.5.3.1 Overview

This application demonstrates how to implement a basic TCP server on NetBurner hardware. The server listens for incoming connections and provides a simple echo-style communication interface where it receives data from clients and displays it locally.

10.14.2.5.3.2 Features

- **TCP Server:** Creates a listening socket on port 23 (standard Telnet port)
- **Client Connection Handling:** Accepts incoming TCP connections from clients
- **Data Reception:** Reads data from connected clients using a simple `read()` function

- **Connection Management:** Handles client disconnections gracefully
- **Multi-tasking:** Uses RTOS tasks for concurrent operation

10.14.2.5.3.3 Technical Details

Configuration

- **Listen Port:** 23 (Telnet)
- **Receive Buffer Size:** 4096 bytes
- **Maximum Pending Connections:** 5
- **Task Priority:** MAIN_PRIO - 1 (higher than UserMain)

10.14.2.5.3.4 Key Components

TcpServerTask

The main server task that:

1. Creates a listening socket bound to INADDR_ANY on the specified port
2. Waits for client connections using `accept ()`
3. Sends welcome messages to connected clients
4. Continuously reads data from clients until disconnection
5. Closes connections and waits for new ones

Buffer Management

- Uses a global receive buffer (`RXBuffer`) of 4096 bytes
- Null-terminates received data for string processing
- Displays received data length and content via `printf()`

10.14.2.5.3.5 Usage

Testing with Telnet

To test the TCP server, use a Telnet client from any system on the same network:

```
telnet <netburner_ip_address>
```

Where `<netburner_ip_address>` is the IP address of your NetBurner device.

Expected Behavior

1. The server displays "Waiting for connection on port 23..."
2. When a client connects, it shows the client's IP address
3. The client receives welcome messages:
 - "Welcome to the NetBurner TCP Server"
 - Connection details with server IP and port
4. Any data sent by the client is echoed to the server console
5. When the client disconnects, the server closes the connection and waits for new ones

10.14.2.5.3.6 Implementation Notes

Connection Lifecycle

- The server uses a blocking `accept()` call to wait for connections
- Each client connection is handled sequentially (single client at a time)
- The `read()` function blocks until data is received or connection closes
- Connection closure is detected when `read()` returns 0 or negative value

Limitations

- **Single Client:** This example handles one client connection at a time
- **Sequential Processing:** New connections must wait until the current client disconnects
- **No Timeout:** Connections remain open indefinitely until client closes

For Multiple Clients

This example uses a simple `read()` approach for single client connections. For handling multiple simultaneous client connections, refer to the "TCP Multi Socket Server" example.

```
#include <init.h>
#include <tcp.h>
#include <iosys.h>
#include <fdprintf.h>

const char *AppName = "Simple TCP Server Example";

#define TCP_LISTEN_PORT 23 // Telnet port number
#define RX_BUFSIZE (4096)

char RXBuffer[RX_BUFSIZE];

// Allocate task stack for TCP listen task
uint32_t TcpServerTaskStack[USER_TASK_STK_SIZE];

/*-----
   TCP Server Task
   -----*/
void TcpServerTask(void *pd)
{
    int listenPort = (int)pd;

    // Set up the listening TCP socket
    int fdListen = listen(INADDR_ANY, listenPort, 5);

    if (fdListen > 0)
    {
        IPADDR clientAddress;
        uint16_t clientPort;

        while (1)
        {
            /* The accept() function will block the current task until a TCP client
               requests a connection. Once a client connection is accepting, the
               file descriptor fdAccept is used to read/write to it.
            */
            printf("Waiting for connection on port %d...\n", listenPort);
            int fdAccept = accept(fdListen, &clientAddress, &clientPort, 0);
            printf("Connected to: %I\r\n", GetSocketRemoteAddr(fdAccept));

            writestring(fdAccept, "Welcome to the NetBurner TCP Server\r\n");
            fdprintf(fdAccept, "You are connected to IP Address %I:%d\r\n", GetSocketRemoteAddr(fdAccept),
                GetSocketRemotePort(fdAccept));

            while (fdAccept > 0)
            {
                /* Loop while connection is valid. The read() function will return 0 or a negative number if
                   the client closes the connection, so we test the return value in the loop. Note: you can also
                   use ReadWithTimeout() in place of read to enable the connection to terminate after a period of
                   inactivity.
                */
                int n = 0;
                do
                {

```

```

        n = read(fdAccept, RXBuffer, RX_BUFSIZE);
        RXBuffer[n] = '\0';
        printf("Read %d bytes: %s\n", n, RXBuffer);
    } while (n > 0);

    printf("Closing client connection: %I\r\n", GetSocketRemoteAddr(fdAccept));
    close(fdAccept);
    fdAccept = 0;
}
} // while(1)
} // while listen
}

/*-----
User Main
-----*/
void UserMain(void *pd)
{
    init();
    //Enable system diagnostics. Probably should remove for production code.
    EnableSystemDiagnostics();
    WaitForActiveNetwork(TICKS_PER_SECOND*5);

    // Create TCP Server task
    OSTaskCreateWName(TcpServerTask,
        (void *)TCP_LISTEN_PORT,
        &TcpServerTaskStack[USER_TASK_STK_SIZE],
        TcpServerTaskStack,
        MAIN_PRIO - 1, /* higher priority than UserMain */
        "TCP Server");

    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND * 5);
    }
}

```

10.14.2.5.3.7 Basic Server Limitations

This simple example demonstrates core TCP server concepts but has limitations:

- Listens on only one port
- Handles only one connection at a time
- No graceful connection termination from server side
- Client must close connection

10.14.2.5.4 Multi-Connection Server Example

For production applications, use `select()` to handle multiple simultaneous connections efficiently.

10.14.2.5.4.1 Overview

This application demonstrates a TCP server implementation that can handle multiple simultaneous client connections using the NetBurner embedded platform. The server uses the `select()` function to efficiently manage multiple TCP connections concurrently without blocking.

10.14.2.5.4.2 Features

- **Multi-client support:** Handles up to 10 simultaneous TCP connections
- **Non-blocking I/O:** Uses `select()` to efficiently manage multiple file descriptors
- **Connection management:** Automatically accepts new connections and manages client disconnections
- **Simple protocol:** Clients can send 'Q' to quit their connection
- **Debug output:** Status messages displayed on debug serial port

10.14.2.5.4.3 Configuration

The application uses the following configurable parameters:

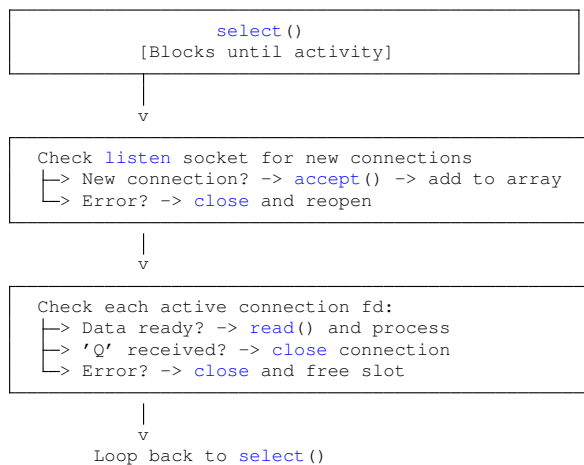
- **Listen Port:** 23 (Telnet port)
- **Maximum Connections:** 10 simultaneous clients
- **Read Buffer Size:** 1024 bytes per connection

These values can be modified by changing the corresponding `#define` statements in `main.cpp`.

10.14.2.5.4.4 How It Works

Multi-Connection Architecture

Select () Multiplexing Pattern:



Server Initialization

1. Initializes the NetBurner system and waits for network connectivity
2. Creates a listening socket on port 23
3. Displays the device's IP addresses for client connection reference

Connection Management

The server uses a file descriptor array (`fdArray`) to track active client connections. The main loop:

1. **Sets up file descriptor sets** for `select()` monitoring
2. **Calls `select()`** to block until activity occurs on any socket
3. **Processes new connections** when clients connect to the listening port
4. **Handles client data** when existing connections have data to read
5. **Manages errors** by closing problematic connections

Client Interaction

- **Welcome message:** New clients receive a greeting upon connection
- **Quit command:** Clients can send 'Q' to cleanly disconnect
- **Echo behavior:** The server reads client data and logs it to the debug port
- **Connection limits:** If the server is full (10 connections), new clients are rejected with an error message

10.14.2.5.4.5 Testing the Application

Using Telnet

The easiest way to test this application is using multiple Telnet sessions:

```
telnet <netburner_ip_address> 23
```

You can open up to 10 simultaneous Telnet connections to test the multi-client functionality.

Expected Behavior

1. **Successful connection:** You'll see "Welcome to the NetBurner Multi-Socket TCP Server! 'Q' to quit."
2. **Server full:** If you try to connect when 10 clients are already connected, you'll see "I am sorry, but the server is full"
3. **Quit command:** Type 'Q' to cleanly disconnect
4. **Debug output:** Connection status, received data, and errors are logged to the serial debug port

10.14.2.5.4.6 Code Structure

Key Functions

- `**UserMain()` **: Main application entry point and event loop
- `**select()` **: Monitors multiple file descriptors for activity
- `**accept()` **: Accepts new client connections
- `**read()` **: Reads data from client connections
- `**close()` **: Closes file descriptors and cleans up connections

File Descriptor Management

The application maintains an array of file descriptors (`fdArray`) where:

- Index 0-9: Active client connections (0 = unused slot)
- Separate `fdListen`: The listening socket for new connections

Error Handling

- **Connection errors:** Automatically detected and connections are closed
- **Read errors:** Treated as connection problems and handled gracefully
- **Listen socket errors:** The listening socket is closed and reopened

10.14.2.5.4.7 Troubleshooting

- **Cannot connect:** Verify the NetBurner device has network connectivity and check IP address
- **Connection refused:** Ensure no other service is using port 23
- **Unexpected disconnections:** Check debug serial output for error messages
- **Server appears full:** Existing connections may not have closed properly; restart the application

10.14.2.5.4.8 Source Code

```

#include <init.h>
#include <iosys.h>
#include <tcp.h>
#include <ipshow.h>

const char *AppName = "TCP Multisocket Server";

#define listenPort (23)          // TCP port number to listen on
#define maxConnections (10)      // Max number of file descriptors/connections
#define readBufferSize (1024)   // Connection read buffer size

int fdArray[maxConnections];    // Array of TCP file descriptors

/*-----
  User Main
  -----*/
void UserMain(void *pd)
{
    init();
    //Enable system diagnostics. Probably should remove for production code.
    EnableSystemDiagnostics();
    WaitForActiveNetwork(TICKS_PER_SECOND * 10);
    showIpAddresses();

    // Listen for incoming TCP connections. You only need to call listen() one time.
    // Any incoming IP address is allowed, queue up to 5 connection requests at one time
    int32_t fdListen = listen(INADDR_ANY, listenPort, 5);
    printf("Listening for incoming connections on port %d\r\n", listenPort);

    while (1)
    {
        // Declare file descriptor sets for select()
        fd_set readFds;
        fd_set errorFds;

        // Init the fd sets
        FD_ZERO(&readFds);
        FD_ZERO(&errorFds);

        // Configure the fd sets so select() knows what to process. In this case any fd data to be read, or
        // an error
        for (int32_t i = 0; i < maxConnections; i++)
        {
            if (fdArray[i])    // The fd in the array will be > 0 if open and valid, so reset the file
                descriptor sets
            {
                FD_SET(fdArray[i], &readFds);
                FD_SET(fdArray[i], &errorFds);
            }
        }

        // select() should also process the listen fd
        FD_SET(fdListen, &readFds);
        FD_SET(fdListen, &errorFds);

        /* select() will block until any fd has data to be read, or has an error. When select() returns,
           readFds and/or errorFds variables will have been modified to reflect the events.
        */
        select(FD_SETSIZE, &readFds, (fd_set *)0, &errorFds, 0);

        // If the listen fd has a connection request, accept it.
        if (FD_ISSET(fdListen, &readFds))
        {
            IPADDR clientIp;
            uint16_t clientPort;

            int fdAccept = accept(fdListen, &clientIp, &clientPort, 0);

            // If accept() returned, find an open fd array slot
            if (fdAccept > 0)
            {
                for (int i = 0; i < maxConnections; i++)
                {
                    if (fdArray[i] == 0)
                    {
                        fdArray[i] = fdAccept;
                        writestring(fdAccept, "Welcome to the NetBurner Multi-Socket TCP Server! 'Q' to
quit.");
                        printf("Added connection on fd[%d] = %d, Client IP: %I:%d\r\n", i, fdAccept,
                            GetSocketRemoteAddr(fdAccept), GetSocketRemotePort(fdAccept));
                        fdAccept = 0;
                        break;
                    }
                }
            }
        }
    }
}

```

```

    }

    // If no array positions are open, close the connection
    if (fdAccept)
    {
        writestring(fdAccept, "I am sorry, but the server is full\r\n");
        printf("Server Full\r\n");
        close(fdAccept);
    }
}

// If the listen fd has an error, close it and reopen
if (FD_ISSET(fdListen, &errorFds))
{
    close(fdListen);
    fdListen = listen(INADDR_ANY, listenPort, 5);
}

// Process each fd array element and check it against readFds and errorFds.
for (int i = 0; i < maxConnections; i++)
{
    if (fdArray[i])
    {
        // Check for data to be read
        if (FD_ISSET(fdArray[i], &readFds))
        {
            char buffer[readBufferSize];
            int rv = read(fdArray[i], buffer, readBufferSize);
            if (rv > 0)
            {
                buffer[rv] = 0;
                if (buffer[0] == 'Q')
                {
                    printf("Closing connection fd[%d]\r\n", i);
                    writestring(fdArray[i], "Bye\r\n");
                    close(fdArray[i]);
                    fdArray[i] = 0;
                }
                else
                {
                    printf("Read \"%s\" from fd[%d]\r\n", buffer, i);
                    // snprintf( buffer, readBufferSize, "Server read %d byte(s)\r\n", rv );
                    // writestring( fdArray[i], buffer );
                }
            }
            else
            {
                printf("Read Error on fd[%d]\r\n", fdArray[i]);
                FD_SET(fdArray[i], &errorFds);
            }
        } // data available to read

        // Check for errors
        if (FD_ISSET(fdArray[i], &errorFds))
        {
            printf("Error on fd[%d], closing connection\r\n", i);
            close(fdArray[i]);
            fdArray[i] = 0;
        }
    } // if fd is valid
} // process each connection in the array
} // while (1)
} // UserMain

```

10.14.2.6 TCP Client Implementation

A TCP client initiates connections to TCP servers using the `connect()` function. This section demonstrates a web-based TCP client interface.

This is a simple TCP client application built for NetBurner hardware that demonstrates how to create and use a TCP client connection. The application provides a web-based interface that allows users to send messages to any TCP server by specifying the destination IP address, port number, and message content.

10.14.2.6.1 Features

- **Web-based Interface:** Complete interaction through a web page interface
- **Configurable Connection:** Set destination IP address and port number
- **Message Transmission:** Send custom messages to TCP servers
- **Auto-fill Functionality:** Automatically fills in the requesting client's IP address as default destination

- **Error Handling:** Comprehensive error reporting and logging
- **Real-time Feedback:** Connection status and transmission results displayed

10.14.2.6.2 User Main (main.cpp)

```
#include <init.h>
#include <nbrtos.h>
#include <ipshow.h>

const char *AppName = "Simple TCP Client Example";

/*-----
User Main
-----*/
void UserMain(void *pd)
{
    init();
    //Enable system diagnostics. Probably should remove for production code.
    EnableSystemDiagnostics();
    StartHttp();
    WaitForActiveNetwork(TICKS_PER_SECOND * 10);
    showIpAddresses();

    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

10.14.2.6.3 Client Web Page User Interface to Connect (clientweb.cpp)

```
/*NB_REVISION*/

/*NB_COPYRIGHT*/

/*-----
This code implements the web page entries for the message,
destination IP address and destination port number. When the
web page first loads it will automatically fill in the IP
address from the source requesting the web page, because in
most cases it will also be the address of the TCP Server.
The web page is a form, and when a user presses the submit
button the SendMsg() function will open a TCP connection to
the server, send the message, and close the connection.
Any error messages will be sent to stdout and can be viewed
with MTTY.

A TCP server program must already be listening at the specified
IP address and port number for the message to be sent. A simple
TCP Server called TcpServerWin.exe is located in the
\ncburn\pctools directory of your NetBurner installation.
-----*/

#include <iosys.h>
#include <tcp.h>
#include <fdprintf.h>
#include <httppost.h>

#define APP_VERSION "Version 1.3 7/29/2025"

#define MAX_MSG_SIZE 1024

int      gDestPort;
IPADDR   gDestIp;
char      gMessage[MAX_MSG_SIZE];

/*-----
Sends a message to the specified host.
-----*/
void SendMsg(IPADDR destIp, int destPort, char *msg)
{
    printf("Connecting to: %I:%d\r\n", destIp, destPort);
    int fd = connect(destIp, destPort, TICKS_PER_SECOND * 5);
    if (fd < 0) { printf("Error: Connection failed, return code: %d\r\n", fd); }
    else
    {
        printf("Attempting to write: \"%s\" \r\n", msg);
        int n = write(fd, msg, strlen(msg));
        printf("Wrote %d bytes\r\n", n);
        close(fd);
    }
}
```

```

}

/*-----
   Show destination port number on web page
   -----*/
void WebDestPort(int sock, PCSTR url)
{
    if (gDestPort == 0) // If no dest port is specified, use a default
        gDestPort = 2000;
    fprintf(sock, "VALUE=\"%d\" ", gDestPort);
}

/*-----
   Show destination ip address on web page
   -----*/
void WebDestIp(int sock, PCSTR url)
{
    if (gDestIp.IsNull()) // If no dest ip address has been entered, use the one that requested the web
        page.
        fprintf(sock, "VALUE=\"%I\" ", GetSocketRemoteAddr(sock));
    else
        fprintf(sock, "VALUE=\"%I\" ", gDestIp);
}

/*-----
   Show destination ip address on web page
   -----*/
void WebShowClientIp(int sock, PCSTR url)
{
    fprintf(sock, "%I", GetSocketRemoteAddr(sock));
}

/*-----
   Parse form post variables.
   -----*/
void processPostVariables(const char *pName, const char *pValue)
{
    printf("Processing: %s\r\n", pName);

    if (strcmp(pName, "tfDestPortNum") == 0)
    {
        gDestPort = (uint16_t)atoi(pValue);
        printf("Destination port set to: %d\r\n", gDestPort);
    }
    else if (strcmp(pName, "tfDestIpAddr") == 0)
    {
        printf("DestIpAddr set to: %s\r\n", pValue);
        gDestIp = AsciiToIp(pValue);
    }
    else if (strcmp(pName, "tfMessage") == 0)
    {
        strncpy(gMessage, pValue, MAX_MSG_SIZE - 1);
    }
    else if (strcmp(pName, "SendMessage") == 0)
    {
        SendMsg(gDestIp, gDestPort, (char *)pValue);
    }
    else
    {
        printf("Error processing %s\r\n", pName);
    }
}

/*-----
   Form post callback function
   -----*/
int PostCallBack(int sock, PostEvents event, const char *pName, const char *pValue)
{
    // Received a call back with an event, check for event type
    switch (event)
    {
        case eStartingPost: // Called at the beginning of the post before any data is sent
            break;

        case eVariable: // Called once for each variable in the form
            processPostVariables(pName, pValue);
            break;

        // Called back with a file handle if the post had a file
        case eFile:
            break; // No file type here so we do nothing

        // Called back when the post is complete. You should send your response here.
        case eEndOfPost:
        {
            printf("\r\n\r\n"); // just some new lines for easier console reading
            RedirectResponse(sock, "INDEX.HTML"); // Our response is to redirect to the index page
        }
    }
}

```

```

    }
    break;
} // Switch

return 0;
}

// Create a global static post handling object that responds to the specified html page.
// A separate post handler can be created for each form in your application.
HtmlPostVariableListCallback poster("index.html", PostCallBack);

```

10.14.2.7 TCP Error Handling

Status codes and error values returned by TCP socket operations. These error codes are returned by TCP socket functions such as `listen()`, `accept()`, `connect()`, `read()`, `write()`, and other socket operations. Understanding these codes is essential for proper error handling and debugging network issues.

10.14.2.7.1 Return Value Convention

TCP socket functions follow this convention:

- **Positive values:** Valid file descriptor (socket successfully created/connected)
- **Zero:** Success for non-FD operations (e.g., `TCP_ERR_NORMAL`)
- **Negative values:** Error condition (use these constants to identify the error)

10.14.2.7.2 Error Handling Pattern

```

int sock = connect("example.com", 80, TICKS_PER_SECOND * 10);
if (sock < 0) {
    // Error occurred - check which one
    switch (sock) {
        case TCP_ERR_TIMEOUT:
            printf("Connection timed out\r\n");
            break;
        case TCP_ERR_NONE_AVAIL:
            printf("No sockets available\r\n");
            break;
        default:
            printf("Connection error: %d\r\n", sock);
            break;
    }
} else {
    // Success - sock is a valid file descriptor
    printf("Connected on socket %d\r\n", sock);
}

```

10.14.2.7.3 Common Usage Scenarios

10.14.2.7.3.1 Client Connections

- `TCP_ERR_TIMEOUT`: Server unreachable or not responding
- `TCP_ERR_NONE_AVAIL`: Too many open connections
- `TCP_ERR_CON_RESET`: Server rejected connection

10.14.2.7.3.2 Server Operations

- `TCP_ERR_TIMEOUT`: `accept()` timeout (not necessarily an error)
- `TCP_ERR_NONE_AVAIL`: Socket pool exhausted

10.14.2.7.3.3 Data Transfer

- `TCP_ERR_NOCON`: Connection closed before operation
- `TCP_ERR_CLOSING`: Connection in teardown state
- `TCP_ERR_CON_RESET`: Remote host reset connection

Expand for Example Usage

Examples

Example 1: Basic Error Handling in Client Connection

```
int ConnectToServer(const char *host, uint16_t port) {
    printf("Connecting to %s:%d...\r\n", host, port);

    int sock = connect(host, port, TICKS_PER_SECOND * 10);

    if (sock == TCP_ERR_TIMEOUT) {
        printf("ERROR: Connection timed out\r\n");
        return -1;
    } else if (sock == TCP_ERR_NONE_AVAIL) {
        printf("ERROR: No sockets available\r\n");
        return -1;
    } else if (sock < 0) {
        printf("ERROR: Connection failed with code %d\r\n", sock);
        return -1;
    }

    printf("Successfully connected (socket %d)\r\n", sock);
    return sock;
}
```

Example 2: Server Accept with Timeout Handling

```
void ServerAcceptLoop(int listenSock) {
    while (1) {
        // Accept with 5 second timeout
        int clientSock = accept(listenSock, NULL, NULL, TICKS_PER_SECOND * 5);

        if (clientSock == TCP_ERR_TIMEOUT) {
            // Timeout is normal - just loop again
            continue;
        } else if (clientSock == TCP_ERR_NONE_AVAIL) {
            printf("WARNING: No sockets available for new connection\r\n");
            OSTimeDly(TICKS_PER_SECOND); // Wait before retry
            continue;
        } else if (clientSock < 0) {
            printf("ERROR: Accept failed with code %d\r\n", clientSock);
            break;
        }

        // Valid connection
        printf("New client connected on socket %d\r\n", clientSock);
        HandleClient(clientSock);
    }
}
```

Example 3: Comprehensive Error Reporting

```
const char* GetTCPErrString(int errorCode) {
    switch (errorCode) {
        case TCP_ERR_NORMAL:
            return "Success";
        case TCP_ERR_TIMEOUT:
            return "Operation timed out";
        case TCP_ERR_NOCON:
            return "No active connection";
        case TCP_ERR_CLOSING:
            return "Socket is closing";
        case TCP_ERR_NOSUCH_SOCKET:
            return "Socket does not exist";
        case TCP_ERR_NONE_AVAIL:
            return "No sockets available";
        case TCP_ERR_CON_RESET:
            return "Connection reset by peer";
        case TCP_ERR_CON_ABORT:
            return "Connection aborted";
        default:
            return "Unknown error";
    }
}

void ReportConnectionError(int errorCode, const char *operation) {
    printf("TCP Error during %s: %s (code: %d)\r\n",
        operation, GetTCPErrString(errorCode), errorCode);
}

// Usage
int sock = connect("server.com", 80, TICKS_PER_SECOND * 10);
if (sock < 0) {
    ReportConnectionError(sock, "connect");
}
```


Example 4: Read with Connection Error Detection

```

int ReadWithErrorHandling(int sock, char *buffer, int bufSize) {
    int bytesRead = read(sock, buffer, bufSize);

    if (bytesRead > 0) {
        // Data received successfully
        return bytesRead;
    } else if (bytesRead == 0) {
        // Connection closed gracefully
        printf("Connection closed by remote host\r\n");
        return 0;
    } else {
        // Error occurred
        if (bytesRead == TCP_ERR_NOCON) {
            printf("ERROR: No active connection\r\n");
        } else if (bytesRead == TCP_ERR_CLOSING) {
            printf("ERROR: Socket is closing\r\n");
        } else if (bytesRead == TCP_ERR_CON_RESET) {
            printf("ERROR: Connection reset by peer\r\n");
        } else if (bytesRead == TCP_ERR_TIMEOUT) {
            printf("ERROR: Read timeout\r\n");
        } else {
            printf("ERROR: Read failed with code %d\r\n", bytesRead);
        }
        return -1;
    }
}

```

Example 5: Socket Pool Management

```

#define MAX_RETRY_ATTEMPTS 3
int ConnectWithRetry(const char *host, uint16_t port) {
    for (int attempt = 1; attempt <= MAX_RETRY_ATTEMPTS; attempt++) {
        printf("Connection attempt %d/%d...\r\n", attempt, MAX_RETRY_ATTEMPTS);

        int sock = connect(host, port, TICKS_PER_SECOND * 5);

        if (sock >= 0) {
            // Success
            return sock;
        }

        if (sock == TCP_ERR_NONE_AVAIL) {
            printf("No sockets available, waiting for one to free up...\r\n");
            OSTimeDly(TICKS_PER_SECOND * 2); // Wait for sockets to free
        } else if (sock == TCP_ERR_TIMEOUT) {
            printf("Timeout, retrying...\r\n");
            OSTimeDly(TICKS_PER_SECOND); // Brief delay before retry
        } else {
            // Other error - don't retry
            printf("Fatal error %d, aborting\r\n", sock);
            return sock;
        }
    }

    printf("Failed to connect after %d attempts\r\n", MAX_RETRY_ATTEMPTS);
    return TCP_ERR_TIMEOUT;
}

```

Example 6: Connection State Monitoring

```

typedef struct {
    int sock;
    bool isConnected;
    uint32_t lastActivity;
    int lastError;
} ConnectionState;

void UpdateConnectionState(ConnectionState *conn) {
    // Try a zero-length read to check connection status
    char dummy;
    int result = read(conn->sock, &dummy, 0);

    if (result == TCP_ERR_NOCON || result == TCP_ERR_CON_RESET ||
        result == TCP_ERR_CON_ABORT) {
        if (conn->isConnected) {
            printf("Connection lost: %s\r\n", GetTCPErrString(result));
            conn->isConnected = false;
            conn->lastError = result;
        }
    } else if (result == TCP_ERR_CLOSING) {
        printf("Connection closing...\r\n");
        conn->isConnected = false;
        conn->lastError = result;
    } else {

```

```

        // Connection still active
        conn->lastActivity = Secs;
    }
}

```

Example 7: Graceful Shutdown on Error

```

void CloseConnectionGracefully(int sock) {
    if (sock < 0) {
        return; // Invalid socket
    }

    // Try to shutdown gracefully
    printf("Closing socket %d...\r\n", sock);

    // Attempt to send any pending data with timeout
    OSTimeDly(TICKS_PER_SECOND / 10); // 100ms grace period

    close(sock);
    printf("Socket closed\r\n");
}

void HandleConnectionError(int sock, int errorCode) {
    printf("Handling error: %s\r\n", GetTCPErrorString(errorCode));

    switch (errorCode) {
        case TCP_ERR_CON_RESET:
        case TCP_ERR_CON_ABORT:
            // Connection already gone, just close FD
            close(sock);
            break;

        case TCP_ERR_CLOSING:
            // Wait briefly for close to complete
            OSTimeDly(TICKS_PER_SECOND / 2);
            close(sock);
            break;

        default:
            CloseConnectionGracefully(sock);
            break;
    }
}

```

Example 8: Diagnostic Logging

```

void LogSocketOperation(const char *operation, int result) {
    if (result >= 0) {
        printf("[OK] %s: fd=%d\r\n", operation, result);
    } else {
        printf("[ERROR] %s: %s (code=%d)\r\n",
            operation, GetTCPErrorString(result), result);
    }
}

// Usage
int sock = connect("example.com", 80, TICKS_PER_SECOND * 10);
LogSocketOperation("connect", sock);
if (sock >= 0) {
    int bytes = write(sock, "GET /", 5);
    LogSocketOperation("write", bytes);
}

```

10.14.2.7.4 Functions that return TCP Status Codes:

Function	Description
connect()	Establish TCP client connection
listen()	Create listening TCP socket
accept()	Accept incoming TCP connection
read()	Read data from TCP socket
write()	Write data to TCP socket
close()	Close TCP socket

10.14.2.7.5 TCP Return Codes

TCP Socket Status and Error Codes are defined here: [TCP Socket Status](#). A summary is provided below and the following sections detail each error code as well as possible causes and corrective measures.

```
#define TCP_ERR_NORMAL      (0)    // Operation completed successfully with no errors
#define TCP_ERR_TIMEOUT    (-1)   // Operation timed out before completion
#define TCP_ERR_NOCON      (-2)   // No active connection exists on this socket
#define TCP_ERR_CLOSING    (-3)   // Socket is currently in the closing state
#define TCP_ERR_NOSUCH_SOCKET (-4) // The specified socket does not exist
#define TCP_ERR_NONE_AVAIL (-5)   // No sockets available in the system pool
#define TCP_ERR_CON_RESET  (-6)   // Connection was reset by remote host
#define TCP_ERR_CON_ABORT  (-7)   // Connection was aborted
```

10.14.2.7.5.1 Operation completed successfully with no errors

```
#define TCP_ERR_NORMAL (0)
```

Indicates that a socket operation completed without any error conditions. This is the expected return value for successful non-FD operations.

10.14.2.7.5.2 Operation timed out before completion

```
#define TCP_ERR_TIMEOUT (-1)
```

The socket operation did not complete within the specified timeout period. Common causes:

- Remote host unreachable or not responding
- Network congestion or packet loss
- Timeout value too short for operation
- Firewall blocking connection

For [accept\(\)](#) with timeout, this is not necessarily an error - it just means no connection arrived within the timeout period.

Note

When this occurs during [connect\(\)](#), the remote host is likely down or unreachable

For [read\(\)](#) operations, this may indicate the peer stopped sending data

10.14.2.7.5.3 No active connection exists on this socket

```
#define TCP_ERR_NOCON (-2)
```

Attempted to perform an operation on a socket that has no active connection. Common causes:

- Trying to read/write on a socket that was never connected
- Connection was closed by remote peer
- Connection attempt failed but socket was still used
- Using a socket after [close\(\)](#) was called

Note

Always check connection establishment before attempting I/O operations

This error indicates the socket exists but has no connection

10.14.2.7.5.4 Socket is currently in the closing state

```
#define TCP_ERR_CLOSING (-3)
```

The socket is in the process of closing and cannot accept new operations. This is a transitional state during connection teardown.

Common causes:

- [close\(\)](#) was called and TCP is performing graceful shutdown
 - FIN packet sent but waiting for acknowledgment
 - TCP TIME_WAIT or CLOSE_WAIT state
-

Note

Wait briefly and the socket will complete closing
Do not attempt further I/O on this socket

10.14.2.7.5.5 The specified socket does not exist

```
#define TCP_ERR_NOSUCH_SOCKET (-4)
```

The file descriptor does not correspond to a valid socket in the system. Common causes:

- Using an invalid file descriptor number
- Socket was already closed
- File descriptor number was never allocated
- Corruption of the socket handle

Note

This indicates a programming error - verify FD validity before use
Check that socket creation succeeded before using the FD

10.14.2.7.5.6 No sockets available in the system pool

```
#define TCP_ERR_NONE_AVAIL (-5)
```

The system has exhausted its pool of available socket resources and cannot create new connections. This is a resource limitation error.

Common causes:

- Too many simultaneous connections open
- Sockets not being closed properly (resource leak)
- Application exceeding design limits
- System configuration limits too low

Solutions:

- Close unused sockets promptly
- Implement connection pooling or limiting
- Increase system socket pool size if possible
- Check for socket leaks in application code

Warning

This error indicates a serious resource constraint

Note

Monitor socket usage to prevent this condition

10.14.2.7.5.7 Connection was reset by remote host

```
#define TCP_ERR_CON_RESET (-6)
```

The remote peer forcibly closed the connection, typically by sending a TCP RST (reset) packet. This indicates an abnormal connection termination.

Common causes:

- Remote application crashed
- Remote host rebooted
- Firewall or security device terminated connection
- Protocol violation or invalid data sent
- Remote application explicitly reset the socket

Note

This is different from a graceful close (FIN packet)

Data may have been lost when this occurs

Warning

Connection cannot be recovered - must establish new connection

10.14.2.7.5.8 Connection was aborted

```
#define TCP_ERR_CON_ABORT (-7)
```

The connection was terminated abnormally, typically due to a local or network error condition. This is a general abort condition.

Common causes:

- Local TCP stack detected fatal error
- Network path broken
- Excessive retransmission failures
- Local application requested abort
- System resources exhausted during transfer

Note

Similar to TCP_ERR_CON_RESET but indicates local-side abort

Warning

Connection cannot be recovered - must establish new connection

10.14.3 UDP Protocol**10.14.3.1 UDP Protocol**

The NetBurner API provides two methods to implement UDP: a C++ class interface and traditional socket functions. Both methods offer equivalent performance; choose based on your coding preference.

10.14.3.2 UDP Implementation Comparison

Class Interface

```
UDPPacket
.Send()
.Validate()
```

vs

Socket Interface

```
recvfrom()
sendto()
select()
```

Object-oriented

Function-based

10.14.3.3 UDP Class Interface

The `UDPPacket` class provides an object-oriented approach to UDP communication.

10.14.3.3.1 Receiving UDP Packets (Class Method)

Setup Process**:

1. Create an `OS_FIFO` to hold incoming packets
2. Register the FIFO with `RegisterUDPFifo()` for specific port(s)
3. Construct `UDPPacket` object to receive packets (blocking)
4. Validate packet with `Validate()` member function
5. Extract data with `GetDataBuffer()` member function

Multi-Port Listening**:

- Use separate `OS_FIFO` for each port, OR
- Use single `OS_FIFO` for multiple ports by calling `RegisterUDPFifo()` multiple times

10.14.3.3.2 Sending UDP Packets (Class Method)

Transmission Process**:

1. Declare `UDPPacket` instance
2. Set source and destination ports
3. Add data using member functions
4. Call `Send()` to transmit packet

10.14.3.3.3 UDP Class Example

This example demonstrates bidirectional UDP communication using the `UDPPacket` class.

```
/*
   UDP Send/Receive Example using UDPPacket Class

   Features:
   - Receives UDP packets and displays on serial port
   - Sends user-typed messages as UDP packets
   - Uses OS_FIFO for packet buffering
   - Demonstrates task-based architecture

   Testing:
   - Use NetBurner UDP Terminal on Windows
   - Configure matching IP addresses and ports
   - Messages sent from terminal appear in MTTY
   - Messages typed in MTTY sent via UDP
*/

#include <init.h>
#include <stdlib.h>
#include <system.h>
#include <udp.h>
#include <utils.h>

const char *AppName = "UDP Send/Receive Example";

// Allocate stack space for the listen task
uint32_t UdpReceiveTaskStack[USER_TASK_STK_SIZE];

/*
   UDP Receive Task
   Waits for incoming packets on specified port
*/
void UdpReceiveTask(void *pd)
{
    static OS_FIFO fifo;    // FIFO for incoming packets
```

```

int listenPort = (int)pd;

printf("Listening on UDP port: %d\r\n", listenPort);

// Register FIFO to receive packets on this port
RegisterUDPFifo(listenPort, &fifo);

while (1)
{
    // Construct UDP packet object (blocks until packet received)
    // Second parameter is timeout (0 = wait forever)
    UDPPacket upkt(&fifo, 0 * TICKS_PER_SECOND);

    // Check if valid packet received
    if (upkt.Validate())
    {
        uint16_t len = upkt.GetDataSize();
        printf("Received UDP packet with %d bytes from: %I\r\n",
            (int)len, upkt.GetSourceAddress());
        ShowData(upkt.GetDataBuffer(), len); // Hex dump
        printf("\r\n");
    }
}

/*
User Main
*/
void UserMain(void *pd)
{
    int portNumber;
    IPADDR destIpAddress;
    char buffer[80];

    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    printf("Application: %s\r\nNDK Revision: %s\r\n",
        AppName, GetReleaseTag());

    printf("Enter the UDP port number (send & receive): ");
    gets(buffer);
    portNumber = atoi(buffer);

    printf("\r\nEnter the destination IP Address: ");
    gets(buffer);
    destIpAddress = AsciiToIp(buffer);

    printf("Listening/Sending on UDP Port %d, Sending to: %I\r\n",
        portNumber, destIpAddress);

    // Create UDP receive task
    OSTaskCreateName(UdpReceiveTask,
        (void *)portNumber,
        &UdpReceiveTaskStack[USER_TASK_STK_SIZE],
        UdpReceiveTaskStack,
        MAIN_PRIO - 1, // Higher priority than UserMain
        "UDP Receive");

    // Main loop: send user input as UDP packets
    printf("Enter data and hit return to send.\r\n");
    while (1)
    {
        gets(buffer);

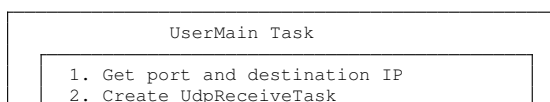
        UDPPacket pkt;
        pkt.SetSourcePort(portNumber);
        pkt.SetDestinationPort(portNumber);
        pkt.AddData(buffer);
        pkt.AddDataByte(0);
        pkt.Send(destIpAddress);
        printf("\r\n");

        printf("Sent \"%s\" to %I:%d\r\n", buffer, destIpAddress, portNumber);
    }
}

```

10.14.3.3.4 UDP Class Architecture

Task Architecture:



```

3. Loop:
    -> Read user input
    -> Create UDPPacket
    -> Set ports
    -> Add data
    -> Send()

```

UdpReceiveTask (Higher Priority)

```

1. Create OS_FIFO
2. RegisterUDPFifo(port, &fifo)
3. Loop:
    -> UDPPacket upkt(&fifo, timeout)
       [BLOCKS until packet received]
    -> Validate()
    -> GetDataBuffer()
    -> Display data

```

10.14.3.3.5 Key Class Methods

Sending**:

```

UDPPacket pkt;
pkt.SetSourcePort(localPort);
pkt.SetDestinationPort(remotePort);
pkt.AddData(dataBuffer);
pkt.Send(destinationIP);

```

Receiving**:

```

OS_FIFO fifo;
RegisterUDPFifo(port, &fifo);
UDPPacket upkt(&fifo, timeout);
if (upkt.Validate()) {
    uint8_t *data = upkt.GetDataBuffer();
    uint16_t len = upkt.GetDataSize();
    IPADDR source = upkt.GetSourceAddress();
}

```

10.14.3.4 UDP Sockets Interface

The socket-based UDP interface provides a traditional function-based approach to UDP communication.

10.14.3.4.1 Receiving UDP Packets (Socket Method)

Setup Process**:

1. Call `CreateRxUdpSocket()` to open listening socket (returns file descriptor)
2. Call `recvfrom()` to receive packets (blocks by default)
3. Optionally wrap `recvfrom()` in `select()` for timeout control

10.14.3.4.2 Sending UDP Packets (Socket Method)

Transmission Process**:

1. Call `CreateTxUdpSocket()` to open socket (returns file descriptor)
2. Call `sendto()` to transmit packets

10.14.3.4.3 UDP Socket Error Processing

These negative values are returned by UDP socket creation and operation functions to indicate specific error conditions. All successful operations return non-negative values (valid file descriptors ≥ 0 or byte counts ≥ 0).

```

#define UDP_ERR_NOSUCH_SOCKET    (-1)
#define UDP_ERR_NOTOPEN_TO_WRITE (-2)
#define UDP_ERR_NOTOPEN_TO_READ  (-3)

```


10.14.3.4.3.1 UDP_ERR_NOSUCH_SOCKET

Socket does not exist or operation failed

```
#define UDP_ERR_NOSUCH_SOCKET (-1)
```

Returned when:

- Socket creation fails (port conflict, resource exhaustion)
- Invalid socket descriptor used in operation
- Socket was closed but still being referenced
- System cannot allocate required resources

Common Scenarios:

Scenario 1: Port Already in Use

```
int sock1 = CreateRxUdpSocket(8080); // Success: sock1 = valid descriptor
int sock2 = CreateRxUdpSocket(8080); // Failure: returns UDP_ERR_NOSUCH_SOCKET
// Solution: Use different port or close sock1 first
```

Scenario 2: Using Closed Socket

```
int sock = CreateRxUdpSocket(5000);
close(sock);
int result = sendto(sock, data, len, ip, port); // Returns UDP_ERR_NOSUCH_SOCKET
// Solution: Don't use socket after closing
```

Scenario 3: Resource Exhaustion

```
// Creating too many sockets
for (int i = 0; i < 100; i++) {
    int sock = CreateRxUdpSocket(5000 + i);
    if (sock < 0) {
        printf("Failed at socket %d: UDP_ERR_NOSUCH_SOCKET\r\n", i);
        // Solution: Close unused sockets, reduce simultaneous socket count
        break;
    }
}
```

10.14.3.4.3.2 UDP_ERR_NOTOPEN_TO_WRITE

Socket not configured for sending operations

```
UDP_ERR_NOTOPEN_TO_WRITE (-2)
```

Returned when attempting to send data through a socket that was created in receive-only mode (using `CreateRxUdpSocket()`). The socket can only receive packets, not send them.

Common Scenarios:

Scenario 1: Sending on Receive-Only Socket

```
// Wrong: Create receive-only socket, then try to send
int sock = CreateRxUdpSocket(5000); // Receive-only!
uint8_t data[] = "Hello";
int result = sendto(sock, data, sizeof(data), destIP, 8080);
// Returns UDP_ERR_NOTOPEN_TO_WRITE

// Correct: Use transmit or bidirectional socket
int sock = CreateTxUdpSocket(destIP, 8080, 0); // Can send
int result = sendto(sock, data, sizeof(data), destIP, 8080); // Success
```

Scenario 2: Echo Server Without Bidirectional Socket

```
// Wrong: Echo server needs to send responses
int sock = CreateRxUdpSocket(5000); // Can only receive
uint8_t buf[512];
IPADDR clientIP;
uint16_t lp, rp;

int n = recvfrom(sock, buf, sizeof(buf), &clientIP, &lp, &rp); // OK
sendto(sock, buf, n, clientIP, rp); // Fails: UDP_ERR_NOTOPEN_TO_WRITE

// Correct: Use bidirectional socket
int sock = CreateRxTxUdpSocket(clientIP, rp, 5000); // Can send & receive
int n = recvfrom(sock, buf, sizeof(buf), &clientIP, &lp, &rp); // OK
sendto(sock, buf, n, clientIP, rp); // OK
```

Solution Patterns:

Pattern 1: Use Correct Socket Type

- For **servers that only listen**: `CreateRxUdpSocket()` is correct
- For **clients that only send**: `CreateTxUdpSocket()`
- For **request/response**: `CreateRxTxUdpSocket()`
- For **echo servers**: `CreateRxTxUdpSocket()` or use separate sockets

Pattern 2: Separate Sockets for Send/Receive

```
// Alternative: Use two separate sockets
int rxSock = CreateRxUdpSocket(5000); // For receiving
int txSock = CreateTxUdpSocket(destIP, destPort, 0); // For sending

// Receive on one socket
recvfrom(rxSock, buf, len, &srcIP, &lp, &rp);

// Send on different socket
sendto(txSock, data, dataLen, destIP, destPort);
```

Warning

This error does NOT mean the socket descriptor is invalid. The socket exists and may be used for receiving, just not sending.

10.14.3.4.3.3 UDP_ERR_NOTOPEN_TO_READ

Socket not configured for receiving operations

```
#define UDP_ERR_NOTOPEN_TO_READ (-3)
```

Returned when attempting to receive data from a socket that was created in transmit-only mode (using [CreateTxUdpSocket\(\)](#)). The socket can only send packets, not receive them.

Common Scenarios:**

Scenario 1: Receiving on Transmit-Only Socket**

```
// Wrong: Create transmit-only socket, then try to receive
IPADDR serverIP = AsciiToIp("192.168.1.100");
int sock = CreateTxUdpSocket(serverIP, 8080, 0); // Transmit-only!

uint8_t buffer[512];
IPADDR srcIP;
uint16_t lp, rp;
int n = recvfrom(sock, buffer, sizeof(buffer), &srcIP, &lp, &rp);
// Returns UDP_ERR_NOTOPEN_TO_READ

// Correct: Use receive or bidirectional socket
int sock = CreateRxUdpSocket(localPort); // Can receive
int n = recvfrom(sock, buffer, sizeof(buffer), &srcIP, &lp, &rp); // Success
```

Scenario 2: Request/Response Without Bidirectional Socket**

```
// Wrong: Client expecting response but using transmit-only socket
IPADDR serverIP = AsciiToIp("10.0.1.50");
int sock = CreateTxUdpSocket(serverIP, 6000, 0); // Transmit-only

// Send request
uint8_t request[] = "GET /data";
sendto(sock, request, sizeof(request), serverIP, 6000); // OK

// Try to receive response
uint8_t response[512];
IPADDR respIP;
uint16_t lp, rp;
int n = recvfrom(sock, response, sizeof(response), &respIP, &lp, &rp);
// Fails: UDP_ERR_NOTOPEN_TO_READ

// Correct: Use bidirectional socket for request/response
int sock = CreateRxTxUdpSocket(serverIP, 6000, 0); // Can send & receive
sendto(sock, request, sizeof(request), serverIP, 6000); // OK
int n = recvfrom(sock, response, sizeof(response), &respIP, &lp, &rp); // OK
```

Scenario 3: DNS Query (Typical Request/Response)**

```
// Wrong approach
IPADDR dnsServer = AsciiToIp("8.8.8.8");
int sock = CreateTxUdpSocket(dnsServer, 53, 0); // Can't receive response!

// Correct approach
int sock = CreateRxTxUdpSocket(dnsServer, 53, 0); // Full duplex

// Send DNS query
uint8_t query[512];
int qlen = BuildDnsQuery(query, "example.com");
sendto(sock, query, qlen, dnsServer, 53);

// Receive DNS response
uint8_t response[512];
IPADDR srcIP;
uint16_t lp, rp;
int rlen = recvfrom(sock, response, sizeof(response), &srcIP, &lp, &rp);
```

Solution Patterns:**

Pattern 1: Choose Appropriate Socket Type**

- One-way transmission: [CreateTxUdpSocket\(\)](#) is correct

- **One-way reception:** [CreateRxUdpSocket\(\)](#) is correct
- **Request/Response:** [CreateRxTxUdpSocket\(\)](#) required
- **Client expecting replies:** [CreateRxTxUdpSocket\(\)](#) required

Pattern 2: Evaluate Communication Pattern**

```
// Ask: "Does this need to receive?"

// NO (logging, notifications, broadcast) -> CreateTxUdpSocket()
int logSock = CreateTxUdpSocket(logServer, 514, 0);
sendto(logSock, logMsg, len, logServer, 514);

// YES (queries, RPC, protocols with ACK) -> CreateRxTxUdpSocket()
int rpcSock = CreateRxTxUdpSocket(server, 8080, 0);
sendto(rpcSock, request, reqLen, server, 8080);
recvfrom(rpcSock, response, respLen, &srcIP, &lp, &rp);
```

Warning

This error does NOT mean the socket descriptor is invalid. The socket exists and may be used for sending, just not receiving.

Note

If you need truly one-way fire-and-forget communication, [CreateTxUdpSocket\(\)](#) is more efficient than [CreateRxTxUdpSocket\(\)](#) because it doesn't allocate receive buffers. However, most protocols benefit from at least basic acknowledgment, making bidirectional sockets the better choice.

See also

[CreateRxUdpSocket\(\)](#) - For sockets that can receive
[CreateRxTxUdpSocket\(\)](#) - For bidirectional communication
[recvfrom\(\)](#) - Function that may return this error

10.14.3.4.3.4 Error Handling Pattern

Always check return values from UDP socket functions:

```
int sock = CreateRxUdpSocket(5000);
if (sock < 0) {
    switch (sock) {
        case UDP_ERR_NOSUCH_SOCKET:
            printf("Socket does not exist or creation failed\r\n");
            break;
        case UDP_ERR_NOTOPEN_TO_WRITE:
            printf("Socket not configured for sending\r\n");
            break;
        case UDP_ERR_NOTOPEN_TO_READ:
            printf("Socket not configured for receiving\r\n");
            break;
        default:
            printf("Unknown error: %d\r\n", sock);
            break;
    }
    return; // Don't proceed with invalid socket
}

// Safe to use socket
uint8_t buffer[1024];
IPADDR srcIP;
uint16_t lp, rp;
int bytesReceived = recvfrom(sock, buffer, sizeof(buffer), &srcIP, &lp, &rp);
```

10.14.3.4.3.5 Common Causes and Solutions

UDP_ERR_NOSUCH_SOCKET

Causes:

- Port already in use by another socket
- System resource exhaustion (too many sockets)

- Invalid socket descriptor passed to function
- Socket was closed but still being referenced

Solutions:

- Check that port is not already bound
- Close unused sockets to free resources
- Verify socket is valid before operations
- Use different port numbers for different sockets

```
// BAD: Port conflict
int sock1 = CreateRxUdpSocket(8080);
int sock2 = CreateRxUdpSocket(8080); // Returns UDP_ERR_NOSUCH_SOCKET

// GOOD: Different ports or close first socket
int sock1 = CreateRxUdpSocket(8080);
// ... use sock1 ...
close(sock1);
int sock2 = CreateRxUdpSocket(8080); // Now succeeds
```

UDP_ERR_NOTOPEN_TO_WRITE

Causes

- Attempting `sendto()` on receive-only socket (`CreateRxUdpSocket`)
- Socket was created for receiving but not transmitting
- Socket configuration flags incorrect

Solutions:**

- Use `CreateTxUdpSocket()` or `CreateRxTxUdpSocket()` for sending
- Create separate sockets for different operations
- Verify socket type matches intended operation

```
// BAD: Using receive-only socket for sending
int rxSock = CreateRxUdpSocket(5000);
sendto(rxSock, data, len, destIP, destPort); // Returns UDP_ERR_NOTOPEN_TO_WRITE

// GOOD: Use appropriate socket type
int txSock = CreateTxUdpSocket(destIP, destPort, 0);
sendto(txSock, data, len, destIP, destPort); // Succeeds

// OR: Use bidirectional socket
int rxtxSock = CreateRxTxUdpSocket(destIP, destPort, 5000);
sendto(rxtxSock, data, len, destIP, destPort); // Succeeds
```

UDP_ERR_NOTOPEN_TO_READ

Causes

- Attempting `recvfrom()` on transmit-only socket (`CreateTxUdpSocket`)
- Socket was created for sending but not receiving
- Socket configuration flags incorrect

Solutions:**

- Use `CreateRxUdpSocket()` or `CreateRxTxUdpSocket()` for receiving
- Create separate sockets for different operations
- Verify socket type matches intended operation

```
// BAD: Using transmit-only socket for receiving
IPADDR serverIP = AsciiToIp("192.168.1.100");
int txSock = CreateTxUdpSocket(serverIP, 8080, 0);
recvfrom(txSock, buffer, len, &srcIP, &lp, &rp); // Returns UDP_ERR_NOTOPEN_TO_READ

// GOOD: Use appropriate socket type
int rxSock = CreateRxUdpSocket(8080);
recvfrom(rxSock, buffer, len, &srcIP, &lp, &rp); // Succeeds

// OR: Use bidirectional socket for request/response
int rxtxSock = CreateRxTxUdpSocket(serverIP, 8080, 0);
sendto(rxtxSock, request, reqLen, serverIP, 8080);
recvfrom(rxtxSock, buffer, len, &srcIP, &lp, &rp); // Both succeed
```

10.14.3.4.3.6 Error Checking Best Practices

1. Always Validate Socket Creation

```
int sock = CreateRxUdpSocket(port);
if (sock < 0) {
    // Handle error appropriately
    printf("Socket creation failed: %d\r\n", sock);
    return ERROR_CODE; // Don't continue
}
// Safe to use sock
```

2. Check Operation Return Values

```
int bytesSent = sendto(sock, data, len, destIP, destPort);
if (bytesSent < 0) {
    printf("Send failed: error %d\r\n", bytesSent);
} else if (bytesSent != len) {
    printf("Warning: Partial send (%d of %d bytes)\r\n", bytesSent, len);
}
```

3. Defensive Programming

```
// Store error code in variable for debugging
int result = CreateRxUdpSocket(port);
if (result < 0) {
    // Log detailed error information
    printf("CreateRxUdpSocket(%d) failed with error %d at %s:%d\r\n",
        port, result, __FILE__, __LINE__);

    // Take appropriate recovery action
    switch (result) {
        case UDP_ERR_NOSUCH_SOCKET:
            // Maybe try alternative port
            result = CreateRxUdpSocket(port + 1);
            break;
        // ... other cases ...
    }
}
```

4. Resource Cleanup on Error

```
int sock1 = CreateRxUdpSocket(5000);
if (sock1 < 0) {
    return ERROR_SOCKET1;
}

int sock2 = CreateRxUdpSocket(5001);
if (sock2 < 0) {
    close(sock1); // Clean up first socket
    return ERROR_SOCKET2;
}

// Both sockets valid
// ... use sockets ...

// Cleanup
close(sock1);
close(sock2);
```

10.14.3.4.3.7 Debugging Errors

When encountering UDP errors:

1. Enable System Diagnostics

```
void UserMain(void* pd) {
    init();
    EnableSystemDiagnostics(); // Shows resource usage
    // ... rest of code ...
}
```

2. Check Resource Availability

- Monitor socket count
- Check buffer pool usage
- Verify network interface status

3. Test with Simple Cases

- Start with single socket
- Verify basic send/receive works
- Add complexity incrementally

4. Use Network Tools

- Wireshark to capture packets
- netcat (nc) to test connectivity
- Verify firewall settings

10.14.3.4.3.8 Related Functions

Functions that may return these error codes:

- [CreateRxUdpSocket\(\)](#) - Can return UDP_ERR_NOSUCH_SOCKET
- [CreateTxUdpSocket\(\)](#) - Can return UDP_ERR_NOSUCH_SOCKET
- [CreateRxTxUdpSocket\(\)](#) - Can return UDP_ERR_NOSUCH_SOCKET
- [sendto\(\)](#) - Can return UDP_ERR_NOTOPEN_TO_WRITE or UDP_ERR_NOSUCH_SOCKET
- [recvfrom\(\)](#) - Can return UDP_ERR_NOTOPEN_TO_READ or UDP_ERR_NOSUCH_SOCKET
- [sendtovia\(\)](#) - Can return UDP_ERR_NOTOPEN_TO_WRITE or UDP_ERR_NOSUCH_SOCKET

10.14.3.4.4 UDP Sockets Example

This example demonstrates UDP communication using socket functions instead of the [UDPPacket](#) class.

UDP Sockets Send/Receive Example

Features:

- Bidirectional UDP communication
- Task-based receive handling
- Main loop for sending

```
#include <init.h>
#include <stdlib.h>
#include <string.h>
#include <system.h>
#include <udp.h>
#include <utils.h>

const char *AppName = "UDP Sockets Example";

// Allocate stack space for the listen task
uint32_t UdpReceiveTaskStack[USER_TASK_STK_SIZE];

//
// UDP Receive Task
// Waits for incoming UDP packets and processes them
//
void UdpReceiveTask(void *pd)
{
```

```

int listenPort = (int)pd;

printf("UdpReceiveTask monitoring port %d\r\n", listenPort);

// Create UDP socket for receiving
int udpFd = CreateRxUdpSocket(listenPort);
if (udpFd <= 0)
{
    printf("Error Creating UDP Listen Socket: %d\r\n", udpFd);
    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
else
{
    printf("Listening for UDP packets on port %d\r\n", listenPort);

    while (1)
    {
        IPADDR sourceIpAddress;    // Packet source IP
        uint16_t localPort;        // Local port (destination)
        uint16_t sourcePort;       // Source port
        char buffer[80];

        int len = recvfrom(udpFd, (uint8_t *)buffer, 80,
                           0, &sourceIpAddress, &localPort, &sourcePort);
        buffer[len] = '\0';

        printf("\r\nReceived a UDP packet with %d bytes from: %I\r\n%s\r\n",
               len, sourceIpAddress, buffer);
    }
}

//
// User Main
//
void UserMain(void *pd)
{
    int portNumber;
    IPADDR destIpAddress;
    char buffer[80];

    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    printf("Application: %s\r\nNNDK Revision: %s\r\n", AppName, GetReleaseTag());

    // Get destination IP address
    printf("Enter the UDP Server destination IP address: ");
    gets(buffer);
    destIpAddress = AsciiToIp(buffer);
    printf("\r\n");

    // Get port number (used for send and receive)
    printf("Enter the source/destination port number: ");
    gets(buffer);
    portNumber = atoi(buffer);
    printf("\r\n");

    // Create UDP socket for sending
    int udpFd = CreateTxUdpSocket(destIpAddress, portNumber, portNumber);
    if (udpFd <= 0)
    {
        printf("Error Creating UDP Socket: %d\r\n", udpFd);
        while (1)
        {
            OSTimeDly(TICKS_PER_SECOND);
        }
    }
    else
    {
        printf("Sending/Receiving with host %I:%d\r\n", destIpAddress, portNumber);
    }

    // Create UDP receive task (higher priority for responsive reception)
    OSTaskCreateName(UdpReceiveTask,
                    (void *)portNumber,
                    &UdpReceiveTaskStack[USER_TASK_STK_SIZE],
                    UdpReceiveTaskStack,
                    MAIN_PRIO - 1,
                    "UDP Receive");

    printf("Enter data and hit enter to send.\r\n");

    // Main loop: send user input as UDP packets

```

```

while (1)
{
    gets(buffer);
    printf("\r\n");
    printf("Sending \"%s\" using UDP to %I:%d\r\n", buffer, destIpAddress, portNumber);

    sendto(udpFd, (uint8_t *)buffer, strlen(buffer), destIpAddress, portNumber);
    printf("\r\n");
}
}

```

10.14.3.4.5 Socket Function Reference

Receiving**:

```
int udpFd = CreateRxUdpSocket(localPort);
```

```
IPADDR sourceIP;
uint16_t sourcePort, destPort;
char buffer[SIZE];
```

```
int len = recvfrom(udpFd, (uint8_t*)buffer, SIZE,
                  &sourceIP, &destPort, &sourcePort);
```

Sending**:

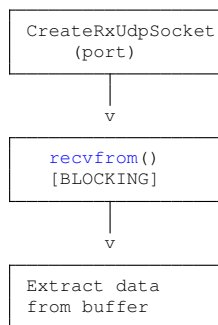
```
int udpFd = CreateTxUdpSocket(destIP, destPort, sourcePort);
```

```
sendto(udpFd, (uint8_t*)buffer, length, destIP, destPort);
```

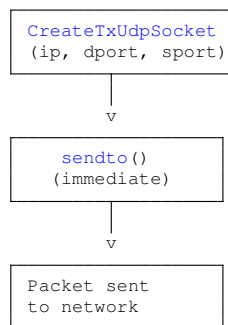
10.14.3.4.6 UDP Socket Architecture

Socket-Based UDP Flow:

Receive Path:



Send Path:



10.14.3.4.7 Best Practices Summary

10.14.3.4.8 General Guidelines

1. Protocol Selection:

- Choose TCP for reliability and ordered delivery
- Choose UDP for speed and efficiency
- Consider application-layer reliability over UDP when appropriate

2. Resource Management:

- Always close sockets when done
- Check return values from all network functions
- Handle connection errors gracefully

3. Task Architecture:

- Use separate tasks for listening/receiving
- Set appropriate task priorities
- Consider using `select()` for multiple connections

4. Buffer Management:

- Allocate adequate buffer sizes
- Null-terminate string data
- Use static buffers in tasks to save stack space

5. Error Handling:

- Check socket creation return values
- Handle connection failures
- Implement timeout mechanisms

10.14.3.4.9 Performance Optimization

- Use `select()` for multiple connections (TCP)
- Consider OS_FIFO size for UDP packet bursts
- Set appropriate task priorities for responsive I/O
- Use `ReadWithTimeout()` for connection management

10.14.3.4.10 Security Considerations

- Validate all received data
- Implement authentication when needed
- Use random source port numbers
- Consider connection limits for servers

10.15 SSL/TLS and Certificates

10.15.1 Introduction

This comprehensive guide covers SSL/TLS security implementation on NetBurner embedded devices. Whether you're securing a web interface, IoT communication, or industrial control system, this guide provides practical solutions for establishing encrypted, authenticated connections.

What You'll Learn:**

- Core SSL/TLS concepts explained clearly
- Multiple implementation approaches from simple to advanced
- Certificate management strategies
- Common issues and their solutions
- Production-ready deployment practices

10.15.2 Understanding SSL/TLS Fundamentals

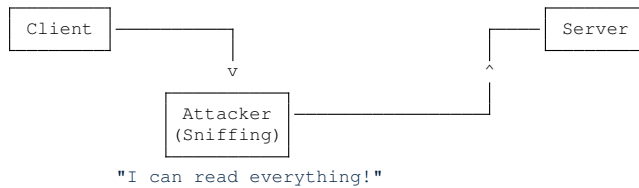
Note: This document uses "SSL" as shorthand for SSL/TLS protocols. Modern implementations use TLS (Transport Layer Security), the successor to SSL.

10.15.3 The Security Problem

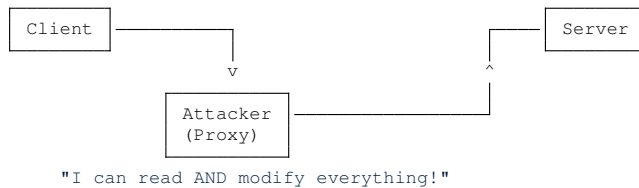
In a normal network connection, data travels in plain text. This creates two critical vulnerabilities:



Vulnerability 1: Eavesdropping**



Vulnerability 2: Man-in-the-Middle (MITM)**



10.15.4 How SSL/TLS Solves These Problems

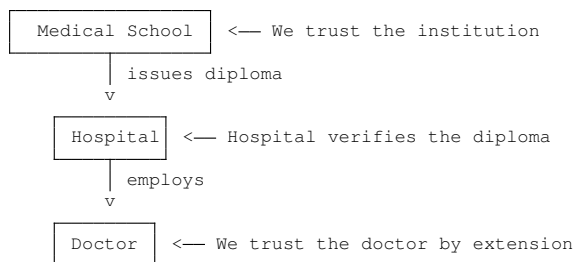
SSL/TLS provides three critical security features:

1. **Encryption:** Data is scrambled so eavesdroppers see gibberish
2. **Authentication:** Verify you're talking to the real server (not an attacker)
3. **Integrity:** Detect if data has been tampered with

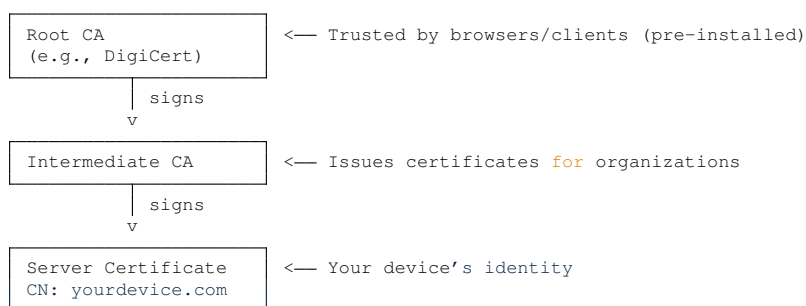
10.15.5 The Trust Chain: How Authentication Works

SSL/TLS uses a "chain of trust" similar to how we trust professionals:

10.15.5.1 Real-World Analogy: Trusting a Doctor



10.15.5.2 SSL/TLS Certificate Chain



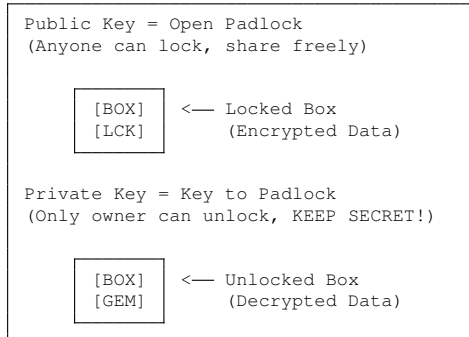
Key Components of a Certificate:**

1. **Identity** (Common Name/CN) - Who this certificate represents

2. **Public Key** - Like an open padlock anyone can use to encrypt
3. **Signature** - Cryptographic proof from a trusted CA

10.15.6 Public Key Cryptography Simplified

Think of it like a special mailbox:



Critical Security Principle:**

[LOCK] **The system is only as secure as the private key.** If an attacker gets your private key, they can decrypt all traffic and impersonate your device.

10.15.7 Certificate Types at a Glance

Type	Trust Level	Browser Warnings	Best For	Cost
Self-Signed	Low	[!] Yes	Development, testing, internal networks	Free
Private CA	Medium	[!] Yes (until CA installed)	Enterprise deployments, controlled environments	Free
Public CA	High	[OK] No	Production, public-facing devices	Varies
ACME (Let's Encrypt)	High	[OK] No	Production with public DNS	Free

10.15.8 Quick Start Guide

10.15.9 For SSL Servers (Your NetBurner Device)

10.15.9.1 Option 1: Automatic (Recommended for Production)

If you have a public DNS name:**

```
// ACME with Let's Encrypt - Automatic trusted certificates
#include <acmeservlet.h>
```

```
void UserMain(void *pd) {
    init();

    // Configure ACME
    AcmeConfig config;
    config.domain = "mydevice.example.com";
    config.email = "admin@example.com";

    // Automatic certificate issuance and renewal
    StartAcmeHttps(config);

    WaitForActiveNetwork(TICKS_PER_SECOND * 5);
    printf("Secure HTTPS server running\n");

    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}
```

If you only have local/private network:**

```
// Auto-generated self-signed certificate
#include <init.h>
#include <startnet.h>
```

```
void UserMain(void *pd) {
    init();
```

```
// Automatically generates self-signed cert on first run
StartHttps(); // Port 443 by default

WaitForActiveNetwork(TICKS_PER_SECOND * 5);
printf("HTTPS server running (self-signed)\n");

while(1) { OSTimeDly(TICKS_PER_SECOND); }
}
```

10.15.10 For SSL Clients (Connecting to Servers)

```
#include <ssl.h>
#include <nettcp.h>

// CA certificates you trust (example: Let's Encrypt root)
extern const char letsEncryptCA[];
extern const int letsEncryptCA_len;

void ConnectSecurely() {
    // Initialize SSL system
    SslInit();

    // Add trusted CA certificates
    SSL_AddCertToClientCaList(letsEncryptCA, letsEncryptCA_len);

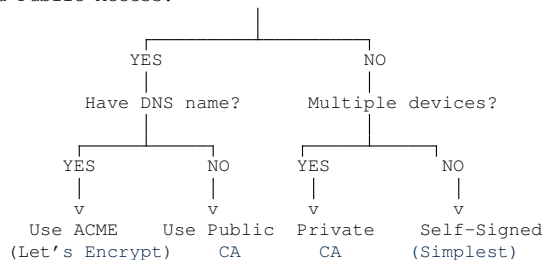
    // Create secure connection
    int fd = connect("api.example.com", 443);
    WOLFSSL* ssl = SSL_connect(fd);

    if (ssl) {
        SSL_write(ssl, "GET / HTTP/1.1\r\n\r\n", 18);
        // ... read response ...
        SSL_close(ssl);
    }
}
```

10.15.11 Certificate Types and Use Cases

10.15.12 Decision Matrix

Need Public Access?



10.15.13 1. Self-Signed Certificates

What It Is:** Device acts as its own Certificate Authority

Visual Flow:**

NetBurner Device

1. Generate Key Pair
2. Create Certificate
3. Sign with Own Key
4. Use **for** Connections

v

Browser/Client

```
[!] Certificate Warning
"Not Trusted"

[Proceed Anyway]
```

When to Use:**

- [YES] Development and testing
- [YES] Single device, few users

- [YES] Internal network only
- [YES] Quick prototyping
- Limitations:**
- [NO] Browser warnings for every user
- [NO] Certificate must be manually installed on each client
- [NO] No central management
- [NO] Looks unprofessional in production

NetBurner Implementation:**

```
// Automatic - just enable HTTPS
StartHttps(); // Self-signed cert auto-generated

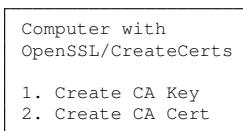
// Or manual control
#define ENABLE_AUTOCERT_REGEN // In predef.h
// Regenerates when expired (checked every minute by default)
```

10.15.14 2. Private Certificate Authority (CA)

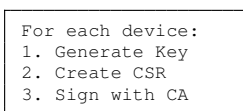
What It Is:** You create your own CA and sign device certificates

Visual Flow:**

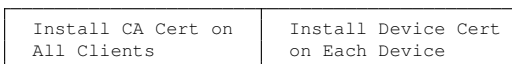
Step 1: Create Your CA



Step 2: Sign Device Certs



Step 3: Deploy



When to Use:**

- [YES] Multiple devices in controlled environment
- [YES] Corporate/enterprise deployment
- [YES] Need certificate management
- [YES] Air-gapped or private networks

Advantages:**

- [YES] One-time CA installation for all clients
- [YES] Central control over certificate issuance
- [YES] Can revoke compromised certificates
- [YES] No external dependencies

Limitations:**

- [NO] More complex initial setup
- [NO] Must manually manage certificate expiration

- [NO] CA certificate must be distributed to all clients

NetBurner Implementation:**

Use the provided scripts in \nburn\CreateCerts\:

```
# Step 1: Create your CA (one time)
cd C:\nburn\CreateCerts\ECDsa
createCA.bat

# Step 2: Create device certificate
createDeviceCert.bat mydevice.example.com

# Step 3: Convert to C++ for NetBurner
compile device.key comp_key comp_key_len key.cpp
compile device.crt comp_cert comp_cert_len cert.cpp
// In your application
extern const unsigned char comp_cert[];
extern const int comp_cert_len;
extern const unsigned char comp_key[];
extern const int comp_key_len;

void UserMain(void *pd) {
    init();

    // Install certificate before starting HTTPS
    SSL_InstallCertificate(comp_cert, comp_cert_len,
                          comp_key, comp_key_len);

    StartHttps();

    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}
```

10.15.15 3. Public CA Certificates

What It Is:** Purchase certificates from trusted authorities (DigiCert, GoDaddy, etc.)

Visual Flow:**

Step 1: Generate CSR

```
Device/Computer
1. Create Key
2. Generate CSR
```

v

Step 2: Submit to CA

```
Certificate Authority
(DigiCert, etc.)
1. Verify Domain
2. Issue Certificate
```

v

Step 3: Install

```
NetBurner Device
1. Install Cert
2. Install Chain
```

When to Use:**

- [YES] Public-facing production devices
- [YES] Need universal trust (no warnings)
- [YES] Customer/client access
- [YES] Professional appearance required

Advantages:**

- [YES] No browser warnings
- [YES] Pre-trusted by all clients
- [YES] Professional appearance

- [YES] Insurance/warranty often included

Limitations:**

- [NO] Annual costs (\$50-\$500+)
- [NO] Manual renewal process
- [NO] Requires public DNS name
- [NO] Validation requirements

Process:**

1. Generate Certificate Signing Request (CSR):

```
openssl genrsa -out device.key 2048
openssl req -new -key device.key -out device.csr
# Enter device domain name as Common Name
```

2. Submit CSR to CA and complete verification

3. Receive certificate and chain files

4. Combine certificate with chain:

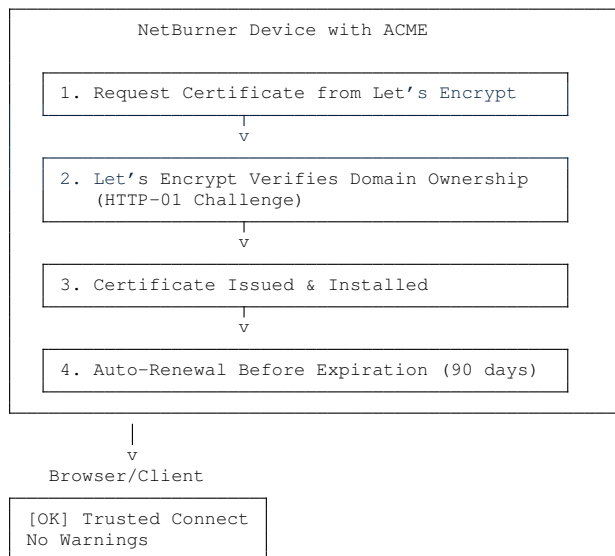
```
# Create complete chain file
cat device.crt intermediate.crt root.crt > device_chain.crt
```

5. Convert and install on NetBurner (same as Private CA method)

10.15.16 4. ACME (Automated Certificate Management)

What It Is:** Automated certificate issuance from Let's Encrypt or BuyPass

Visual Flow:**



When to Use:**

- [YES] Production devices with public DNS
 - [YES] Need automatic renewal
 - [YES] Want trusted certs without cost
 - [YES] Device accessible from internet
- Requirements:**
- [YES] Public DNS name pointing to device
 - [YES] Device reachable on port 80 (for validation)

- [YES] NNDK 3.5.0 or later
- Advantages:**
- [YES] Completely automatic
- [YES] Free trusted certificates
- [YES] Auto-renewal every 60 days
- [YES] No manual certificate management

Limitations:**

- [NO] Requires public internet access
- [NO] Must have DNS name (no IP-only)
- [NO] Shorter validity (90 days vs 1 year)

NetBurner Implementation:**

```
#include <acmeservlet.h>

void UserMain(void *pd) {
    init();
    EnableSystemDiagnostics();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Configure ACME settings
    AcmeConfig config;
    config.domain = "iot-device.example.com";
    config.email = "admin@example.com";
    config.acmeServer = ACME_LETSENCRYPT_PRODUCTION;

    // Start HTTPS with automatic ACME management
    int result = StartAcmeHttps(config);

    if (result == 0) {
        printf("ACME HTTPS server started successfully\n");
        printf("Certificate will auto-renew before expiration\n");
    } else {
        printf("ACME setup failed: %d\n", result);
    }

    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}
```

10.15.17 Use Case Decision Table

Scenario	Recommended Method	Why
Development/Testing	Self-Signed	Fastest, simplest
Internal corporate network, 10+ devices	Private CA	Central management, one-time client setup
Public website/API	ACME or Public CA	Trusted by all browsers
Air-gapped system	Private CA or Self-Signed	No external dependencies
IoT fleet with DNS names	ACME	Automatic, scalable, free
Single device, few users	Self-Signed	Adequate for small scale
Requires IP address (no DNS)	Self-Signed or Private CA	ACME requires DNS
Critical infrastructure	Public CA	Professional support, insurance

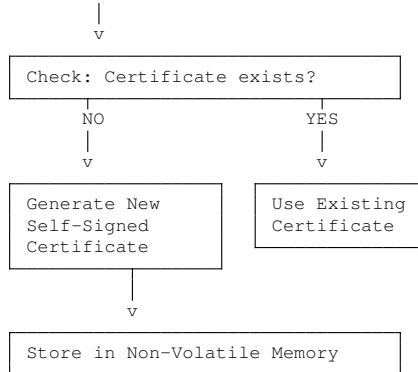
10.15.18 Implementation Methods

10.15.19 Method 1: Automatic Onboard Certificate Generation

Simplest approach - certificates generated automatically on first use.**

10.15.19.1 How It Works

First Boot / Certificate Needed



10.15.19.2 Basic Implementation

```

#include <init.h>

const char *AppName = "Secure Device";

void UserMain(void *pd) {
    init();
    EnableSystemDiagnostics();

    // Certificate automatically generated on first HTTPS connection
    StartHttps(); // Default port 443

    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    printf("HTTPS Server: %s\n", AppName);
    printf("Certificate auto-generated if needed\n");

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

10.15.19.3 Advanced: Auto-Regeneration

Enable automatic regeneration when certificates expire:

```

// In your project's predef.h or compile flags:
#define ENABLE_AUTOCERT_REGEN
#define AUTO_CERT_GEN_CHECK (60 * 20) // Check every 20 minutes

// Default settings if not defined:
// - Checks every 1 minute
// - Certificate valid for 1 year
// - Uses SECP384R1 ECC by default

```

10.15.19.4 Customizing Certificate Generation

```

#include <ssl.h>

// Callback for custom certificate attributes
void CustomCertCallback(SslCertAttributes_t *attrs) {
    // Set custom fields
    attrs->country = "US";
    attrs->state = "California";
    attrs->locality = "San Diego";
    attrs->organization = "My Company";
    attrs->commonName = "mydevice.local";

    // Add alternative names
    attrs->altNames = "mydevice.local,192.168.1.100,#LOCAL";
    // #LOCAL = automatically use mDNS .local name
}

void UserMain(void *pd) {
    init();

    // Register custom certificate callback
    SSL_SetCertificateCallback(CustomCertCallback);

    StartHttps();
}

```

```

    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}

```

10.15.20 Method 2: ACME Automated Certificates

Production-ready trusted certificates with automatic renewal.**

10.15.20.1 Complete ACME Example

```

#include <init.h>
#include <acmeservlet.h>
#include <nbrtos.h>

const char *AppName = "ACME Secure Device";

void UserMain(void *pd) {
    init();
    EnableSystemDiagnostics();

    WaitForActiveNetwork(TICKS_PER_SECOND * 10);

    // Ensure DNS is working (critical for ACME)
    printf("Device IP: %s\n", GetIP().c_str());
    printf("Waiting for DNS resolution...\n");

    // Configure ACME
    AcmeConfig config;
    config.domain = "iot-device.example.com"; // Must match DNS
    config.email = "admin@example.com";      // For renewal notices
    config.acmeServer = ACME_LETSENCRYPT_PRODUCTION;
    config.acceptTerms = true;

    // Optional: Use staging server for testing
    // config.acmeServer = ACME_LETSENCRYPT_STAGING;

    printf("Starting ACME certificate process for: %s\n", config.domain);

    int result = StartAcmeHttps(config);

    if (result == 0) {
        printf("[OK] ACME HTTPS server running\n");
        printf("Certificate valid for 90 days\n");
        printf("Auto-renewal occurs at 60 days\n");
    } else {
        printf("[ERR] ACME failed with error: %d\n", result);
        printf("Check: DNS resolution, firewall, port 80 access\n");
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND * 60);
        // ACME renewal happens automatically in background
    }
}

```

10.15.20.2 ACME Troubleshooting Checklist

Prerequisites Check:

```

[ ] Device has public IP address
[ ] DNS A record points to device IP
[ ] Port 80 open and accessible from internet
[ ] Port 443 open for final HTTPS
[ ] NTP time synchronized (critical!)
[ ] Sufficient free memory (>50KB)

```

Common Issues:

```

[X] DNS not resolving      -> Check DNS propagation (nslookup)
[X] Port 80 blocked       -> Check firewall rules
[X] Challenge fails       -> Verify HTTP-01 accessibility
[X] Time sync error       -> Enable and wait for NTP
[X] Memory allocation fail -> Check heap usage

```

10.15.21 Method 3: Manual Certificate Installation

For private CAs or purchased certificates.**

10.15.21.1 Step-by-Step Process

Step 1: Generate Certificate Files**

Using NetBurner CreateCerts scripts:

```
# Navigate to CreateCerts directory
cd C:\nburn\CreateCerts\ECDsa

# Create CA (one time, if needed)
createCA.bat

# Create device certificate
createDeviceCert.bat mydevice.example.com
```

```
# Output files:
# - device.key (private key - KEEP SECRET!)
# - device.crt (certificate)
# - ca.crt (CA certificate)
```

Step 2: Convert to C++ Code**

```
# Convert key
compfile device.key comp_key comp_key_len key.cpp

# Convert certificate
compfile device.crt comp_cert comp_cert_len cert.cpp

# Optional: Convert CA for chain
compfile ca.crt comp_ca comp_ca_len ca.cpp
```

Step 3: Add to Project**

```
YourProject/
├── src/
│   ├── main.cpp
│   ├── key.cpp      <- Add these
│   ├── cert.cpp     <- Add these
│   └── ca.cpp       <- Optional
└── makefile
```

Update makefile if using command line builds:

```
CPP_SRC += \
    src/main.cpp \
    src/key.cpp \
    src/cert.cpp
```

Step 4: Install in Code**

```
#include <init.h>
#include <ssl.h>

// External declarations for compiled certificates
extern const unsigned char comp_cert[];
extern const int comp_cert_len;
extern const unsigned char comp_key[];
extern const int comp_key_len;

void UserMain(void *pd)
{
    init();
    EnableSystemDiagnostics();

    // Install certificate BEFORE starting HTTPS
    int result = SSL_InstallCertificate(
        comp_cert, comp_cert_len,
        comp_key, comp_key_len
    );

    if (result == 0) {
        printf("[OK] Certificate installed successfully\n");
    } else {
        printf("[ERR] Certificate installation failed: %d\n", result);
    }

    StartHttps();

    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

10.15.21.2 Installing Certificate Chain

For trusted certificates, install the full chain:

```
// Combine certificate with intermediate and root CAs
extern const unsigned char comp_device_cert[];
extern const int comp_device_cert_len;
extern const unsigned char comp_chain[];
extern const int comp_chain_len;
extern const unsigned char comp_key[];
```

```

extern const int comp_key_len;

void InstallWithChain() {
    // Method 1: Combined file (preferred)
    SSL_InstallCertificate(comp_chain, comp_chain_len,
                          comp_key, comp_key_len);

    // Method 2: Separate installation
    SSL_InstallCertificate(comp_device_cert, comp_device_cert_len,
                          comp_key, comp_key_len);
    SSL_AddIntermediateCert(comp_chain, comp_chain_len);
}

```

10.15.22 Method 4: SSL Client Implementation

Connecting to secure servers with certificate validation.**

10.15.22.1 Basic SSL Client

```

#include <ssl.h>
#include <nettcp.h>

// CA certificate for validation
extern const char trustedCA[];
extern const int trustedCA_len;

void SecureClientConnection()
{
    // Initialize SSL (once at startup)
    SslInit();

    // Add trusted CA certificates
    SSL_AddCertToClientCaList(trustedCA, trustedCA_len);

    // Create TCP connection
    int fd = connect("api.example.com", 443);

    if (fd > 0) {
        // Upgrade to SSL
        WOLFSSL* ssl = SSL_connect(fd);

        if (ssl) {
            printf("[OK] Secure connection established\n");

            // Send data
            const char* request = "GET / HTTP/1.1\r\n"
                                  "Host: api.example.com\r\n"
                                  "\r\n";
            SSL_write(ssl, request, strlen(request));

            // Read response
            char buffer[1024];
            int n = SSL_read(ssl, buffer, sizeof(buffer)-1);
            if (n > 0) {
                buffer[n] = '\0';
                printf("Response: %s\n", buffer);
            }

            SSL_close(ssl);
        } else {
            printf("[ERR] SSL handshake failed\n");
        }
        close(fd);
    }
}

```

10.15.22.2 Advanced: Certificate Verification

```

#include <ssl.h>

// Custom verification callback
int VerifyCertCallback(int preverify_ok, WOLFSSL_X509_STORE_CTX* store) {
    if (!preverify_ok) {
        // Get error details
        int err = X509_STORE_CTX_get_error(store);
        int depth = X509_STORE_CTX_get_error_depth(store);

        printf("Certificate verification failed at depth %d\n", depth);
        printf("Error: %s\n", X509_verify_cert_error_string(err));

        // Decide whether to accept anyway
        // return 1; // Accept despite error (insecure!)
        return 0; // Reject connection (secure)
    }
}

```

```

    }
    return 1; // Accept
}

void SetupSecureClient() {
    SslInit();

    // Add CA certificates
    SSL_AddCertToClientCaList(trustedCA, trustedCA_len);

    // Set verification callback
    SSL_SetVerifyCallback(VerifyCertCallback);

    // Set verification mode
    SSL_SetVerifyMode(SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT);
}

```

10.15.22.3 Loading CAs from Storage

```

#include <effs_fat.h>
#include <ssl.h>

void LoadCAsFromSD() {
    SslInit();

    F_FILE* f = f_open("ca-bundle.crt", "r");
    if (f) {
        // Get file size
        f_seek(f, 0, SEEK_END);
        long size = f_tell(f);
        f_seek(f, 0, SEEK_SET);

        // Read CA bundle
        char* certs = new char[size + 1];
        f_read(certs, 1, size, f);
        certs[size] = '\0';
        f_close(f);

        // Add to trusted list
        SSL_AddCertToClientCaList(certs, size);

        delete[] certs;
        printf("[OK] Loaded CA certificates from SD\n");
    }
}

```

10.15.23 Troubleshooting

10.15.24 Common Browser Warning Messages

10.15.24.1 1. "Your Connection is Not Private"

```

[!] Your connection is not private

Attackers might be trying to steal
your information...

NET::ERR_CERT_AUTHORITY_INVALID

```

Cause:** Certificate not signed by trusted CA

Solutions:**

- [OK] Use ACME for automatic trusted certificates
- [OK] Install your CA certificate on client devices
- [OK] Purchase certificate from public CA
- [!] For testing only: Click "Advanced" -> "Proceed (unsafe)"

10.15.24.2 2. "Certificate Name Mismatch"

```

[!] Certificate doesn't match

Certificate: device.example.com
Accessing: 192.168.1.100

NET::ERR_CERT_COMMON_NAME_INVALID

```

Cause:** Certificate issued for different name/IP

Solutions:**

- [OK] Access device using correct hostname
- [OK] Add Subject Alternative Names (SANs) to certificate
- [OK] Use wildcard certificate (*.example.com)
- [OK] Generate new certificate with correct CN/SAN

Example with multiple names:

```
void CertCallback(SslCertAttributes_t *attrs) {
    attrs->commonName = "device.example.com";
    attrs->altNames = "device.example.com,"
                     "device.local,"
                     "192.168.1.100,"
                     "10.0.0.50";
}
```

10.15.24.3 3. "Certificate Expired"

```
[!] This certificate has expired

Valid Until: Jan 15, 2025
Today:      Feb 20, 2025

NET::ERR_CERT_DATE_INVALID
```

Cause:** Certificate past expiration date

Solutions:**

- [OK] Use ACME for automatic renewal
- [OK] Enable auto-regeneration for self-signed certs
- [OK] Set calendar reminders for manual renewal
- [OK] Check device clock is synchronized (NTP)

10.15.24.4 4. "Certificate Not Yet Valid"

Cause:** Device clock incorrect or certificate future-dated

Solutions:**

```
#include <ntpclient.h>

void UserMain(void *pd) {
    init();

    // Synchronize time BEFORE SSL operations
    SetNTPServer("pool.ntp.org");
    EnableNTP();

    // Wait for time sync
    while (!NTPTimeSet()) {
        OSTimeDly(TICKS_PER_SECOND);
    }

    printf("[OK] Time synchronized\n");

    StartHttps(); // Now SSL will work

    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}
```

10.15.25 SSL Handshake Failures

10.15.25.1 Debugging Connection Issues

```
#include <ssl.h>

void DebugSSLConnection(const char* host, int port) {
    printf("\n=== SSL Debug Info ===\n");
}
```

```

// 1. Test TCP connectivity
printf("Testing TCP connection to %s:%d...\n", host, port);
int fd = connect(host, port);
if (fd < 0) {
    printf("[ERR] TCP connection failed\n");
    return;
}
printf("[OK] TCP connected\n");

// 2. Enable SSL debugging
wolfSSL_Debugging_ON();

// 3. Attempt SSL handshake
printf("Attempting SSL handshake...\n");
WOLFSSL* ssl = SSL_connect(fd);

if (ssl) {
    printf("[OK] SSL handshake successful\n");

    // Get connection details
    WOLFSSL_CIPHER* cipher = SSL_get_current_cipher(ssl);
    printf("Cipher: %s\n", SSL_CIPHER_get_name(cipher));
    printf("Protocol: %s\n", SSL_get_version(ssl));

    SSL_close(ssl);
} else {
    printf("[ERR] SSL handshake failed\n");
    int err = SSL_get_error(ssl, -1);
    printf("Error code: %d\n", err);
}

close(fd);
wolfSSL_Debugging_OFF();
}

```

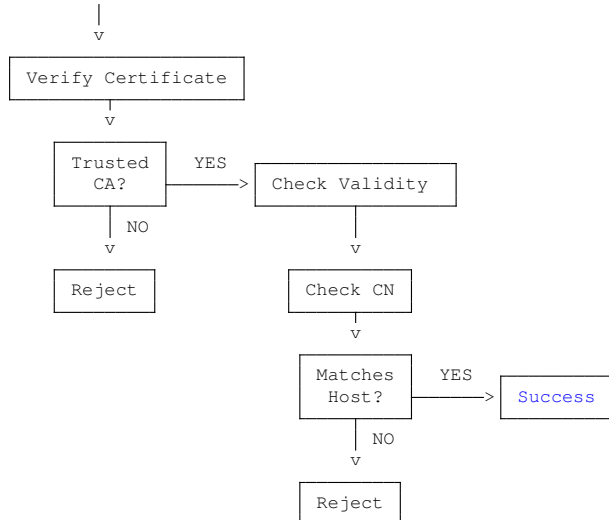
10.15.25.2 Common Error Codes

Error	Meaning	Solution
-188	ASN parse error	Certificate format invalid
-173	Bad certificate	Corrupted or wrong cert format
-155	No CA match	Add appropriate CA to trust list
-313	Cipher suite mismatch	Update wolfSSL or adjust cipher list
-101	Connection reset	Network issue or server rejected

10.15.26 Certificate Verification Issues

10.15.26.1 Verification Flowchart

SSL Handshake Started



10.15.26.2 Bypass Verification (Development Only!)

```
// [!] WARNING: This is insecure! Only for development!

void InsecureClientSetup() {
    SslInit();

    // Disable certificate verification
    SSL_SetVerifyMode(SSL_VERIFY_NONE);

    printf("[!] Certificate verification DISABLED\n");
    printf("[!] Use only for development/testing\n");
}

// For production, always verify:
void SecureClientSetup() {
    SslInit();
    SSL_AddCertToClientCaList(trustedCA, trustedCA_len);
    SSL_SetVerifyMode(SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT);
}
```

10.15.27 Memory and Resource Issues

10.15.27.1 Memory Requirements

Typical SSL Memory Usage:

Component	Size
SSL Context	~8-12 KB
Certificate Chain	2-8 KB
Per Connection	40-60 KB
Crypto Operations	10-20 KB

Minimum Recommended: 128 KB free heap Comfortable: 256 KB+ free heap

10.15.27.2 Checking Memory

```
#include <system.h>

void CheckMemoryForSSL() {
    uint32_t freeHeap = FreeRTOSMemRemaining();
    uint32_t minFree = FreeRTOSMemMinEver();

    printf("\n=== Memory Status ===\n");
    printf("Current Free: %lu KB\n", freeHeap / 1024);
    printf("Minimum Ever: %lu KB\n", minFree / 1024);

    if (freeHeap < 128 * 1024) {
        printf("[!] WARNING: Low memory for SSL\n");
        printf("    Consider: Reducing buffer sizes\n");
        printf("    Limiting connections\n");
        printf("    Using lighter ciphers\n");
    } else {
        printf("[OK] Sufficient memory for SSL\n");
    }
}
```

10.15.28 Performance Optimization

10.15.28.1 Cipher Suite Selection

```
#include <ssl.h>

// Faster but less secure (use only if needed)
const char* fastCiphers =
    "ECDHE-ECDSA-AES128-GCM-SHA256:"
    "ECDHE-RSA-AES128-GCM-SHA256";

// Balanced (recommended)
const char* balancedCiphers =
    "ECDHE-ECDSA-AES256-GCM-SHA384:"
    "ECDHE-ECDSA-AES128-GCM-SHA256:"
    "ECDHE-RSA-AES256-GCM-SHA384";

// Maximum security (slower)
const char* secureCiphers =
    "ECDHE-ECDSA-CHACHA20-POLY1305:"
```



```

    "ECDHE-ECDSA-AES256-GCM-SHA384";

void ConfigureCiphers() {
    SSL_SetCipherList(balancedCiphers);
    printf("Cipher list configured\n");
}

```

10.15.28.2 Connection Reuse

```

// Reuse SSL context for multiple connections
WOLFSSL_CTX* ctx = nullptr;

void InitSSLContext() {
    if (!ctx) {
        ctx = wolfSSL_CTX_new(wolfTLSv1_2_client_method());
        SSL_AddCertToClientCaList(trustedCA, trustedCA_len);
    }
}

WOLFSSL* CreateConnection(const char* host, int port) {
    InitSSLContext(); // Only runs once

    int fd = connect(host, port);
    if (fd < 0) return nullptr;

    WOLFSSL* ssl = wolfSSL_new(ctx);
    wolfSSL_set_fd(ssl, fd);

    if (wolfSSL_connect(ssl) != SSL_SUCCESS) {
        wolfSSL_free(ssl);
        close(fd);
        return nullptr;
    }

    return ssl;
}

```

10.15.29 Best Practices

10.15.30 Security Principles

10.15.30.1 1. Private Key Protection

Private Key Security Hierarchy
Most Secure: └ Hardware Security Module (HSM) └ Secure Enclave / TPM └ Encrypted Storage Moderate: └ Compiled into firmware └ Flash storage with permissions └ Protected file system Least Secure: └ Plain text on SD card └ Unprotected flash └ Debug accessible memory

Best Practices:**

- [YES] Never store private keys in version control
- [YES] Use different keys for dev/test/production
- [YES] Rotate keys periodically
- [YES] Monitor for unauthorized access
- [NO] Don't log private keys
- [NO] Don't transmit private keys unencrypted
- [NO] Don't reuse keys across devices (except CA)

10.15.30.2 2. Certificate Validation

```
// [X] BAD: Accepting any certificate
SSL_SetVerifyMode(SSL_VERIFY_NONE);

// [OK] GOOD: Proper validation
SSL_SetVerifyMode(SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT);
SSL_AddCertToClientCaList(trustedCA, trustedCA_len);

// [OK] BETTER: With custom verification
int VerifyCallback(int preverify, WOLFSSL_X509_STORE_CTX* store) {
    // Additional checks: expiration, revocation, etc.
    return preverify && CustomSecurityChecks(store);
}
SSL_SetVerifyCallback(VerifyCallback);
```

10.15.30.3 3. Protocol and Cipher Configuration

```
// [X] BAD: Allowing old protocols
SSL_SetMinVersion(SSL_3_0); // SSL is broken!

// [OK] GOOD: Modern protocols only
SSL_SetMinVersion(TLS_1_2);

// [OK] BETTER: Latest protocols
SSL_SetMinVersion(TLS_1_3);

// Configure strong cipher suites
const char* strongCiphers =
    "TLS_AES_256_GCM_SHA384:"           // TLS 1.3
    "TLS_CHACHA20_POLY1305_SHA256:"    // TLS 1.3
    "ECDHE-ECDSA-AES256-GCM-SHA384:"    // TLS 1.2
    "ECDHE-RSA-AES256-GCM-SHA384";      // TLS 1.2

SSL_SetCipherList(strongCiphers);
```

10.15.31 Production Checklist

10.15.31.1 Pre-Deployment

```
[ ] Security Review:
[ ] All private keys properly protected
[ ] Certificate chain complete and valid
[ ] Strong cipher suites configured
[ ] Old protocols disabled (SSL, TLS 1.0/1.1)
[ ] Certificate validation enabled for clients

[ ] Certificate Management:
[ ] Expiration dates documented
[ ] Renewal process established
[ ] Contact information current
[ ] Backup certificates available

[ ] Monitoring Setup:
[ ] Certificate expiration alerts
[ ] Connection failure logging
[ ] Performance metrics
[ ] Security event logging

[ ] Testing Complete:
[ ] Browser compatibility verified
[ ] Client application tested
[ ] Load testing performed
[ ] Failover scenarios tested

[ ] Documentation:
[ ] Certificate inventory maintained
[ ] Configuration documented
[ ] Recovery procedures written
[ ] Contact list updated
```

10.15.31.2 Post-Deployment Monitoring

```
#include <ssl.h>

// Monitor SSL health
void MonitorSSLStatus() {
    uint32_t activeConnections = SSL_GetActiveConnections();
    uint32_t failedHandshakes = SSL_GetFailedHandshakes();
    uint32_t certificateErrors = SSL_GetCertErrors();

    printf("\n=== SSL Status ===\n");
    printf("Active Connections: %lu\n", activeConnections);
}
```

```

printf("Failed Handshakes: %lu\n", failedHandshakes);
printf("Certificate Errors: %lu\n", certificateErrors);

// Check certificate expiration
uint32_t daysUntilExpiry = SSL_GetCertDaysRemaining();
if (daysUntilExpiry < 30) {
    printf("[!] Certificate expires in %lu days!\n", daysUntilExpiry);
    // Trigger renewal process
}

// Alert if too many failures
if (failedHandshakes > 100) {
    printf("[!] High handshake failure rate!\n");
    // Send alert to administrator
}
}
}

```

10.15.32 Development Workflow

10.15.32.1 Recommended Development Cycle

Development Stages
1. Development (Local Network) ↳ Self-Signed Certificates - Fast iteration - No external dependencies 2. Integration Testing ↳ Private CA Certificates - Test certificate chains - Verify validation logic 3. Staging (Pre-Production) ↳ ACME Staging Certs - Test automation - Verify renewal process 4. Production ↳ ACME Production or Public CA - Trusted by all clients - Automatic renewal

10.15.32.2 Configuration Management

```

// Use preprocessor for environment-specific config

#ifdef DEVELOPMENT
    // Development: Self-signed, relaxed validation
    #define SSL_MODE          SSL_SELF_SIGNED
    #define VERIFY_MODE       SSL_VERIFY_NONE
    #define DEBUG_SSL        1

#elif defined(STAGING)
    // Staging: Test ACME, strict validation
    #define SSL_MODE          SSL_ACME_STAGING
    #define VERIFY_MODE       SSL_VERIFY_PEER
    #define DEBUG_SSL        1

#elif defined(PRODUCTION)
    // Production: Full security, monitoring
    #define SSL_MODE          SSL_ACME_PRODUCTION
    #define VERIFY_MODE       SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT
    #define DEBUG_SSL        0
    #define ENABLE_SSL_MONITORING 1

#endif

void ConfigureSSL() {
    #if SSL_MODE == SSL_SELF_SIGNED
        StartHttps(); // Auto-generated cert

    #elif SSL_MODE == SSL_ACME_STAGING
        AcmeConfig config;
        config.acmeServer = ACME_LETSENCRYPT_STAGING;
        StartAcmeHttps(config);

    #elif SSL_MODE == SSL_ACME_PRODUCTION
        AcmeConfig config;
        config.acmeServer = ACME_LETSENCRYPT_PRODUCTION;

```

```

        StartAcmeHttps(config);
    #endif

    SSL_SetVerifyMode(VERIFY_MODE);

    #if DEBUG_SSL
        wolfSSL_Debugging_ON();
    #endif
}

```

10.15.33 Common Pitfalls to Avoid

10.15.33.1 1. Time Synchronization

```

// [X] BAD: Starting SSL before time sync
void UserMain(void *pd) {
    init();
    StartHttps(); // Certificates may appear expired!
    EnableNTP(); // Too late!
}

// [OK] GOOD: Ensure time is synced first
void UserMain(void *pd) {
    init();

    EnableNTP();
    SetNTPServer("pool.ntp.org");

    // Wait for time sync
    uint32_t timeout = 0;
    while (!NTPTimeSet() && timeout++ < 30) {
        OSTimeDly(TICKS_PER_SECOND);
    }

    if (NTPTimeSet()) {
        printf("[OK] Time synchronized\n");
        StartHttps(); // Now safe
    } else {
        printf("[!] Time sync failed, SSL may have issues\n");
    }
}

```

10.15.33.2 2. Certificate Chain Issues

```

// [X] BAD: Installing only device certificate
SSL_InstallCertificate(device_cert, device_cert_len,
                      device_key, device_key_len);
// Clients may not trust the certificate!

// [OK] GOOD: Installing complete chain
// Combine: device cert + intermediate CA + root CA
SSL_InstallCertificate(full_chain, full_chain_len,
                      device_key, device_key_len);

```

10.15.33.3 3. Resource Leaks

```

// [X] BAD: Not cleaning up
void MakeRequest() {
    WOLFSSL* ssl = SSL_connect(fd);
    SSL_write(ssl, data, len);
    // Forgot SSL_close() and close(fd)!
}

// [OK] GOOD: Proper cleanup
void MakeRequest() {
    int fd = connect(host, port);
    if (fd < 0) return;

    WOLFSSL* ssl = SSL_connect(fd);
    if (ssl) {
        SSL_write(ssl, data, len);
        SSL_close(ssl); // Free SSL resources
    }
    close(fd); // Close socket
}

```

10.15.33.4 4. Hardcoded Credentials

```

// [X] VERY BAD: Hardcoded in source
#define CA_KEY "-----BEGIN PRIVATE KEY-----\nMIIE..."

```

```
#define DEVICE_PASSWORD "admin123"

// [OK] BETTER: Compiled externally
extern const char comp_key[];
extern const int comp_key_len;

// [OK] BEST: Retrieved from secure storage
void LoadCredentials() {
    // Read from encrypted flash, secure element, etc.
    ReadSecureKey(keyBuffer, sizeof(keyBuffer));
}
```

10.15.34 Additional Resources

10.15.35 NetBurner Examples

Located in your NNDK installation:

```
C:\nburn\examples\SSL\
+-- acmeservlet\          <- ACME automation
+-- SslOnboardCertGeneration\ <- Auto-generated certs
|   +-- Simple\
|   +-- Advanced\
|   +-- CompiledCa\
+-- SslClientVerifyPeerBasic\ <- Client validation
+-- SslClientVerifyPeerEfs\   <- Loading CAs from storage
+-- CreateCerts\             <- Certificate generation scripts
    +-- ECDSA\
    +-- RSA\
```

10.15.36 Documentation and References

10.15.36.1 Official Documentation

- **wolfSSL Manual:** <https://www.wolfssl.com/documentation/manuals/wolfssl/>
- **NetBurner SSL API:** See `nburn\include\ssl.h`
- **ACME Protocol:** RFC 8555
- **TLS Protocol:** RFC 8446 (TLS 1.3)

10.15.36.2 Educational Resources

Books:**

- *Bulletproof SSL and TLS* by Ivan Ristic
 - *Applied Cryptography* by Bruce Schneier
 - *SSL and TLS: Designing and Building Secure Systems* by Eric Rescorla
- Online:**

- [SSL Labs](#) - SSL testing and best practices
- [Let's Encrypt Documentation](#)
- [Mozilla SSL Configuration Generator](#)

10.15.36.3 Wikipedia Articles

- [Public Key Infrastructure \(PKI\)](#)
- [Transport Layer Security \(TLS\)](#)
- [Certificate Authority](#)
- [X.509 Certificates](#)

10.15.37 Tools and Utilities

10.15.37.1 Certificate Management

- **OpenSSL:** Included in NNDK at \Nburn\gcc\msys\1.0\bin\openssl.exe
- **NetBurner CreateCerts:** Scripts in \nburn\CreateCerts\
- **compfile:** NetBurner utility to convert files to C++ arrays

10.15.37.2 Testing and Debugging

```
# Test SSL/TLS connection
openssl s_client -connect device.example.com:443 -showcerts

# Check certificate details
openssl x509 -in device.crt -text -noout

# Verify private key
openssl rsa -in device.key -check

# Test certificate chain
openssl verify -CAfile ca.crt device.crt
```

10.15.37.3 Online Testing

- **SSL Labs Server Test:** <https://www.ssllabs.com/ssltest/>
- **Let's Debug:** <https://letsdebug.net/> (for ACME troubleshooting)
- **Certificate Decoder:** <https://www.sslshopper.com/certificate-decoder.html>

10.15.38 Quick Reference

10.15.38.1 Essential SSL Functions

```
// Server Setup
StartHttps(); // Auto-generated cert
StartAcmeHttps(config); // ACME automation
SSL_InstallCertificate(cert, key); // Manual install

// Client Setup
SslInit(); // Initialize SSL
SSL_AddCertToClientCaList(ca); // Add trusted CA
SSL_connect(fd); // Secure connection
SSL_SetVerifyMode(mode); // Verification level

// Certificate Management
SSL_CreateNewSelfSignedCert(); // Generate cert
SSL_GetCertDaysRemaining(); // Check expiration
SSL_SetCertificateCallback(fn); // Custom attributes

// Data Transfer
SSL_write(ssl, data, len); // Send data
SSL_read(ssl, buffer, size); // Receive data
SSL_close(ssl); // Close connection

// Debugging
wolfSSL_Debugging_ON(); // Enable debug logs
SSL_get_error(ssl, ret); // Get error code
```

10.15.38.2 Configuration Constants

```
// Verification Modes
SSL_VERIFY_NONE // No validation (insecure!)
SSL_VERIFY_PEER // Validate peer
SSL_VERIFY_FAIL_IF_NO_PEER_CERT // Require peer cert

// Protocol Versions
SSL_3_0 // Deprecated
TLS_1_0 // Deprecated
TLS_1_1 // Deprecated
TLS_1_2 // Minimum recommended
TLS_1_3 // Latest

// ACME Servers
ACME_LETSENCRYPT_STAGING // Testing
ACME_LETSENCRYPT_PRODUCTION // Production
```

```
ACME_BUYPASS_PRODUCTION           // Alternative CA
// Certificate Types
SSL_FILETYPE_PEM                   // Text format
SSL_FILETYPE_DER                   // Binary format
```

Recommended Approach for Most Users:**

1. **Development:** Use auto-generated self-signed certificates
2. **Testing:** Set up private CA for realistic testing
3. **Production:** Deploy ACME if possible, otherwise use public CA

Key Takeaways:**

- [LOCK] Private keys are your most sensitive security asset
- [SYNC] ACME automation eliminates manual certificate management
- [TIME] Time synchronization is critical for SSL/TLS
- [LINK] Complete certificate chains prevent trust issues
- [STAT] Monitor certificate expiration and connection health

10.16 SSH Programming Guide

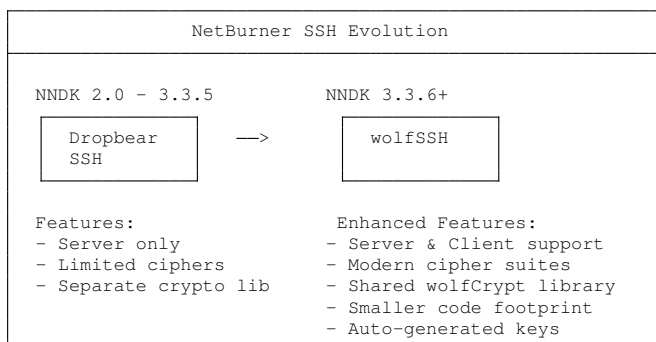
10.16.1 SSH Programming Guide for NetBurner Devices

10.16.2 Overview

The Secure Shell Protocol (SSH) is a cryptographic network protocol for secure remote access and command execution over unsecured networks. First introduced in 1995, SSH remains the industry standard for secure remote administration.

10.16.2.1 NetBurner SSH Implementation

NetBurner has supported SSH since NNDK 2.0. As of **NNDK 3.3.6**, we've migrated from Dropbear to **wolfSSH**, providing significant improvements:



10.16.2.2 Key Benefits of wolfSSH

Enhanced Security**

- Updated cipher suites and algorithms
- Support for modern cryptographic standards
- Regular security updates

Client Support**

- SSH client functionality now available

- Connect to remote SSH servers from NetBurner devices
- Automated remote management capabilities
Code Efficiency**
- Shared wolfCrypt library with SSL/TLS
- Reduced memory footprint when using both protocols
- Optimized for embedded systems
Simplified Key Management**
- Automatic key generation on device
- Multiple key format support
- Flexible key installation methods

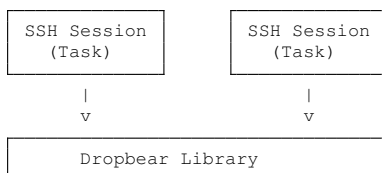
10.16.3 What's New in NNDK 3.3.6+

10.16.3.1 Major Changes Summary

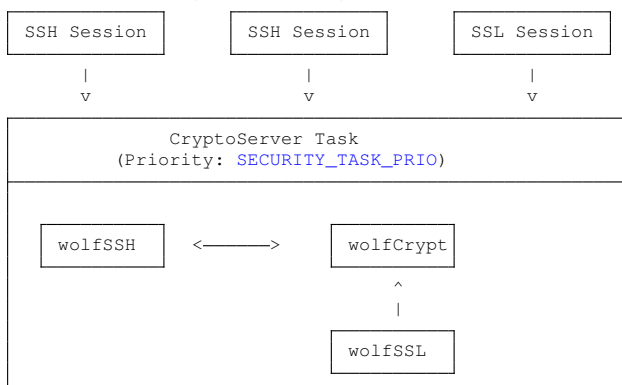
wolfSSH Migration	
[ADDED]	SSH Client Support
[ADDED]	Onboard Key Generation
[ADDED]	UserAuthManager Class
[ADDED]	Enhanced Authentication (keys + passwords)
[IMPROVED]	Modern Cipher Suites
[IMPROVED]	Shared Crypto Library (wolfCrypt)
[CHANGED]	Task Management (CryptoServer)
[REMOVED]	Dropbear-specific Functions

10.16.3.2 Architecture Change

Previous Architecture (NNDK 3.3.5 and earlier):**



New Architecture (NNDK 3.3.6+):**



Key Architectural Improvements:**

- Unified task management through CryptoServer
- Shared cryptographic primitives (wolfCrypt)
- Reduced resource overhead
- Consistent security across protocols

10.16.4 Key Types and Formats

10.16.4.1 Supported Key Types

The SSH library supports both **ECC** (Elliptic Curve Cryptography) and **RSA** keys.

Supported SSH Key Types	
ECC (Elliptic Curve) - RECOMMENDED	
Key Sizes:	160 to 512 bits
Advantages:	- Smaller key size - Faster operations - Lower memory usage
Use Case:	Embedded systems, IoT devices
RSA (Rivest-Shamir-Adleman)	
Default Size:	2048 bits
Optional:	4096 bits (define ENABLE_RSA_4K)
Advantages:	- Widely compatible - Well-established
Use Case:	Legacy system compatibility

10.16.4.1.1 Enabling RSA 4096-bit Keys

To enable 4096-bit RSA key support:

1. Open `<nndk_install>\nbrtos\include\predef.h`
2. Add the following definition:

```
#define ENABLE_RSA_4K
```

1. Rebuild your application

10.16.4.2 Supported Key Formats

Both key types support two formats:

SSH Key Format Support	
PEM Format (Privacy-Enhanced Mail)	
<pre>-----BEGIN RSA PRIVATE KEY----- Base64 encoded key data... -----END RSA PRIVATE KEY-----</pre> - Human readable (text) - Base64 encoding - Easier to copy/paste - Larger file size	
ASN.1/DER Format (Distinguished Encoding Rules)	
<pre>Binary encoded key data (not human readable)</pre> - Binary format - Smaller file size - Faster parsing - More efficient for embedded systems	

10.16.5 Migration Guide

10.16.5.1 Project Configuration

Step 1: Add Required Include Paths**

When building SSH applications in NBEclipse, add the following include paths:

```

${NNDK_ROOT}/libraries/include/ssh
${NNDK_ROOT}/libraries/include/crypto

```

Step 2: Configure in NBEclipse**

Project Configuration Flow:

1. Right-click project
|
v
2. Select "Properties"
|
v
3. Navigate to "C/C++ Build" > "Settings"
|
v
4. Select "Includes" under:
- GNU C Compiler
- GNU C++ Compiler
|
v
5. Click green "+" icon
|
v
6. Add paths:
\${NNDK_ROOT}/libraries/include/ssh
\${NNDK_ROOT}/libraries/include/crypto
|
v
7. Click "Apply"
|
v
8. Clean and rebuild project

10.16.5.2 Removed Functions

The following Dropbear-specific functions have been removed:

Removed Functions
<pre> SshConvertDecodedOpenSSLKey() ├ Purpose: Converted PEM to Dropbear format ├ Reason: No longer needed - native PEM/ASN.1 support └ Action: Remove from code SshSetchansessionrequest() ├ Purpose: Set callback for channel requests ├ Reason: Handled internally by wolfSSH └ Action: Remove callback - automatic handling SshSetTaskPriority() ├ Purpose: Set SSH task priority ├ Reason: Managed by CryptoServer task └ Action: Remove - uses SECURITY_TASK_PRIO </pre>

10.16.5.3 Updated Functions

Replace deprecated functions with their enhanced versions:

10.16.5.3.1 Authentication Function Pointer

Old (Deprecated):**

```

// Old function pointer - password only
typedef int (*sshUserAuthenticateFn)(const char* username, const char* password);

// Old setter
void SshSetUserAuthenticate(sshUserAuthenticateFn fn);

// Old getter
sshUserAuthenticateFn SshGetUserAuthenticate(void);

```

New (Recommended):**

```

// New function pointer - supports password AND key authentication
typedef int (*sshUserAuthenticateWithTypeFn)(const char* username,
                                             const char* authValue,
                                             AuthType authType);

// New setter
void SshSetUserAuthenticateWithType(sshUserAuthenticateWithTypeFn fn);

// New getter

```

```
sshUserAuthenticateWithTypeFn SshGetUserAuthenticateWithType(void);

Migration Example:**
// OLD CODE (NNDK 3.3.5 and earlier)
int MyAuthCallback(const char* username, const char* password) {
    // Only handled password authentication
    if (strcmp(username, "admin") == 0 && strcmp(password, "secret") == 0) {
        return 1; // Success
    }
    return 0; // Failure
}

void setup() {
    SshSetUserAuthenticate(MyAuthCallback);
}

// NEW CODE (NNDK 3.3.6+)
int MyAuthCallbackWithType(const char* username,
                           const char* authValue,
                           AuthType authType) {
    // Handle both password and key authentication
    if (authType == AUTH_TYPE_PASSWORD) {
        if (strcmp(username, "admin") == 0 &&
            strcmp(authValue, "secret") == 0) {
            return 1; // Success
        }
    }
    else if (authType == AUTH_TYPE_KEY) {
        // Validate public key
        if (ValidatePublicKey(username, authValue)) {
            return 1; // Success
        }
    }
    return 0; // Failure
}

void setup() {
    SshSetUserAuthenticateWithType(MyAuthCallbackWithType);
}

Authentication Type Enum:**
typedef enum {
    AUTH_TYPE_PASSWORD, // Password-based authentication
    AUTH_TYPE_KEY,      // Public key authentication
    AUTH_TYPE_UNKNOWN   // Unknown/unsupported type
} AuthType;
```

10.16.6 Key Management

10.16.6.1 Key Installation Methods

SSH keys can be installed using three methods, checked in the following priority order:

Key Selection Priority (Checked in Order):

1. Runtime Callback

`SshSetUserGetKey()` callback
Returns key dynamically

|
| Not set?
v

2. Compiled-In Key

Override `GetPrivateKeyPEM()`
Weak function override

|
| Not overridden?
v

3. Auto-Generated Key

Onboard generated key
Same as SSL/TLS key

10.16.6.2 Method 1: Runtime Callback (Highest Priority)

Use this method for dynamic key selection or key rotation.

```
#include <ssh.h>

// Callback signature
typedef int (*SshUserGetKeyFn)(const char** keyData,
                               int* keyLen,
```

```

        int* keyFormat);

// Example implementation
int GetSshKey(const char** keyData, int* keyLen, int* keyFormat) {
    // Load key from secure storage, file system, etc.
    static const char myKey[] =
        "-----BEGIN EC PRIVATE KEY-----\n"
        "MHcCAQEEIIGN...key data...zqk\n"
        "-----END EC PRIVATE KEY-----\n";

    *keyData = myKey;
    *keyLen = sizeof(myKey) - 1; // Exclude null terminator
    *keyFormat = SSH_KEY_FORMAT_PEM;

    return 1; // Success
}

void UserMain(void* pd) {
    init();

    // Set the callback
    SshSetUserGetKey(GetSshKey);

    // Start SSH server
    StartSsh();

    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}

Key Format Constants:**
#define SSH_KEY_FORMAT_PEM    0 // PEM format
#define SSH_KEY_FORMAT_DER    1 // DER/ASN.1 format

```

10.16.6.3 Method 2: Compiled-In Key (Medium Priority)

Override the weak function to provide a compiled-in key.

```

#include <ssh.h>

// Weak function (defined in SSH library)
// Override this in your application
extern const unsigned char myPrivateKey[];
extern const int myPrivateKeyLen;

// Override the weak function
const char* GetPrivateKeyPEM(void) {
    return (const char*)myPrivateKey;
}

// In a separate file (e.g., ssh_key.cpp)
// Generated using compfile utility
extern const unsigned char myPrivateKey[] = {
    0x2d, 0x2d, 0x2d, 0x2d, 0x2d, // PEM header
    // ... key data ...
};
extern const int myPrivateKeyLen = sizeof(myPrivateKey);

Generating Compiled Key File:**
# Convert PEM key to C array using compfile
compfile ssh_key.pem myPrivateKey myPrivateKeyLen ssh_key.cpp

# Add ssh_key.cpp to your project's makefile
CPP_SRC += src/ssh_key.cpp

```

10.16.6.4 Method 3: Auto-Generated Key (Lowest Priority)

Enable automatic key generation on the device.

```

// The device will automatically generate a key if:
// 1. No callback is set (Method 1)
// 2. GetPrivateKeyPEM() is not overridden (Method 2)
// 3. No existing key is found in storage

// The auto-generated key is the same key used for SSL/TLS
// certificates, reducing storage and memory requirements

void UserMain(void* pd) {
    init();

    // Start SSH - will auto-generate key if needed
    StartSsh();

    printf("SSH server started with auto-generated key\n");

    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}

```

Enable Auto-Regeneration:**

To automatically regenerate expired keys, add to `predef.h`:

```
// Enable automatic key regeneration
#define ENABLE_AUTOCERT_REGEN

// Optional: Set check interval (default: 1 minute)
#define AUTO_CERT_GEN_CHECK (60 * 20) // Check every 20 minutes
```

Key Generation Details:**

- Generated key is stored in non-volatile memory
- Persists across reboots
- Shared with SSL/TLS (same key for both protocols)
- Automatically regenerated when expired (if enabled)
- ECC SECP384R1 by default (configurable)

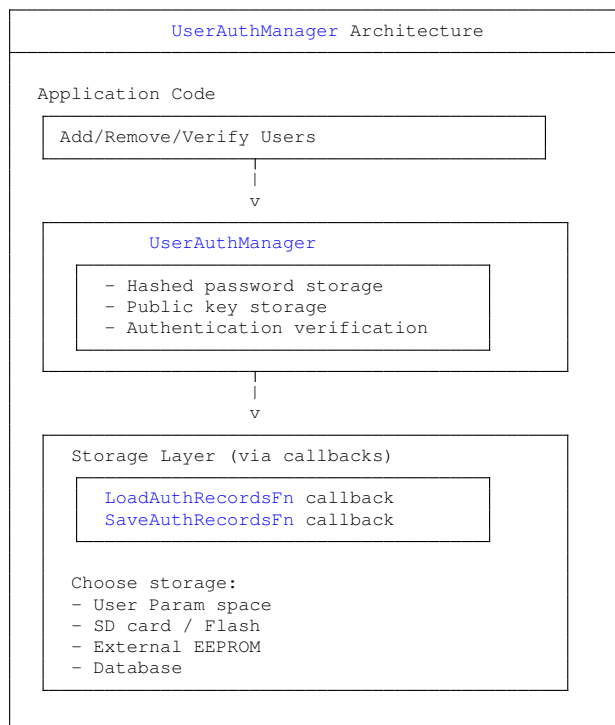
Related Examples:** See `<nndk_install>/examples/SSL/SslOnboardCertGeneration/` for:

- Simple: Basic auto-generation
- Advanced: Custom key attributes
- CompiledCa: Using a compiled CA

10.16.7 User Authentication

10.16.7.1 UserAuthManager Class

NetBurner provides the `UserAuthManager` class to simplify user credential management.



10.16.7.2 Key Features

Security Features:**

- Passwords are hashed before storage
- Keys are stored securely

- No plain-text credential leakage
- Secure comparison functions
- Flexibility:**
- Custom storage backends via callbacks
- Support for password and key authentication
- User-defined storage format (JSON, binary, etc.)
- In-memory or persistent storage

10.16.7.3 Basic Usage

```
#include <UserAuthManager.h>

// Global instance
UserAuthManager authManager;

// Initialize with storage callbacks
void InitializeAuth() {
    // Set load/save callbacks (see example below)
    authManager.SetLoadCallback(LoadAuthRecords);
    authManager.SetSaveCallback(SaveAuthRecords);

    // Load existing users from storage
    authManager.Load();
}

// Add a user with password
void AddPasswordUser(const char* username, const char* password) {
    if (authManager.AddUser(username, password, AUTH_TYPE_PASSWORD)) {
        printf("User %s added successfully\n", username);
        authManager.Save(); // Persist to storage
    } else {
        printf("Failed to add user %s\n", username);
    }
}

// Add a user with public key
void AddKeyUser(const char* username, const char* publicKey) {
    if (authManager.AddUser(username, publicKey, AUTH_TYPE_KEY)) {
        printf("User %s added with key authentication\n", username);
        authManager.Save();
    } else {
        printf("Failed to add user %s\n", username);
    }
}

// Verify user credentials
int AuthenticateUser(const char* username,
                    const char* authValue,
                    AuthType authType) {
    return authManager.Authenticate(username, authValue, authType);
}

// Remove a user
void RemoveUser(const char* username) {
    if (authManager.RemoveUser(username)) {
        printf("User %s removed\n", username);
        authManager.Save();
    }
}
```

10.16.7.4 Storage Callbacks

Implement custom storage using callbacks:

```
#include <UserAuthManager.h>
#include <njson.h>
#include <UserParams.h>

// Callback type definitions
typedef int (*LoadAuthRecordsFn)(UserAuthManager& manager);
typedef int (*SaveAuthRecordsFn)(UserAuthManager& manager);

// Load users from User Param space (example)
int LoadAuthRecords(UserAuthManager& manager) {
    // Read JSON from User Param space
    const char* jsonData = GetUserParam("SSH_USERS");
    if (!jsonData) return 0;
}
```

```

// Parse JSON and populate manager
ParsedJsonDataSet jsonSet;
if (ParseJsonData(jsonData, jsonSet)) {
    // Extract users from JSON and add to manager
    for (int i = 0; i < jsonSet.m_numDataElements; i++) {
        const char* username = jsonSet.FindName("username", i);
        const char* authData = jsonSet.FindName("auth", i);
        const char* typeStr = jsonSet.FindName("type", i);

        AuthType type = (strcmp(typeStr, "key") == 0) ?
            AUTH_TYPE_KEY : AUTH_TYPE_PASSWORD;

        manager.AddUserHashed(username, authData, type);
    }
    return 1;
}
return 0;
}

// Save users to User Param space (example)
int SaveAuthRecords(UserAuthManager& manager) {
    // Build JSON from manager data
    char jsonBuffer[4096];
    int offset = 0;

    offset += sprintf(jsonBuffer + offset, "{\\"users\\":[");

    // Iterate through users and build JSON
    // (Implementation depends on UserAuthManager API)

    offset += sprintf(jsonBuffer + offset, "]}");

    // Save to User Param space
    return SetUserParam("SSH_USERS", jsonBuffer);
}

// Register callbacks
void InitializeAuthStorage() {
    authManager.SetLoadCallback(LoadAuthRecords);
    authManager.SetSaveCallback(SaveAuthRecords);
    authManager.Load();
}

```

10.16.7.5 Integration with SSH Authentication

```

#include <ssh.h>
#include <UserAuthManager.h>

// Global manager
UserAuthManager g_authManager;

// Authentication callback for SSH
int SshAuthCallback(const char* username,
                   const char* authValue,
                   AuthType authType) {
    // Use UserAuthManager to verify credentials
    int result = g_authManager.Authenticate(username, authValue, authType);

    if (result) {
        printf("SSH: User '%s' authenticated successfully\n", username);
    } else {
        printf("SSH: Authentication failed for user '%s'\n", username);
    }

    return result;
}

void UserMain(void* pd) {
    init();

    // Initialize authentication manager
    g_authManager.SetLoadCallback(LoadAuthRecords);
    g_authManager.SetSaveCallback(SaveAuthRecords);
    g_authManager.Load();

    // Register SSH authentication callback
    SshSetUserAuthenticateWithType(SshAuthCallback);

    // Start SSH server
    StartSsh();

    printf("SSH server running with UserAuthManager\n");

    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}

```

10.16.7.6 Security Best Practices

UserAuthManager Security Features	
[HASH]	Passwords hashed before storage - Never store plain-text passwords - Secure hash algorithms used
[SAFE]	Constant-time comparison - Prevents timing attacks - Secure credential verification
[FLEX]	Pluggable storage - Choose your storage backend - Encrypt at rest if needed - Control access permissions
[KEYS]	Public key support - Store SSH public keys - Key-based authentication - More secure than passwords

Recommendations:**

1. Always hash passwords before storage
2. Use secure storage (encrypted if possible)
3. Implement proper access controls
4. Prefer key-based authentication
5. Regularly audit user accounts
6. Implement account lockout policies
7. Log authentication attempts

10.16.8 Configuration

10.16.8.1 SSH Server Configuration

Basic SSH server setup:

```
#include <init.h>
#include <ssh.h>

const char* AppName = "SSH Server Example";

void UserMain(void* pd) {
    init();

    // Wait for network
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Configure SSH (optional - uses defaults if not called)
    // SshSetPort(22); // Default SSH port

    // Start SSH server
    if (StartSsh() == 0) {
        printf("SSH server started successfully\n");
    } else {
        printf("Failed to start SSH server\n");
    }

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

10.16.8.2 SSH Client Configuration

Connect to remote SSH servers:

```
#include <ssh.h>

void ConnectToRemoteServer() {
    SshClientConfig config;
```



```

// Configure connection
config.hostname = "remote.server.com";
config.port = 22;
config.username = "admin";
config.password = "secret"; // Or use key authentication

// Connect
SSH_CLIENT* client = SshClientConnect(&config);

if (client) {
    printf("Connected to remote SSH server\n");

    // Execute command
    char response[1024];
    int result = SshClientExecute(client, "ls -la",
                                response, sizeof(response));

    if (result > 0) {
        printf("Command output:\n%s\n", response);
    }

    // Disconnect
    SshClientDisconnect(client);
} else {
    printf("Failed to connect to SSH server\n");
}
}

```

10.16.8.3 Advanced Configuration

```

#include <ssh.h>

void ConfigureSshAdvanced() {
    // Set custom port
    SshSetPort(2222);

    // Set connection timeout
    SshSetTimeout(30); // 30 seconds

    // Set maximum connections
    SshSetMaxConnections(5);

    // Enable/disable features
    SshEnablePasswordAuth(true);
    SshEnableKeyAuth(true);

    // Set custom banner
    SshSetBanner("Welcome to NetBurner SSH Server\n");

    // Configure cipher suites (if needed)
    // SshSetCipherList("aes128-ctr,aes256-ctr");

    // Start server with custom configuration
    StartSsh();
}

```

10.16.8.4 Configuration Constants

Common SSH configuration constants:

```

// Default values
#define SSH_DEFAULT_PORT 22
#define SSH_DEFAULT_TIMEOUT 60 // seconds
#define SSH_DEFAULT_MAX_CONNECTIONS 10
#define SSH_DEFAULT_KEY_SIZE 2048 // RSA

// Task priority (defined in constants.h)
#define SECURITY_TASK_PRIO 45 // CryptoServer priority

// Buffer sizes
#define SSH_MAX_USERNAME_LEN 32
#define SSH_MAX_PASSWORD_LEN 64
#define SSH_MAX_COMMAND_LEN 1024
#define SSH_MAX_RESPONSE_LEN 4096

```

10.16.9 Examples and Best Practices

10.16.9.1 Example 1: Basic SSH Server

```

#include <init.h>
#include <ssh.h>

```

```

#include <UserAuthManager.h>

const char* AppName = "Basic SSH Server";
UserAuthManager authManager;

// Simple authentication callback
int AuthUser(const char* username, const char* authValue, AuthType type) {
    return authManager.Authenticate(username, authValue, type);
}

// Initialize with default admin user
void InitDefaultUsers() {
    authManager.AddUser("admin", "admin123", AUTH_TYPE_PASSWORD);
    printf("Default user 'admin' created\n");
}

void UserMain(void* pd) {
    init();

    printf("Starting %s\n", AppName);

    // Wait for network
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Setup authentication
    InitDefaultUsers();
    SshSetUserAuthenticateWithType(AuthUser);

    // Start SSH server (auto-generated keys)
    StartSsh();

    printf("SSH server running on port 22\n");
    printf("Connect with: ssh admin@%s\n", GetIP().c_str());

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

10.16.9.2 Example 2: SSH Server with Persistent Users

```

#include <init.h>
#include <ssh.h>
#include <UserAuthManager.h>
#include <UserParams.h>
#include <nbjson.h>

UserAuthManager authManager;

// Load users from User Param space
int LoadUsers(UserAuthManager& mgr) {
    const char* jsonData = GetUserParam("SSH_USERS");
    if (!jsonData) return 0;

    ParsedJsonDataSet jsonSet;
    if (ParseJsonData(jsonData, jsonSet)) {
        for (int i = 0; i < jsonSet.m_numDataElements; i++) {
            const char* user = jsonSet.FindName("user", i);
            const char* hash = jsonSet.FindName("hash", i);
            const char* type = jsonSet.FindName("type", i);

            AuthType authType = (strcmp(type, "key") == 0) ?
                AUTH_TYPE_KEY : AUTH_TYPE_PASSWORD;
            mgr.AddUserHashed(user, hash, authType);
        }
        return 1;
    }
    return 0;
}

// Save users to User Param space
int SaveUsers(UserAuthManager& mgr) {
    char json[2048];
    // Build JSON from manager
    // ... implementation ...
    return SetUserParam("SSH_USERS", json);
}

int AuthUser(const char* username, const char* authValue, AuthType type) {
    int result = authManager.Authenticate(username, authValue, type);

    // Log authentication attempts
    if (result) {
        printf("[SSH] Login success: %s\n", username);
    } else {
        printf("[SSH] Login failed: %s\n", username);
    }
}

```

```

    }

    return result;
}

void UserMain(void* pd) {
    init();

    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Setup persistent user storage
    authManager.SetLoadCallback(LoadUsers);
    authManager.SetSaveCallback(SaveUsers);
    authManager.Load();

    // Create default user if none exist
    if (authManager.GetUserCount() == 0) {
        authManager.AddUser("admin", "netburner", AUTH_TYPE_PASSWORD);
        authManager.Save();
        printf("Created default admin user\n");
    }

    // Configure and start SSH
    SshSetUserAuthenticateWithType(AuthUser);
    StartSsh();

    printf("SSH server ready - %d users loaded\n",
        authManager.GetUserCount());

    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}

```

10.16.9.3 Example 3: SSH Client

```

#include <init.h>
#include <ssh.h>

void ExecuteRemoteCommand(const char* host, const char* command) {
    SshClientConfig config;
    config.hostname = host;
    config.port = 22;
    config.username = "admin";
    config.password = "password";

    SSH_CLIENT* client = SshClientConnect(&config);

    if (client) {
        char response[4096];
        int len = SshClientExecute(client, command,
            response, sizeof(response));

        if (len > 0) {
            printf("Response from %s:\n%s\n", host, response);
        } else {
            printf("Command execution failed\n");
        }

        SshClientDisconnect(client);
    }
}

void UserMain(void* pd) {
    init();

    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Execute commands on remote servers
    ExecuteRemoteCommand("192.168.1.100", "uptime");
    ExecuteRemoteCommand("192.168.1.101", "df -h");

    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}

```

10.16.9.4 Example 4: Custom Key Management

```

#include <init.h>
#include <ssh.h>

// Store multiple keys for different purposes
const char* productionKey =
    "-----BEGIN EC PRIVATE KEY-----\n"
    "MHcCAQEE...production key...\n"
    "-----END EC PRIVATE KEY-----\n";

const char* developmentKey =

```

```

"-----BEGIN EC PRIVATE KEY-----\n"
"MHcCAQEE...development key...\n"
"-----END EC PRIVATE KEY-----\n";

// Dynamic key selection callback
int GetSshKey(const char** keyData, int* keyLen, int* keyFormat) {
    // Select key based on environment
    #ifdef PRODUCTION
        *keyData = productionKey;
        *keyLen = strlen(productionKey);
    #else
        *keyData = developmentKey;
        *keyLen = strlen(developmentKey);
    #endif

    *keyFormat = SSH_KEY_FORMAT_PEM;
    return 1;
}

void UserMain(void* pd) {
    init();

    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Set custom key callback
    SshSetUserGetKey(GetSshKey);

    // Start SSH server with selected key
    StartSsh();

    #ifdef PRODUCTION
        printf("SSH: Using production key\n");
    #else
        printf("SSH: Using development key\n");
    #endif

    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}

```

10.16.9.5 Best Practices

SSH Security Best Practices	
[1] Key Management	- Use ECC keys when possible (smaller, faster) - Rotate keys regularly - Never hardcode keys in source code - Store keys securely
[2] Authentication	- Prefer key-based over password authentication - Use strong passwords (12+ characters) - Implement account lockout after failed attempts - Use <code>UserAuthManager</code> for credential storage
[3] Network Security	- Use non-standard ports when possible - Implement IP whitelisting/blacklisting - Set appropriate connection timeouts - Limit maximum concurrent connections
[4] Monitoring	- Log all authentication attempts - Monitor <code>for</code> suspicious activity - Track active sessions - Alert on anomalies
[5] Updates	- Keep NNDK updated <code>for</code> security patches - Review wolfSSH release notes - Test updates in development first

10.16.10 Quick Reference

10.16.10.1 Essential Functions

10.16.10.1.1 Server Functions

```
// Start/Stop SSH server
```

```

int StartSsh(void);
int StopSsh(void);

// Configuration
void SshSetPort(uint16_t port);
void SshSetTimeout(uint32_t seconds);
void SshSetMaxConnections(int max);

// Authentication
void SshSetUserAuthenticateWithType(sshUserAuthenticateWithTypeFn fn);
sshUserAuthenticateWithTypeFn SshGetUserAuthenticateWithType(void);

// Key management
void SshSetUserGetKey(SshUserGetKeyFn fn);

// Legacy (deprecated but still available)
void SshSetUserAuthenticate(sshUserAuthenticateFn fn);
sshUserAuthenticateFn SshGetUserAuthenticate(void);

```

10.16.10.1.2 Client Functions

```

// Connection
SSH_CLIENT* SshClientConnect(SshClientConfig* config);
void SshClientDisconnect(SSH_CLIENT* client);

// Command execution
int SshClientExecute(SSH_CLIENT* client, const char* command,
                    char* response, int maxlen);

// File transfer
int SshClientUpload(SSH_CLIENT* client, const char* localPath,
                   const char* remotePath);
int SshClientDownload(SSH_CLIENT* client, const char* remotePath,
                     const char* localPath);

```

10.16.10.1.3 UserAuthManager Functions

```

// User management
bool AddUser(const char* username, const char* authValue, AuthType type);
bool RemoveUser(const char* username);
bool UserExists(const char* username);
int GetUserCount(void);

// Authentication
int Authenticate(const char* username, const char* authValue,
                AuthType type);

// Storage
void SetLoadCallback(LoadAuthRecordsFn fn);
void SetSaveCallback(SaveAuthRecordsFn fn);
bool Load(void);
bool Save(void);

```

10.16.10.2 Common Patterns

10.16.10.2.1 Pattern 1: Simple Server

```

void UserMain(void* pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);
    StartSsh(); // Auto-generated key, default settings
    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}

```

10.16.10.2.2 Pattern 2: Server with Authentication

```

int AuthCallback(const char* user, const char* auth, AuthType type) {
    return authManager.Authenticate(user, auth, type);
}

void UserMain(void* pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);
    authManager.Load();
    SshSetUserAuthenticateWithType(AuthCallback);
    StartSsh();
    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}

```

10.16.10.2.3 Pattern 3: Client Connection

```
void ConnectAndExecute(const char* host, const char* cmd) {
    SshClientConfig cfg = {host, 22, "admin", "password"};
    SSH_CLIENT* client = SshClientConnect(&cfg);
    if (client) {
        char buf[1024];
        SshClientExecute(client, cmd, buf, sizeof(buf));
        printf("%s\n", buf);
        SshClientDisconnect(client);
    }
}
```

10.16.10.3 Troubleshooting

Common SSH Issues & Solutions
Connection Refused - Check SSH server is started - Verify port is not blocked by firewall - Confirm device has network connectivity Authentication Failed - Verify credentials are correct - Check authentication callback is registered - Review UserAuthManager configuration - Check auth logs for details Key Not Found - Verify key callback is set (if using) - Check GetPrivateKeyPEM() override - Ensure auto-generation is enabled - Review key priority order Build Errors - Add required include paths: <code>\${NNDK_ROOT}/libraries/include/ssh</code> <code>\${NNDK_ROOT}/libraries/include/crypto</code> - Clean and rebuild project - Check for deprecated function usage Memory Issues - CryptoServer task runs at SECURITY_TASK_PRIO - Check available heap space - Limit concurrent SSH connections - Review cipher suite configuration

10.16.10.4 Debug Logging

Enable SSH debug output:

```
#include <ssh.h>

void UserMain(void* pd) {
    init();

    // Enable SSH debug logging
    SshSetDebugLevel(SSH_DEBUG_VERBOSE);

    // Levels:
    // SSH_DEBUG_NONE      - No debug output
    // SSH_DEBUG_ERROR     - Errors only
    // SSH_DEBUG_WARN      - Warnings and errors
    // SSH_DEBUG_INFO      - Info, warnings, and errors
    // SSH_DEBUG_VERBOSE   - All debug output

    StartSsh();

    while(1) { OSTimeDly(TICKS_PER_SECOND); }
}
```

10.16.10.5 Migration Checklist

When migrating from NNDK 3.3.5 or earlier:

Migration Steps:

```
[ ] Add include paths to project:
    ${NNDK_ROOT}/libraries/include/ssh
    ${NNDK_ROOT}/libraries/include/crypto
```

```

[ ] Remove deprecated functions:
    [ ] SshConvertDecodedOpenSSLKey()
    [ ] SshSetChansessionrequest()
    [ ] SshSetTaskPriority()

[ ] Update authentication code:
    [ ] Replace sshUserAuthenticateFn with
          sshUserAuthenticateWithTypeFn
    [ ] Replace SshSetUserAuthenticate\(\) with
          SshSetUserAuthenticateWithType\(\)
    [ ] Update callback to handle AuthType parameter

[ ] Test key management:
    [ ] Verify key installation method works
    [ ] Test auto-generated keys if used
    [ ] Confirm key format compatibility

[ ] Validate functionality:
    [ ] Test SSH server connections
    [ ] Verify authentication works
    [ ] Test SSH client if used
    [ ] Check memory usage

[ ] Review security:
    [ ] Update cipher suites if needed
    [ ] Review authentication policies
    [ ] Test key rotation procedures
    [ ] Verify logging and monitoring

```

10.16.11 Summary

This guide has covered:

- wolfSSH migration and benefits
- Supported key types and formats
- Migration from NNDK 3.3.5 to 3.3.6+
- Key management strategies
- User authentication with [UserAuthManager](#)
- Configuration and examples
- Best practices and troubleshooting

10.16.11.1 Key Takeaways

SSH Implementation Summary	
[MODERN]	wolfSSH replaces Dropbear
[CLIENT]	SSH client support now available
[SECURE]	Enhanced ciphers and key management
[AUTO]	Automatic key generation supported
[SHARED]	wolfCrypt shared with SSL/TLS
[MANAGED]	UserAuthManager simplifies auth
[FLEXIBLE]	Multiple key installation methods
[UPDATED]	New authentication API available

Recommended Approach:**

1. Use ECC keys for embedded systems
2. Implement key-based authentication
3. Use [UserAuthManager](#) for credentials
4. Enable auto-regeneration in production
5. Monitor authentication attempts
6. Keep NNDK updated

For the latest information and examples, refer to:

- `<nndk_install>/examples/SSH/`
- `<nndk_install>/include/ssh.h`
- NetBurner documentation at <https://www.netburner.com/learn>

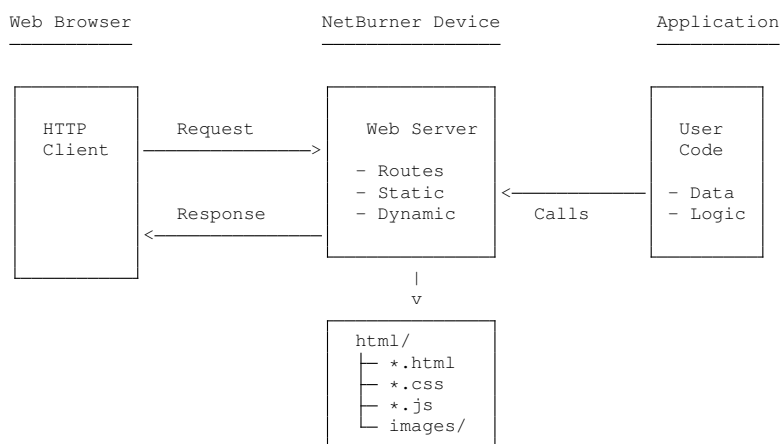
10.17 Web Server

10.17.1 NetBurner Web Server Guide

10.17.2 Overview

The NetBurner web server provides a powerful embedded HTTP server with support for both static and dynamic content. The web server architecture integrates seamlessly with your application, allowing real-time data display and interactive control through web pages.

NetBurner Web Server Architecture



10.17.2.1 Key Features

- **Static Content Serving** - HTML, CSS, JavaScript, images
- **Dynamic Content Generation** - Real-time data display
- **C++ Integration** - Direct callback functions
- **Variable Display** - Automatic type handling
- **URL Parsing** - Request parameter processing
- **File System** - Organized resource management

10.17.3 Basic Web Server Setup

10.17.3.1 Project Structure

When you create a NetBurner web application, the following structure is automatically generated:

Project Directory Structure

```

YourProject/
├── src/
│   ├── main.cpp          <— Application entry point
│   └── htmldata.cpp       <— Auto-generated from html/
├── html/
│   ├── index.html        <— Web content directory
│   ├── index.html        <— Default home page
│   ├── styles.css        <— (Optional) Stylesheets
│   ├── script.js         <— (Optional) JavaScript
│   └── images/           <— (Optional) Image assets
│       └── logo.png
  
```



```

├── icon.jpg
└── makefile
    └── Build configuration

```

10.17.3.2 Initial index.html

The Application Wizard creates a basic index.html file:

```

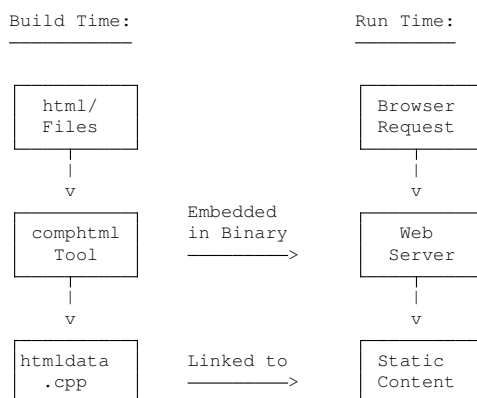
<html>
<body>
The main page for the AppWizard project.
</body>
</html>

```

10.17.3.3 Static Content

All files in the `html/` directory are compiled into your application binary. The web server serves these files when requested by a browser.

Static Content Flow



10.17.3.4 Adding Resources

10.17.3.4.1 Adding Images

Create an `images/` folder in your `html/` directory:

```

html/
├── index.html
├── images/
│   └── logo.jpg

```

Reference images in your HTML:

```

<html>
<body>
    <h1>The main page for the AppWizard project</h1>
    
</body>
</html>

```

10.17.3.4.2 Adding CSS and JavaScript

```

html/
├── index.html
├── styles.css
├── script.js
├── images/

```

styles.css:**

```

body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
}

```

index.html with resources:**

```

<html>
<head>
    <link rel="stylesheet" href="styles.css">
    <script src="script.js"></script>
</head>
<body>
    <h1>My NetBurner Application</h1>
    

```

```
</body>
</html>
```

10.17.3.5 Editing HTML Files in NBEclipse

Right-click on an HTML file and select "Open With" > "Text Editor" to edit the raw HTML content.

10.17.4 Dynamic Web Content

NetBurner provides three HTML tags for dynamic content generation:

Dynamic Content Tags

Tag Type	Purpose	When to Use
CPPCALL	Execute C++ function	Complex logic, control
VARIABLE	Display variable/expr	Data display, sensors
INCLUDE	Link external code	Variable declarations

10.17.4.1 CPPCALL Tag - C++ Callback Functions

The CPPCALL tag executes a C++ function when the web page is served.

10.17.4.1.1 Syntax

```
<!--CPPCALL YourFunctionName -->
```

10.17.4.1.2 Function Signature

```
void YourFunctionName(int sock, PCSTR url)
{
    // sock: TCP socket file descriptor
    // url: Complete URL string from browser
}
```

10.17.4.1.3 Parameters Explained

CPPCALL Function Parameters

Parameter: sock (int)

```
TCP Socket File Descriptor
- Use for writing output
- writestring(sock, ...)
- fprintf(sock, ...)
```

Parameter: url (PCSTR)

```
Complete URL String
"http://192.168.1.100/page?a=1"
- Parse query parameters
- Extract values after '?'
```

10.17.4.1.4 Basic Example

```
main.cpp:**
#include <nbrtos.h>
#include <init.h>
#include <http.h>

void webHelloWorld(int sock, PCSTR url)
{
    writestring(sock, "Hello World");
}

void UserMain(void *pd)
{
    init();
    StartHttp(); // Start web server on port 80

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

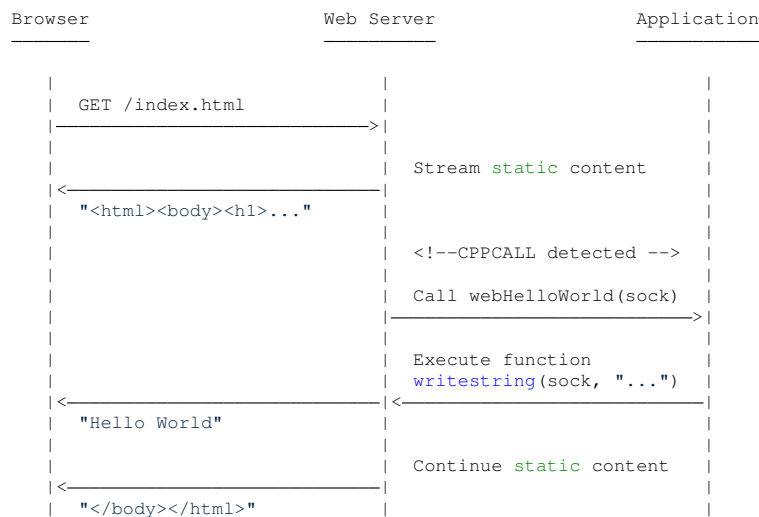
```

}
index.html:**
<html>
<body>
  <h1>The main page for the AppWizard project</h1>
  <p>Message from device: <!--CPPCALL webHelloWorld --></p>
</body>
</html>

```

10.17.4.1.5 Request Flow

CPPCALL Execution Flow



10.17.4.1.6 Advanced CPPCALL Example - URL Parsing

```

void webDeviceControl(int sock, PCSTR url)
{
    // Parse URL for parameters
    // Example: /index.html?action=toggle&pin=23

    if (strstr(url, "action=toggle")) {
        // Extract pin number
        char *pinParam = strstr(url, "pin=");
        if (pinParam) {
            int pinNum = atoi(pinParam + 4);

            // Toggle the pin
            Pins[pinNum].set(!Pins[pinNum].read());

            fprintf(sock, "Pin %d toggled to %s",
                    pinNum,
                    Pins[pinNum].read() ? "HIGH" : "LOW");
        }
    } else if (strstr(url, "action=status")) {
        fprintf(sock, "Device uptime: %lu seconds", Secs);
    } else {
        writestring(sock, "Unknown command");
    }
}

index.html:**
<html>
<body>
  <h1>Device Control</h1>
  <p>
    <a href="?action=toggle&pin=23">Toggle Pin 23</a><br>
    <a href="?action=status">Check Status</a>
  </p>
  <div>Response: <!--CPPCALL webDeviceControl --></div>
</body>
</html>

```

10.17.4.2 VARIABLE Tag - Displaying Variables

The VARIABLE tag displays application variables or expressions on web pages.

10.17.4.2.1 Syntax

```
<!--VARIABLE expression -->
```

10.17.4.2.2 Basic Usage

```
<!-- Display a simple variable -->
<!--VARIABLE TimeTick -->

<!-- Display an expression -->
<!--VARIABLE TimeTick/TICKS_PER_SECOND -->

<!-- Display with calculation -->
<!--VARIABLE (Temperature * 9/5) + 32 -->
```

10.17.4.2.3 Compilation Process

VARIABLE Tag Processing

HTML Source:	Generated Code:
<!--VARIABLE TimeTick -->	WriteHtmlVariable(fd, TimeTick);
<!--VARIABLE TimeTick/ TICKS_PER_SECOND -->	WriteHtmlVariable(fd, TimeTick/TICKS_PER_SECOND);
<!--VARIABLE IPCAST(IpAddress) -->	WriteHtmlVariable(fd, IPCAST(IpAddress));

10.17.4.2.4 Supported Data Types

The following types are automatically handled by WriteHtmlVariable():

```
// Located in: \nburn\include\htmlfiles.h

void WriteHtmlVariable(int fd, char c);
void WriteHtmlVariable(int fd, int i);
void WriteHtmlVariable(int fd, short i);
void WriteHtmlVariable(int fd, long i);
void WriteHtmlVariable(int fd, BYTE b);
void WriteHtmlVariable(int fd, WORD w);
void WriteHtmlVariable(int fd, unsigned long dw);
void WriteHtmlVariable(int fd, const char *);
void WriteHtmlVariable(int fd, MACADDR ip);
```

10.17.4.2.5 Type Handling Overview

Type Conversion Flow

Variable Type	Overloaded Function	Output
char	—> WriteHtmlVariable(fd, c)	—> 'A'
int	—> WriteHtmlVariable(fd, i)	—> 123
const char*	—> WriteHtmlVariable(fd, s)	—> "Hello"
unsigned long	—> WriteHtmlVariable(fd, ul)	—> 4294967295
MACADDR	—> WriteHtmlVariable(fd, mac)	—> 00:03:F4:...

10.17.4.2.6 IP Address Display

Use the IPCAST () helper to convert 32-bit integers to IP address format:

```
<!-- Without IPCAST - displays as integer -->
IP Address: <!--VARIABLE IpAddress -->
<!-- Output: 3232235777 -->

<!-- With IPCAST - displays in dotted notation -->
IP Address: <!--VARIABLE IPCAST(IpAddress) -->
<!-- Output: 192.168.1.1 -->
```

10.17.4.2.7 Complete Example

```
main.cpp:**
#include <init.h>
#include <nbrtos.h>

// Global variables for web display
```

```

int Temperature = 25;
unsigned long Counter = 0;
const char *DeviceStatus = "Running";

void UserMain(void *pd)
{
    init();
    StartHttp();

    while (1) {
        Counter++;
        Temperature = 20 + (Counter % 15); // Simulate changing temp
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

index.html:**

```

<html>
<head>
    <title>Device Status</title>
</head>
<body>
    <h1>Device Status Monitor</h1>

    <table border="1">
        <tr>
            <td>Uptime (seconds):</td>
            <td><!--VARIABLE Secs --></td>
        </tr>
        <tr>
            <td>System Ticks:</td>
            <td><!--VARIABLE TimeTick --></td>
        </tr>
        <tr>
            <td>Temperature (C):</td>
            <td><!--VARIABLE Temperature --></td>
        </tr>
        <tr>
            <td>Counter:</td>
            <td><!--VARIABLE Counter --></td>
        </tr>
        <tr>
            <td>Status:</td>
            <td><!--VARIABLE DeviceStatus --></td>
        </tr>
        <tr>
            <td>IP Address:</td>
            <td><!--VARIABLE IPCAST(IPAddress) --></td>
        </tr>
    </table>
</body>
</html>

```

10.17.4.3 INCLUDE Tag - Linking Variables

To use variables with the VARIABLE tag, your application must link to them. Two methods are available:

Variable Linking Methods

Method 1: htmlvar.h Header

```

Project/
├── src/
│   ├── main.cpp
│   └── htmlvar.h
└── html/
    └── index.html

```

←Link→

Both methods allow
htmldata.cpp to
link to your variables

Method 2: INCLUDE Tag

HTML File:

```
<!--INCLUDE mycustomheader.h -->
```

```

|
v
Project/
├── src/
│   └── mycustomheader.h

```

10.17.4.3.1 Method 1: htmlvar.h Header File

Create htmlvar.h in your project's src/ directory:

htmlvar.h:**

```

#ifndef HTMLVARS_H_
#define HTMLVARS_H_

#include <init.h>

// Declare variables for web display
extern int Temperature;

```

```

extern unsigned long Counter;
extern const char *DeviceStatus;

// Declare function callbacks
const char *webMyVarFunction(int fd, int value);

#endif // HTMLVARS_H_
Project structure:**
Project/
├── src/
│   ├── main.cpp
│   ├── htmlvar.h          <— Add this file
│   └── htmldata.cpp       <— Auto-generated (includes htmlvar.h)
└── html/
    └── index.html

```

10.17.4.3.2 Method 2: INCLUDE Tag in HTML

```

index.html:**
<html>
<!--INCLUDE customheader.h -->
<body>
  <h1>Temperature: <!--VARIABLE Temperature --> C</h1>
</body>
</html>

src/customheader.h:**
#ifndef CUSTOMHEADER_H_
#define CUSTOMHEADER_H_

extern int Temperature;

#endif

```

10.17.4.3.3 Linking Flow

Variable Linking at Compile Time

Step 1: HTML Processing

```
<!--VARIABLE Temp -->
```

Step 2: Code Generation

```
WriteHtmlVariable(fd, Temp);
```

Step 3: Include Headers

```
#include "htmlvar.h"
// or
#include "customheader.h"
```

Step 4: Link Variable

```
extern int Temp;
(declared in header)
```

Step 5: Compilation Success

```
htmldata.cpp links to Temp
in main.cpp
```

10.17.5 Advanced Techniques

10.17.5.1 Function Callbacks with Parameters

The VARIABLE tag can be used for function callbacks that accept parameters, providing more flexibility than CPP-CALL.

10.17.5.1.1 Comparison: CPPCALL vs VARIABLE Functions

CPPCALL vs VARIABLE Function Callbacks

CPPCALL:

Fixed Signature:
void func(int sock, PCSTR url)

Limited Parameters
Can't pass app data

Use for:
- URL parsing
- Control actions
- General callbacks

VARIABLE Function:

Flexible Signature:
const char* func(int fd, int val)
const char* func(int fd, float val)
etc.

Custom Parameters
Can pass any type

Use for:
- Parameterized display
- Formatted output
- Data transformation

10.17.5.1.2 Function Callback Workflow

VARIABLE Function Callback Flow

HTML:

```
<!--VARIABLE myFunc(fd, myVar) -->
```

v

Generated Code:

```
WriteHtmlVariable(fd, myFunc(fd, myVar));
```

v

Your Implementation:

```
const char* myFunc(int fd, int myVar) {
    fdprintf(fd, "Value: %d", myVar);
    return ""; // Return empty string
}
```

10.17.5.1.3 Example: Temperature Formatter

```
htmlvar.h:**
#ifndef HTMLVARS_H_
#define HTMLVARS_H_

#include <init.h>

extern int Temperature;

// Function callback with parameter
const char *FormatTemperature(int fd, int tempC);

#endif

main.cpp:**
#include <nbrtos.h>
#include <init.h>
#include "htmlvar.h"

int Temperature = 25;

const char *FormatTemperature(int fd, int tempC)
{
    // Write formatted output directly to socket
    fdprintf(fd, "<span style='color: ");

    if (tempC < 20) {
        fdprintf(fd, "blue'>Cold: %dC", tempC);
    } else if (tempC < 30) {
        fdprintf(fd, "green'>Normal: %dC", tempC);
    } else {
        fdprintf(fd, "red'>Hot: %dC", tempC);
    }

    fdprintf(fd, "</span>");

    // Return empty string (already wrote to socket)
    return "";
}
```

```

}

void UserMain(void *pd)
{
    init();
    StartHttp();

    while (1) {
        Temperature = 15 + (Secs % 25); // Vary temperature
        OSTimeDly(TICKS_PER_SECOND);
    }
}

index.html:**
<html>
<body>
    <h1>Temperature Monitor</h1>

    <!-- Pass fd and Temperature to function -->
    <p>Current: <!--VARIABLE FormatTemperature(fd, Temperature) --></p>

    <!-- Direct variable display for comparison -->
    <p>Raw value: <!--VARIABLE Temperature --> C</p>
</body>
</html>

```

10.17.5.1.4 Multi-Parameter Function Example

```

htmlvar.h:**
extern int SensorValue;
extern int SensorID;

const char *DisplaySensor(int fd, int id, int value);
main.cpp:**
const char *DisplaySensor(int fd, int id, int value)
{
    fdprintf(fd, "Sensor #%d: %d units", id, value);
    return "";
}

index.html:**
<p><!--VARIABLE DisplaySensor(fd, SensorID, SensorValue) --></p>

```

10.17.5.1.5 Return Value Handling

Function Return Value Options

Option 1: Write to Socket, Return Empty

```

const char *func(int fd, int val) {
    fdprintf(fd, "Data: %d", val);
    return ""; // Nothing more
}

```

↓
v
Output: "Data: 123"

Option 2: Return String for Printing

```

const char *func(int fd, int val) {
    static char buf[80];
    sprintf(buf, "Value: %d", val);
    return buf; // Printed by web
}

```

↓
v
Output: "Value: 123"

Option 3: Both (Write + Return)

```

const char *func(int fd, int val) {
    fdprintf(fd, "Start: ");
    return "End"; // Appended
}

```

↓
v
Output: "Start: End"

10.17.5.2 Custom Data Types

Extend the VARIABLE tag to support user-defined structures and classes.

10.17.5.2.1 Custom Type Flow

Custom Type Support Architecture

Step 1: Define Your Type

```
struct MY_STRUCT {
    int value;
    char name[40];
};

MY_STRUCT myData;
```



Step 2: Declare in htmlvar.h

```
extern MY_STRUCT myData;

void WriteHtmlVariable(int fd,
                      MY_STRUCT s);
```



Step 3: Implement Function

```
void WriteHtmlVariable(int fd,
                      MY_STRUCT s) {
    fprintf(fd, "%s: %d",
           s.name, s.value);
}
```



Step 4: Use in HTML

```
<!--VARIABLE myData -->
```

10.17.5.2.2 Complete Custom Type Example

```
main.cpp:**
#include <nbrtos.h>
#include <init.h>

// Define custom structure
struct SensorData {
    int temperature;
    char location[80];
    uint32_t timestamp;
};

SensorData hallwaySensor;
SensorData laboratorySensor;

// Overload WriteHtmlVariable for custom type
void WriteHtmlVariable(int fd, SensorData sensor)
{
    fprintf(fd, "<div class='sensor'>");
    fprintf(fd, "    <strong>Location:</strong> %s<br>", sensor.location);
    fprintf(fd, "    <strong>Temperature:</strong> %dC<br>", sensor.temperature);
    fprintf(fd, "    <strong>Reading Time:</strong> %lu seconds<br>", sensor.timestamp);
    fprintf(fd, "</div>");
}

void UserMain(void *pd)
{
    init();
    StartHttp();

    // Initialize sensors
    strcpy(hallwaySensor.location, "Main Hallway");
    strcpy(laboratorySensor.location, "Laboratory");

    while (1) {
        hallwaySensor.temperature = 20 + (Secs % 10);
        hallwaySensor.timestamp = Secs;

        laboratorySensor.temperature = 18 + (Secs % 8);
```

```

        laboratorySensor.timestamp = Secs;

        OSTimeDly(TICKS_PER_SECOND);
    }
}

htmlvar.h:**
#ifndef HTMLVARS_H_
#define HTMLVARS_H_

#include <init.h>

struct SensorData {
    int temperature;
    char location[80];
    uint32_t timestamp;
};

extern SensorData hallwaySensor;
extern SensorData laboratorySensor;

// Declare overloaded function
void WriteHtmlVariable(int fd, SensorData sensor);

#endif

index.html:**
<html>
<head>
    <style>
        .sensor {
            border: 1px solid #ccc;
            padding: 10px;
            margin: 10px 0;
            background-color: #f9f9f9;
        }
    </style>
</head>
<body>
    <h1>Multi-Sensor Dashboard</h1>

    <h2>Sensor 1</h2>
    <!--VARIABLE hallwaySensor -->

    <h2>Sensor 2</h2>
    <!--VARIABLE laboratorySensor -->

    <hr>
    <p><small>System Time: <!--VARIABLE Secs --> seconds</small></p>
</body>
</html>

```

10.17.5.2.3 Class-Based Custom Type

```

class DeviceStatus {
public:
    bool online;
    int errorCount;
    const char *version;

    DeviceStatus() : online(true), errorCount(0), version("1.0.0") {}
};

DeviceStatus systemStatus;

// Overload for class
void WriteHtmlVariable(int fd, DeviceStatus status)
{
    fprintf(fd, "<table>");
    fprintf(fd, "<tr><td>Status:</td><td>%s</td></tr>",
        status.online ? "Online" : "Offline");
    fprintf(fd, "<tr><td>Errors:</td><td>%d</td></tr>",
        status.errorCount);
    fprintf(fd, "<tr><td>Version:</td><td>%s</td></tr>",
        status.version);
    fprintf(fd, "</table>");
}

HTML usage:**
<div>
    <h3>System Status</h3>
    <!--VARIABLE systemStatus -->
</div>

```

10.17.6 Best Practices

10.17.6.1 Project Organization

Recommended Project Structure

```

YourProject/
├── src/
│   ├── main.cpp           <— Core application logic
│   ├── htmlvar.h         <— Web variable declarations
│   ├── web_callbacks.cpp  <— CPPCALL implementations
│   ├── web_callbacks.h    <— CPPCALL declarations
│   └── htmldata.cpp       <— Auto-generated (don't edit)
├── html/
│   ├── index.html        <— Main page
│   ├── control.html      <— Control interface
│   ├── status.html       <— Status display
│   ├── styles/
│   │   └── main.css       <— Stylesheets
│   ├── scripts/
│   │   └── main.js        <— JavaScript
│   ├── images/
│   │   ├── logo.png
│   │   └── icons/
└── makefile

```

10.17.6.2 Code Organization

10.17.6.2.1 Separate Web Logic

```

web_callbacks.h:**
#ifndef WEB_CALLBACKS_H_
#define WEB_CALLBACKS_H_

// CPPCALL functions
void webDeviceControl(int sock, PCSTR url);
void webSystemStatus(int sock, PCSTR url);
void webSensorData(int sock, PCSTR url);

// VARIABLE functions
const char *FormatTimestamp(int fd, unsigned long timestamp);
const char *FormatIPAddress(int fd, uint32_t ip);

#endif

web_callbacks.cpp:**
#include "web_callbacks.h"
#include <nbrtos.h>

void webDeviceControl(int sock, PCSTR url)
{
    // Implementation
}

void webSystemStatus(int sock, PCSTR url)
{
    // Implementation
}

// ... etc

```

10.17.6.3 Performance Optimization

Web Server Performance Tips

1. Minimize CPPCALL Processing

```

void fastCallback(int sock,
                  PCSTR url)
{
    // Quick processing
    writestring(sock, data);
    // Exit immediately
}

```

↓
v
Don't block web server thread!

2. Use VARIABLE for Simple Data

```
<!--VARIABLE Temperature -->
```

↓
v

Faster than CPPCALL for display

3. Cache Formatted Strings

```
static char statusBuf[256];
// Update periodically
// in background task
```

↓
v

Reduce real-time formatting

4. Limit Dynamic Content

```
Use static content where
possible (HTML, CSS, JS)
```

10.17.6.4 Security Considerations

Web Security Best Practices

Input Validation

```
void callback(
    int sock,
    PCSTR url) {

    // Validate input
    if (!IsValid(url))
    {
        writestring(
            sock,
            "Invalid");
        return;
    }
}
```

URL Parameter Sanitization

```
void handler(int sock,
             PCSTR url)
{
    // BAD: Direct use
    system(url); // NO!

    // GOOD: Parse & check
    char *param =
        parseParam(url);
    if (isSafe(param)) {
        // Use safely
    }
}
```

Buffer Overflow Prevention

```
char buf[256];

// BAD
sprintf(buf, ...);

// GOOD
sprintf(buf,
        sizeof(buf), ...);
```

Authentication

```
bool authenticated =
    CheckCredentials(url);

if (!authenticated) {
    writestring(sock,
        "Access Denied");
    return;
}
```

10.17.6.5 Error Handling

```
void robustCallback(int sock, PCSTR url)
{
    // Check socket validity
    if (sock < 0) {
        return;
    }

    // Validate URL
    if (!url || strlen(url) == 0) {
        writestring(sock, "Error: Invalid request");
        return;
    }

    // Safe parameter parsing
    char *param = strstr(url, "value=");
    if (!param) {
        writestring(sock, "Error: Missing parameter");
    }
}
```

```

        return;
    }

    // Process with bounds checking
    int value = atoi(param + 6);
    if (value < 0 || value > 100) {
        fdprintf(sock, "Error: Value %d out of range", value);
        return;
    }

    // Success
    fdprintf(sock, "Success: Value set to %d", value);
}

```

10.17.6.6 HTML Best Practices

```

<!-- Good: Clean, organized structure -->
<html>
<head>
    <title>Device Control</title>
    <link rel="stylesheet" href="styles/main.css">
</head>
<body>
    <header>
        <h1>NetBurner Device</h1>
        <p>System Time: <!--VARIABLE Secs --> seconds</p>
    </header>

    <main>
        <section id="status">
            <h2>Status</h2>
            <!--CPPCALL webSystemStatus -->
        </section>

        <section id="control">
            <h2>Control</h2>
            <!--CPPCALL webDeviceControl -->
        </section>
    </main>

    <footer>
        <p>Version: <!--VARIABLE FirmwareVersion --></p>
    </footer>

    <script src="scripts/main.js"></script>
</body>
</html>

```

10.17.6.7 Testing Strategy

Web Application Testing Flow

1. Unit Test Callbacks

Test each CPPCALL
function in
isolation



2. Validate Output

Verify VARIABLE tags
display correctly



3. Test URL Parsing

Various query params
Edge cases



4. Load Testing

Multiple concurrent
connections



5. Security Testing

Invalid inputs

Buffer limits

10.17.6.8 Debugging Tips

```
// Enable debug output in callbacks
void debugCallback(int sock, PCSTR url)
{
    printf("Callback called: %s\r\n", url); // To serial console

    // Parse and display
    char *param = strstr(url, "cmd=");
    if (param) {
        printf("Command parameter: %s\r\n", param);
    }

    // Write response
    writestring(sock, "Response sent");
    printf("Response complete\r\n");
}
```

10.17.6.9 Common Pitfalls to Avoid

Common Mistakes and Solutions

MISTAKE

Using printf() in callback

```
void bad(int sock) {
    printf("Hi");
}
```

Output goes nowhere!

SOLUTION

Use writestring() or fdprintf()

```
void good(int sock) {
    writestring(sock,
        "Hi");
}
```

Output to browser

Forgetting extern

```
// In main.cpp
int Temperature;

// Link error!
```

Declare in htmlvar.h

```
// In htmlvar.h
extern int
    Temperature;
```

htmldata.cpp can link

Blocking in callback

```
void bad(int sock) {
    OSTimeDly(...);
    // Blocks server!
}
```

Quick exit pattern

```
void good(int sock) {
    writestring(sock,
        "OK");
    // Exit fast
}
```

Not validating input

```
int val =
    atoi(url+5);
// May crash!
```

Always validate

```
char *p = strstr(url,
    "val=");
if (p) {
    int val=atoi(p+4);
}
```

10.17.7 Quick Reference

10.17.7.1 Tag Summary

NetBurner Web Tags Quick Reference

Tag	Purpose	Syntax
CPPCALL	Execute C++ function	<!--CPPCALL FuncName --> void FuncName(int sock, PCSTR url)
VARIABLE	Display variable	<!--VARIABLE VarName -->

```

                                or expression      <!--VARIABLE expr -->

INCLUDE                        Link header file    <!--INCLUDE header.h -->
                                for variable access

VARIABLE Function             Parameterized        <!--VARIABLE Func(fd, var) -->
                                callback            const char* Func(int fd,
                                                    int var)

```

10.17.7.2 Function Signatures

```

// CPPCALL signature
void CallbackFunction(int sock, PCSTR url);

// VARIABLE function signature
const char *VariableFunction(int fd, <param_type> param);

// Custom type support
void WriteHtmlVariable(int fd, CustomType data);

```

10.17.7.3 Common Patterns

```

// Pattern 1: Simple status display
void webStatus(int sock, PCSTR url) {
    fdprintf(sock, "Uptime: %lu seconds", Secs);
}

// Pattern 2: URL parameter parsing
void webControl(int sock, PCSTR url) {
    if (strstr(url, "action=reset")) {
        // Handle reset
        writestring(sock, "Device reset");
    }
}

// Pattern 3: Variable formatting
const char *FormatData(int fd, int value) {
    fdprintf(fd, "<b>%d</b>", value);
    return "";
}

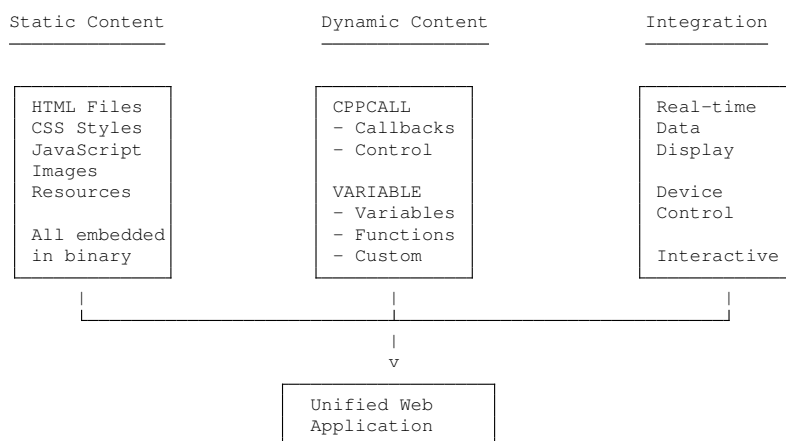
// Pattern 4: Custom type display
void WriteHtmlVariable(int fd, MyStruct s) {
    fdprintf(fd, "%s: %d", s.name, s.value);
}

```

10.17.8 Summary

The NetBurner web server provides a powerful platform for embedded web applications:

Web Server Capabilities Overview



10.17.8.1 Key Takeaways

1. **Static content** is compiled into the application binary
2. **CPPCALL** provides flexible callback functions for dynamic control
3. **VARIABLE** tags display real-time data with automatic type handling

4. **Custom types** can be added by overloading WriteHtmlVariable()
5. **Security** and input validation are critical for robust applications
6. **Performance** is maintained by keeping callbacks fast and efficient

10.18 Web Sockets

10.18.1 WebSocket Protocol Implementation for Real-Time Bidirectional Communication

10.18.2 Overview

The NetBurner WebSocket implementation provides full-duplex communication channels over a single TCP connection, enabling real-time data exchange between clients and servers. WebSockets upgrade from HTTP connections and maintain persistent connections for efficient, low-latency bidirectional messaging.

Key features include:

- RFC 6455 compliant WebSocket protocol implementation
- Automatic connection upgrade from HTTP to WebSocket
- Support for text and binary data frames
- Built-in ping/pong for connection health monitoring
- Efficient buffering and flow control
- Support for both client and server roles
- Graceful connection closure with status codes
- Frame fragmentation for large messages

10.18.3 Typical WebSocket Workflow

Server Side:**

1. Register a WebSocket upgrade handler using SetNewWSHandler()
2. In the handler, call [WSUpgrade\(\)](#) to upgrade the HTTP connection
3. Use WSRead() and WSWrite() for bidirectional communication
4. Send periodic [WSPing\(\)](#) frames to monitor connection health
5. Close gracefully with [close\(\)](#) when done

Client Side:**

1. Call WebSocket::Connect() with server hostname/IP and resource path
2. Receive a WebSocket file descriptor on success
3. Use WSRead() and WSWrite() for communication
4. Monitor connection with [WSPing\(\)](#) and [WSWaitForPingReply\(\)](#)
5. Close gracefully with [close\(\)](#) when done

10.18.4 WebSocket Frame Types

The WebSocket protocol supports several frame types:

- **Text frames** (WS_OP_TEXT): UTF-8 encoded text data
- **Binary frames** (WS_OP_BIN): Raw binary data
- **Continuation frames** (WS_OP_CONT): Fragments of a larger message
- **Close frames** (WS_OP_CLOSE): Graceful connection termination
- **Ping frames** (WS_OP_PING): Connection liveness check
- **Pong frames** (WS_OP_PONG): Response to ping frame

Most applications only need to work with text and binary frames, as control frames (ping, pong, close) are handled automatically by the implementation.

10.18.5 Buffering and Flow Control

The WebSocket implementation uses three internal buffers:

- **RX Buffer**: Incoming data from the TCP connection
- **TX Buffer**: Outgoing data waiting to be sent
- **Control Buffer**: Control frames (ping, pong, close)

Buffer sizes are configurable via compile-time constants:

- WS_BUFFER_SEGMENTS: Number of buffer segments (default: 4)
- WS_BUFFER_MIN: Minimum buffer threshold (default: 10)

Use `GetWriteSpace()` to check available TX buffer space before writing large amounts of data.

10.18.6 Connection Health Monitoring

WebSockets provide built-in mechanisms for connection health monitoring:

Ping/Pong Mechanism:**

- [`WSPing\(\)`](#): Send a ping frame to test connection
- [`WSWaitForPingReply\(\)`](#): Block until pong received or timeout
- [`WSGetPingReplyTick\(\)`](#): Retrieve timestamp of last pong for latency calculation

Recommended practice:

- Send periodic pings (e.g., every 30-60 seconds)
- Measure round-trip time to detect network degradation
- Close connection if multiple pings timeout
- Use adaptive ping intervals based on application requirements

10.18.7 Graceful Connection Closure

WebSocket connections should be closed gracefully:

1. Call [`close\(\)`](#) on the WebSocket file descriptor
2. The implementation sends a close frame with an optional status code
3. The peer responds with a close frame

4. The TCP connection is terminated

Common status codes (WS_StatusCode enum):

- WS_STAT_NORM_CLOSE (1000): Normal closure
- WS_STAT_GOING_AWAY (1001): Endpoint going away (server shutdown, browser navigation)
- WS_STAT_PROT_ERROR (1002): Protocol error
- WS_STAT_MSG_TOO_BIG (1009): Message too large to process
- WS_STAT_UNEXPECTED_COND (1011): Unexpected condition prevented request fulfillment

10.18.8 Thread Safety

The WebSocket implementation uses critical sections (OS_CRIT) to protect shared data structures. Multiple tasks can safely operate on different WebSocket connections. However, operations on the same WebSocket from multiple tasks require external synchronization.

Best practices:

- Use one task per WebSocket connection for simplicity
- If multiple tasks access the same WebSocket, use application-level locking
- The implementation handles internal buffer synchronization automatically

10.18.9 Resource Limits

The implementation has compile-time limits:

- Maximum WebSocket connections: WS_MAX_SOCKS (default: 4)
- Check available slots with GetFreeWebSocketCount()
- Each WebSocket consumes buffer memory and a file descriptor
- Consider system resources when determining WS_MAX_SOCKS value

10.18.10 Error Handling

WebSocket functions return negative values on errors:

- Check return values from all WebSocket functions
- WSRead() returns 0 for connection closure, negative for errors
- WSWrite() returns negative if connection is closed or buffer full
- Use errno or implementation-specific error codes for details

Common error scenarios:

- Connection closed by peer (WSRead returns 0 or negative)
- Write buffer full (WSWrite returns negative, check with GetWriteSpace())
- Invalid frame received (protocol error, connection closed)
- Timeout on blocking operations

10.18.11 Security Considerations

WebSocket security best practices:

- Use WSS (WebSocket Secure) over TLS for production deployments
- Validate Origin headers to prevent CSRF attacks
- Implement authentication before [WSUpgrade\(\)](#)
- Rate-limit connections and messages to prevent DoS
- Validate and sanitize all received data
- Set maximum message size limits
- Use secure protocols for sensitive data
- Implement connection timeouts and limits

10.18.12 Performance Optimization

Tips for optimal WebSocket performance:

- Batch small messages when possible to reduce overhead
- Use binary frames for non-text data (more efficient than base64 text)
- Call Flush() explicitly when immediate transmission is needed
- Adjust WS_BUFFER_SEGMENTS for your message patterns
- Use GetWriteSpace() to avoid blocking on full buffers
- Consider message compression for large text payloads (if supported)
- Reuse connections rather than frequent connect/disconnect cycles

10.18.13 Debugging WebSocket Issues

Debug utilities:

- DumpSock(): Display detailed state of a single WebSocket
- DumpSockets(): Display state of all WebSocket connections
- Monitor buffer levels with GetWriteSpace()
- Track connection state with GetState()
- Use ping/pong to measure latency and detect connection issues
- Enable diagnostic output to track frame exchanges

Expand for Example Usage

Examples

Example 1: Basic WebSocket Echo Server

```

#include <init.h>
#include <nbrtos.h>
#include <http.h>
#include <websockets.h>

const char *AppName = "WebSocket Echo Server";

// WebSocket task for each connection
void WebSocketEchoTask(void *pd) {
    int wsFd = (int)(intptr_t)pd;
    char buffer[512];

    iprintf("WebSocket echo task started on fd %d\r\n", wsFd);

    while (1) {
        // Read incoming message with 30 second timeout
        int len = WSRead(wsFd, buffer, sizeof(buffer) - 1, TICKS_PER_SECOND * 30);

        if (len > 0) {
            buffer[len] = '\0';
            iprintf("Received: %s\r\n", buffer);

            // Echo the message back
            if (WSWrite(wsFd, buffer, len) < 0) {
                iprintf("Failed to send echo\r\n");
                break;
            }
        } else if (len == 0) {
            iprintf("WebSocket connection closed by peer\r\n");
            break;
        } else {
            iprintf("WebSocket read error or timeout\r\n");
            break;
        }
    }

    close(wsFd);
    iprintf("WebSocket echo task ended\r\n");
}

// HTTP handler for WebSocket upgrade
int HandleWebSocketUpgrade(HTTP_Request *req, int sock) {
    // Upgrade the HTTP connection to WebSocket
    if (WSUpgrade(req, sock) >= 0) {
        iprintf("WebSocket connection established\r\n");

        // Create a task to handle this WebSocket
        OSSimpleTaskCreateWithName(WebSocketEchoTask,
                                   MAIN_PRIO - 1,
                                   (void *) (intptr_t) sock,
                                   "WSEcho");

        return 0; // Success
    } else {
        iprintf("WebSocket upgrade failed\r\n");
        return -1;
    }
}

void UserMain(void *pd) {
    init();
    StartHttp();

    // Register WebSocket endpoint at /ws
    SetNewWSHandler("/ws", HandleWebSocketUpgrade);

    iprintf("%s\r\n", AppName);
    iprintf("WebSocket endpoint: ws://<device-ip>/ws\r\n");

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Example 2: WebSocket Client Connection

```

#include <websockets.h>

void WebSocketClientTask(void *pd) {
    // Connect to WebSocket server
    int wsFd = NB::WebSocket::Connect("ws.example.com",
                                       "/api/v1/stream",
                                       "chat", // protocol

```

```

        80,        // port
        false);   // useSSL

if (wsFd < 0) {
    fprintf("Failed to connect to WebSocket server\r\n");
    return;
}

fprintf("Connected to WebSocket server (fd %d)\r\n", wsFd);

// Send a message
const char *msg = "Hello from NetBurner!";
if (WSWrite(wsFd, msg, strlen(msg)) < 0) {
    fprintf("Failed to send message\r\n");
    close(wsFd);
    return;
}

// Wait for response
char buffer[256];
int len = WSRead(wsFd, buffer, sizeof(buffer) - 1, TICKS_PER_SECOND * 10);

if (len > 0) {
    buffer[len] = '\0';
    fprintf("Server response: %s\r\n", buffer);
}

// Clean up
close(wsFd);
fprintf("WebSocket client disconnected\r\n");
}

```

Example 3: WebSocket with Connection Health Monitoring

```

void MonitoredWebSocketTask(void *pd) {
    int wsFd = (int)(intptr_t)pd;
    uint32_t lastPingTime = Secs;
    uint32_t consecutiveFailures = 0;
    const uint32_t PING_INTERVAL_SEC = 30;
    const uint32_t MAX_FAILURES = 3;
    char buffer[512];

    while (1) {
        // Send periodic pings
        if ((Secs - lastPingTime) >= PING_INTERVAL_SEC) {
            uint32_t pingSent;

            if (WSPing(wsFd, 0, &pingSent) >= 0) {
                // Wait for pong with timeout
                if (WSWaitForPingReply(wsFd, TICKS_PER_SECOND * 5) >= 0) {
                    uint32_t pongReceived;
                    if (WSGetPingReplyTick(wsFd, &pongReceived) >= 0) {
                        uint32_t rtt = (pongReceived - pingSent) / (TICKS_PER_SECOND / 1000);
                        fprintf("Ping successful, RTT: %lu ms\r\n", rtt);
                    }
                    consecutiveFailures = 0;
                } else {
                    fprintf("Ping timeout\r\n");
                    consecutiveFailures++;
                }
            } else {
                fprintf("Failed to send ping\r\n");
                consecutiveFailures++;
            }
        }

        lastPingTime = Secs;

        // Check if connection is dead
        if (consecutiveFailures >= MAX_FAILURES) {
            fprintf("Connection appears dead, closing\r\n");
            break;
        }
    }

    // Read messages with short timeout to allow ping checks
    int len = WSRead(wsFd, buffer, sizeof(buffer) - 1, TICKS_PER_SECOND);

    if (len > 0) {
        buffer[len] = '\0';
        fprintf("Received: %s\r\n", buffer);
    } else if (len < 0) {
        fprintf("Read error or connection closed\r\n");
        break;
    }
}

```

```
    close(wsFd);
}
```

Example 4: WebSocket Broadcast Server

```
#define MAX_WS_CLIENTS 10

struct WSClient {
    int fd;
    bool active;
};

WSClient wsClients[MAX_WS_CLIENTS];
OS_CRIT wsClientsCrit;

// Add client to broadcast list
bool AddWSClient(int fd) {
    OSCritEnter(&wsClientsCrit, TICKS_PER_SECOND);

    for (int i = 0; i < MAX_WS_CLIENTS; i++) {
        if (!wsClients[i].active) {
            wsClients[i].fd = fd;
            wsClients[i].active = true;
            OSCritLeave(&wsClientsCrit);
            return true;
        }
    }

    OSCritLeave(&wsClientsCrit);
    return false;
}

// Remove client from broadcast list
void RemoveWSClient(int fd) {
    OSCritEnter(&wsClientsCrit, TICKS_PER_SECOND);

    for (int i = 0; i < MAX_WS_CLIENTS; i++) {
        if (wsClients[i].active && wsClients[i].fd == fd) {
            wsClients[i].active = false;
            break;
        }
    }

    OSCritLeave(&wsClientsCrit);
}

// Broadcast message to all connected clients
void BroadcastMessage(const char *msg, int len) {
    OSCritEnter(&wsClientsCrit, TICKS_PER_SECOND);

    for (int i = 0; i < MAX_WS_CLIENTS; i++) {
        if (wsClients[i].active) {
            if (WSWrite(wsClients[i].fd, msg, len) < 0) {
                fprintf("Failed to send to client %d\r\n", wsClients[i].fd);
            }
        }
    }

    OSCritLeave(&wsClientsCrit);
}

// Task for each WebSocket client
void WSClientTask(void *pd) {
    int wsFd = (int)(intptr_t)pd;
    char buffer[512];

    if (!AddWSClient(wsFd)) {
        fprintf("Too many clients, rejecting connection\r\n");
        close(wsFd);
        return;
    }

    fprintf("Client connected: fd %d\r\n", wsFd);

    while (1) {
        int len = WSRead(wsFd, buffer, sizeof(buffer) - 1, TICKS_PER_SECOND * 30);

        if (len > 0) {
            buffer[len] = '\0';
            fprintf("Broadcasting message from fd %d: %s\r\n", wsFd, buffer);

            // Broadcast to all clients
            BroadcastMessage(buffer, len);
        } else {

```

```

        iprintf("Client disconnected: fd %d\r\n", wsFd);
        break;
    }
}

RemoveWSClient(wsFd);
close(wsFd);
}

int HandleBroadcastWS(HTTP_Request *req, int sock) {
    if (WSUpgrade(req, sock) >= 0) {
        OSSimpleTaskCreateWName(WSCliTask, MAIN_PRIO - 1,
                                (void *) (intptr_t) sock, "WSCliTask");
        return 0;
    }
    return -1;
}

```

Example 5: WebSocket with Authentication and Options

```

#include <http.h>
#include <websockets.h>

// Validate authentication token
bool ValidateAuth(HTTP_Request *req) {
    // Check for Authorization header or query parameter
    const char *token = GetRequestParam(req, "token");
    if (token == NULL) {
        return false;
    }

    // Validate token (simplified)
    return (strcmp(token, "secret_key_12345") == 0);
}

void AuthenticatedWSTask(void *pd) {
    int wsFd = (int) (intptr_t) pd;
    char buffer[1024];

    // Set WebSocket options for text mode
    ws_setopt(wsFd, WS_SO_TEXT);

    iprintf("Authenticated WebSocket connection established\r\n");

    // Send welcome message
    const char *welcome = "{\"type\":\"welcome\",\"message\":\"Connected to secure endpoint\"}";
    WSWrite(wsFd, welcome, strlen(welcome));

    while (1) {
        int len = WSRead(wsFd, buffer, sizeof(buffer) - 1, TICKS_PER_SECOND * 60);

        if (len > 0) {
            buffer[len] = '\0';

            // Process JSON messages or other application protocol
            iprintf("Secure message: %s\r\n", buffer);

            // Echo with status
            char response[1024];
            snprintf(response, sizeof(response),
                    "{\"type\":\"ack\",\"received\":%d}", len);
            WSWrite(wsFd, response, strlen(response));
        } else if (len == 0) {
            iprintf("Client disconnected gracefully\r\n");
            break;
        } else {
            iprintf("Error or timeout\r\n");
            break;
        }
    }

    close(wsFd);
}

int HandleSecureWS(HTTP_Request *req, int sock) {
    // Authenticate before upgrade
    if (!ValidateAuth(req)) {
        iprintf("Authentication failed\r\n");

        writestring(sock, "HTTP/1.1 401 Unauthorized\r\n");
        writestring(sock, "Content-Type: text/plain\r\n\r\n");
        writestring(sock, "Authentication required");
        return -1;
    }
}

```

```

// Upgrade to WebSocket
if (WSUpgrade(req, sock) >= 0) {
    OSSimpleTaskCreateName(AuthenticatedWSTask, MAIN_PRIO - 1,
        (void *) (intptr_t) sock, "WSAuth");
    return 0;
}

return -1;
}

```

Function	Description
WSUpgrade()	Upgrade HTTP connection to WebSocket
WSRead()	Read data from WebSocket (ws_read wrapper)
WSWrite()	Write data to WebSocket (ws_write wrapper)
WSPing()	Send ping frame for connection health check
WSWaitForPingReply()	Wait for pong response with timeout
WSGetPingReplyTick()	Get timestamp of last pong for latency measurement
SetNewWSHandler()	Register WebSocket upgrade handler
NB::WebSocket::Connect()	Create client WebSocket connection
close()	Close WebSocket connection

10.18.14 Configuration and Return Code Definitions

```

#define WS_BUFFER_SEGMENTS 4
#define WS_MAX_SOCKETS 4
#define WS_BUFFER_MIN 10
#define WS_ACCUM_DLY ((TICKS_PER_SECOND / 4) + 1)
#define WS_FLUSH_TIMEOUT 0

#define WS_FIN_BIT 0x80
#define WS_OP_CONT 0x0
#define WS_OP_TEXT 0x1
#define WS_OP_BIN 0x2
#define WS_OP_CLOSE 0x8
#define WS_OP_PING 0x9
#define WS_OP_PONG 0xA

#define WS_SO_TEXT 0x8

```

10.18.14.1 Number of buffer segments allocated for each WebSocket connection.

Each WebSocket connection uses segmented buffers for efficient memory management and flow control. This value determines how many segments are allocated for the receive (RX), transmit (TX), and control buffers.

```
#define WS_BUFFER_SEGMENTS 4
```

Default Value:** 4 segments per buffer

Memory Impact:**

- Each connection uses WS_BUFFER_SEGMENTS segments for three buffers (RX, TX, Control)
 - Total segments per connection: WS_BUFFER_SEGMENTS x 3
 - Adjust based on available memory and expected message sizes
- Tuning Guidelines:**
- Increase for applications with large messages or high throughput
 - Decrease to conserve memory when handling many small messages
 - Each segment is typically sized based on the buffer implementation
 - Consider buffer exhaustion when multiple connections are active

Note

This is a compile-time constant and cannot be changed at runtime

Changing this value requires recompiling the WebSocket library

See also

[WS_MAX SOCKS](#)

[WS_BUFFER_MIN](#)

10.18.14.2 Maximum number of simultaneous WebSocket connections supported.

This defines the compile-time limit for concurrent WebSocket connections that can be active on the device. Each connection consumes memory for buffers and internal state management.

`define WS_MAX SOCKS 4`

Default Value:** 4 connections

Resource Considerations:**

- Each connection uses approximately (WS_BUFFER_SEGMENTS x 3) buffer segments
- Each connection requires a file descriptor and internal state structure
- Total memory usage scales linearly with WS_MAX SOCKS

Tuning Guidelines:**

- Increase for servers handling multiple simultaneous clients
- Decrease to conserve memory on resource-constrained devices
- Use `GetFreeWebSocketCount()` to check available connection slots at runtime
- Consider total system file descriptors and memory when adjusting

Example Memory Calculation:**

```
// Approximate per-connection overhead:
// - Buffers: WS_BUFFER_SEGMENTS x 3 x segment_size
// - State: sizeof(WebSocket) ~= 200+ bytes
// - File descriptor: 1 fd
//
// For WS_MAX SOCKS = 4:
// - Total: 4 connections x overhead_per_connection
```

Note

This is a compile-time constant and cannot be changed at runtime

Exceeding this limit will cause new connection attempts to fail

Use `GetFreeWebSocketCount()` to monitor available slots

See also

[WS_BUFFER_SEGMENTS](#)

`GetFreeWebSocketCount()`

Expand for Example Usage

Examples

Example: Checking Available Connection Slots

```
int HandleWebSocketUpgrade(HTTP_Request *req, int sock) {
    // Check if we can accept more connections
    int freeSlots = NB::WebSocket::GetFreeWebSocketCount();

    if (freeSlots <= 0) {
        iprintf("No WebSocket slots available (%d/%d used)\r\n",
            WS_MAX SOCKS - freeSlots, WS_MAX SOCKS);

        writestring(sock, "HTTP/1.1 503 Service Unavailable\r\n");
        writestring(sock, "Content-Type: text/plain\r\n\r\n");
        writestring(sock, "Maximum WebSocket connections reached");
        return -1;
    }

    iprintf("Accepting WebSocket connection (%d slots remaining)\r\n", freeSlots);

    if (WSUpgrade(req, sock) >= 0) {
```

```

        OSSimpleTaskCreateName(WebSocketTask, MAIN_PRIO - 1,
                               (void *) (intptr_t) sock, "WSTask");
    return 0;
}

return -1;
}

```

10.18.14.3 Minimum buffer space threshold before triggering buffer management actions.

This defines the minimum number of bytes that should remain available in the transmit buffer before the WebSocket implementation takes buffer management actions, such as flushing pending data or blocking writes.

define WS_BUFFER_MIN 10

Default Value:** 10 bytes

Purpose:**

- Prevents buffer exhaustion by maintaining a minimum safety margin
 - Triggers automatic flush operations when buffer space is low
 - Helps prevent write operations from blocking unnecessarily
- Buffer Management:**
- When available space falls below this threshold, writes may block or flush
 - Use GetWriteSpace() to check available buffer space before large writes
 - Automatic flushing helps maintain data flow even with small buffer margins

Note

This is a compile-time constant affecting buffer management behavior

Setting too high may reduce effective buffer capacity

Setting too low may increase risk of buffer overrun

See also

GetWriteSpace()

Flush()

10.18.14.4 Accumulation delay for batching small writes before transmission.

Time delay (in system ticks) that the WebSocket implementation waits to accumulate small writes before sending them as a single frame. This reduces protocol overhead by batching multiple small writes together.

define WS_ACCUM_DLY ((TICKS_PER_SECOND / 4) + 1)

Default Value:** (TICKS_PER_SECOND / 4) + 1 (approximately 250ms + 1 tick)

Purpose:**

- Reduces frame overhead by combining multiple small writes
- Improves network efficiency for applications with frequent small messages
- Balances latency against protocol overhead

Behavior:**

- Small writes are buffered for up to WS_ACCUM_DLY ticks
- If buffer fills or Flush() is called, data is sent immediately
- Larger writes may bypass accumulation delay

Tuning Considerations:**

- Increase for applications prioritizing throughput over latency
-

- Decrease for real-time applications requiring minimal latency
- Set to 0 to disable accumulation (send every write immediately)
- Call Flush() explicitly to bypass accumulation when needed

Note

Value is in system ticks (convert using TICKS_PER_SECOND)
Explicit Flush() calls override this delay

See also

Flush()

[TICKS_PER_SECOND](#)

[WS_FLUSH_TIMEOUT](#)

10.18.14.5 Timeout value for flush operations (in system ticks).

Specifies the maximum time to wait when flushing buffered WebSocket data. A value of 0 indicates non-blocking flush operations.

`define WS_FLUSH_TIMEOUT 0`

Default Value:** 0 (non-blocking)

Behavior:**

- 0: Flush returns immediately, even if not all data is sent
- Positive value: Flush waits up to specified ticks for data transmission
- Used internally by the Flush() method
- Non-blocking Flush (0):**
- Queues data for transmission but doesn't wait
- Suitable for most applications where blocking is undesirable
- Transmission occurs in background as TCP allows

Note

Value is in system ticks

Most applications should use the default non-blocking behavior

See also

Flush()

[WS_ACCUM_DLY](#)

10.18.14.6 WebSocket frame FIN bit mask indicating final fragment.

The FIN bit is set in the first byte of a WebSocket frame to indicate whether this is the final fragment of a message. According to RFC 6455, the FIN bit is the most significant bit (bit 7) of the first frame byte.

`define WS_FIN_BIT 0x80`

Bit Position:** 0x80 (bit 7 set)

Frame Fragmentation:**

- FIN = 1: This is the final (or only) fragment of the message
- FIN = 0: More fragments follow this one

Usage:**

- Set on single-frame messages (most common case)
- Set on the last frame of a fragmented message
- Clear on continuation frames (except the last one)

Message Fragmentation Example:**

```
// Large message sent in 3 fragments:
// Frame 1: opcode=TEXT, FIN=0 (first fragment)
// Frame 2: opcode=CONT, FIN=0 (middle fragment)
// Frame 3: opcode=CONT, FIN=1 (final fragment)
```

Note

This is typically handled automatically by the WebSocket implementation
Fragmentation allows sending large messages without buffering entirely

See also

[WS_OP_CONT](#)

[WS_OP_TEXT](#)

[WS_OP_BIN](#)

10.18.14.7 WebSocket continuation frame opcode.

Opcode value for continuation frames, which carry subsequent fragments of a message that was started with a TEXT or BINARY frame. Continuation frames allow large messages to be sent in multiple fragments.

define WS_OP_CONT 0x0

Opcode Value:** 0x0

Frame Sequence:**

1. First frame: TEXT or BINARY opcode with FIN=0
2. Middle frames: CONT opcode with FIN=0
3. Final frame: CONT opcode with FIN=1

Rules:**

- Must follow a TEXT or BINARY frame with FIN=0
- Cannot start a new message (must be part of fragmented message)
- All fragments must be same type (all text or all binary)
- Control frames (PING, PONG, CLOSE) cannot be fragmented

Note

The WebSocket implementation typically handles fragmentation automatically
Interleaving control frames between fragments is allowed by RFC 6455

See also

[WS_OP_TEXT](#)

[WS_OP_BIN](#)

[WS_FIN_BIT](#)

10.18.14.8 WebSocket text frame opcode for UTF-8 encoded text data.

Opcode value for frames containing UTF-8 encoded text data. Text frames are used for sending human-readable text messages and JSON data.

```
#define WS_OP_TEXT 0x1
```

Opcode Value:** 0x1

Characteristics:**

- Payload must be valid UTF-8 encoded text
 - Most common frame type for web applications
 - Used for JSON, XML, plain text, and other text protocols
 - Invalid UTF-8 results in protocol error and connection closure
- vs Binary Frames:**
- TEXT: Human-readable, UTF-8 validated, JSON-friendly
 - BINARY: Raw bytes, no encoding constraints, more efficient for non-text

Note

Payload validation ensures UTF-8 compliance (per RFC 6455)

For non-text data, use WS_OP_BIN for better performance

See also

[WS_OP_BIN](#)

[WS_OP_CONT](#)

[WS_SO_TEXT](#)

Expand for Example Usage**Examples**

Example: Sending Text vs Binary Data

```
int wsFd = ...; // WebSocket connection

// Send text data (automatically uses WS_OP_TEXT)
const char *jsonMsg = "{\"temp\":25.5,\"humidity\":60}";
WSWrite(wsFd, jsonMsg, strlen(jsonMsg));

// For binary data, use WS_OP_BIN (implicit)
// Set socket option if needed:
ws_cloption(wsFd, WS_SO_TEXT); // Ensure binary mode

uint8_t binaryData[100];
// ... fill with binary data ...
WSWrite(wsFd, (char*)binaryData, sizeof(binaryData));
```

10.18.14.9 WebSocket binary frame opcode for raw binary data.

Opcode value for frames containing arbitrary binary data. Binary frames are more efficient than text frames for non-textual data as they bypass UTF-8 validation and encoding overhead.

```
#define WS_OP_BIN 0x2
```

Opcode Value:** 0x2

Characteristics:**

- Payload can contain any byte sequence
- No UTF-8 validation performed
- More efficient for images, files, compressed data, etc.
- Commonly used for protocol buffers, MessagePack, CBOR

Use Cases:**

- Binary protocols (Protocol Buffers, MessagePack, BSON)
- Image and media data
- Compressed data
- Encrypted payloads
- Raw sensor data
- Any non-text binary format
- vs Text Frames:**
- BINARY: No encoding overhead, any byte values, faster
- TEXT: UTF-8 validated, human-readable, JSON-compatible

Note

More efficient than Base64-encoding binary data in text frames

No content validation means any byte sequence is valid

See also

[WS_OP_TEXT](#)

[WS_OP_CONT](#)

Expand for Example Usage

Examples

Example 1: Sending Binary Sensor Data

```
struct SensorReading {
    uint32_t timestamp;
    float temperature;
    float humidity;
    uint16_t pressure;
} __attribute__((packed));

void SendSensorData(int wsFd) {
    SensorReading reading;
    reading.timestamp = Secs;
    reading.temperature = 25.5f;
    reading.humidity = 60.0f;
    reading.pressure = 1013;

    // Send as binary frame (more efficient than JSON)
    ws_cloption(wsFd, WS_SO_TEXT); // Ensure binary mode
    WSWrite(wsFd, (char*)&reading, sizeof(reading));
}
```

Example 2: Binary vs Text Efficiency

```
// JSON text approach (less efficient):
// {"timestamp":12345,"temp":25.5,"humidity":60,"pressure":1013}
// = ~65 bytes as text

// Binary approach (more efficient):
// struct: 4 + 4 + 4 + 2 = 14 bytes
// Savings: ~78% size reduction

void CompareTextVsBinary(int wsFd) {
    // Text frame approach
    ws_setoption(wsFd, WS_SO_TEXT);
    char jsonBuf[100];
    int jsonLen = snprintf(jsonBuf, sizeof(jsonBuf),
        "{ \"ts\":%lu, \"t\":%.1f, \"h\":%.1f, \"p\":%d }",
        Secs, 25.5f, 60.0f, 1013);
    WSWrite(wsFd, jsonBuf, jsonLen);
    iprintf("Text frame: %d bytes\r\n", jsonLen);

    // Binary frame approach
    ws_cloption(wsFd, WS_SO_TEXT);
    SensorReading reading = {Secs, 25.5f, 60.0f, 1013};
    WSWrite(wsFd, (char*)&reading, sizeof(reading));
    iprintf("Binary frame: %d bytes\r\n", sizeof(reading));
}
```

Example 3: Sending Image Data

```

void SendImageFrame(int wsFd, uint8_t *imageData, uint32_t imageSize) {
    // Ensure binary mode for image data
    ws_clooption(wsFd, WS_SO_TEXT);

    // Check buffer space
    if (GetWriteSpace() < imageSize) {
        iprintf("Insufficient buffer space\r\n");
        return;
    }

    // Send image as binary frame
    int sent = WSWrite(wsFd, (char*)imageData, imageSize);
    if (sent == imageSize) {
        iprintf("Sent %u byte image\r\n", imageSize);
    } else {
        iprintf("Image send failed\r\n");
    }
}

```

10.18.14.10 WebSocket close frame opcode for connection termination.

Opcode value for close frames, which initiate graceful connection closure. Close frames may include a status code and optional reason string explaining why the connection is being closed.

define WS_OP_CLOSE 0x8

Opcode Value:** 0x8

Closure Handshake:**

1. One endpoint sends CLOSE frame with optional status code
2. Peer responds with CLOSE frame
3. Both endpoints close the underlying TCP connection

Close Frame Format:**

- Bytes 0-1: Status code (optional, see WS_StatusCode enum)
- Bytes 2+: Reason string in UTF-8 (optional)

Common Status Codes:**

- 1000: Normal closure (WS_STAT_NORM_CLOSE)
- 1001: Going away (WS_STAT_GOING_AWAY)
- 1002: Protocol error (WS_STAT_PROT_ERROR)
- 1009: Message too big (WS_STAT_MSG_TOO_BIG)
- 1011: Unexpected condition (WS_STAT_UNEXPECTED_COND)

Note

Close frames are automatically handled by the WebSocket implementation

After sending/receiving CLOSE, no more data frames should be sent

Always close gracefully rather than abruptly terminating TCP connection

See also

WS_STAT_NORM_CLOSE

WS_STAT_GOING_AWAY

10.18.14.11 WebSocket ping frame opcode for connection liveness check.

Opcode value for ping frames, which test connection liveness and measure round-trip time. The recipient must respond with a pong frame containing the same payload.

define WS_OP_PING 0x9

Opcode Value:** 0x9

Ping/Pong Protocol:**

- Sender transmits PING frame with optional payload (up to 125 bytes)
- Recipient must respond with PONG frame echoing the same payload
- Used to verify connection is alive and measure latency

Use Cases:**

- Keep-alive mechanism to prevent idle timeouts
- Connection health monitoring
- Round-trip time measurement
- Network quality assessment

- Dead connection detection

Implementation Notes:**

- Ping frames are control frames and cannot be fragmented
- Payload should be kept small (RFC 6455 recommends ≤ 125 bytes)
- Multiple pings can be outstanding simultaneously
- Pongs should be sent in response order

Note

Use [WSPing\(\)](#) function rather than manually constructing frames

Automatic pong responses are handled by the implementation

See also

[WS_OP_PONG](#)

[WSPing\(\)](#)

[WSWaitForPingReply\(\)](#)

[WSGetPingReplyTick\(\)](#)

Expand for Example Usage

Examples

Example: Connection Health Monitoring

```
void MonitorWebSocketHealth(int wsFd) {
    uint32_t pingSent, pongReceived;

    // Send ping frame (handled by WSPing, not raw WS_OP_PING)
    if (WSPing(wsFd, 0, &pingSent) >= 0) {
        // Wait for pong response
        if (WSWaitForPingReply(wsFd, TICKS_PER_SECOND * 5) >= 0) {
            if (WSGetPingReplyTick(wsFd, &pongReceived) >= 0) {
                uint32_t rtt = (pongReceived - pingSent) / (TICKS_PER_SECOND / 1000);
                fprintf("Connection alive, RTT: %lu ms\r\n", rtt);
            }
        } else {
            fprintf("Ping timeout - connection may be dead\r\n");
        }
    }
}
```

10.18.14.12 WebSocket pong frame opcode for ping response.

Opcode value for pong frames, which are sent in response to ping frames. Pong frames must echo back the exact payload received in the corresponding ping frame.

```
#define WS_OP_PONG 0xA
```

Opcode Value:** 0xA

Ping/Pong Protocol:**

- Automatically sent by implementation when PING is received
- Must contain the same payload as the received PING
- Can also be sent unsolicited as a unidirectional heartbeat

Response Rules:**

- Pong payload must exactly match the ping payload
 - Should be sent as soon as practical after receiving ping
 - Multiple pongs may be needed if multiple pings are received
 - Pongs should be sent in the order pings were received
- Unsolicited Pongs:**
- Can be sent without a corresponding ping (RFC 6455 allows this)
 - Useful as unidirectional heartbeat mechanism
 - Receiving endpoint may ignore unsolicited pongs

Note

Pong responses are typically handled automatically by the WebSocket stack

Applications rarely need to manually send pong frames

Use WSPong() if manual pong transmission is needed

See also

[WS_OP_PING](#)

[WSPing\(\)](#)

[WSWaitForPingReply\(\)](#)

10.18.14.13 WebSocket socket option to enable text frame mode.

Socket option flag to configure a WebSocket connection to send data as text frames (WS_OP_TEXT) rather than binary frames (WS_OP_BIN). This affects the opcode used for outgoing data frames.

```
#define WS_SO_TEXT 0x8
```

Option Value:** 0x8

Behavior:**

- When set: Outgoing frames use WS_OP_TEXT opcode
- When clear: Outgoing frames use WS_OP_BIN opcode

Text Frame Requirements:**

- Payload must be valid UTF-8 encoded text
- Implementation may validate UTF-8 encoding
- Use for JSON, XML, HTML, or plain text data

Functions:**

- `ws_setopt(fd, WS_SO_TEXT)`: Enable text mode
- `ws_clroption(fd, WS_SO_TEXT)`: Enable binary mode
- `ws_getoption(fd)`: Check current option state

When to Use Text Mode:**

- Sending JSON, XML, or other text-based protocols
- Human-readable messages
- When client/server expects text frames
- For maximum compatibility with web browsers

When to Use Binary Mode:**

- Raw sensor data or binary protocols
- Image, audio, video data
- Compressed or encrypted payloads
- Maximum efficiency (no UTF-8 overhead)

Note

Default mode may vary; set explicitly for predictable behavior

Mode can be changed dynamically during connection lifetime

See also

[WS_OP_TEXT](#)

[WS_OP_BIN](#)

`ws_setopt()`

`ws_clroption()`

`ws_getoption()`

Expand for Example Usage

Examples

Example 1: Setting Text Mode for JSON

```
void SendJSONMessage(int wsFd, const char *jsonStr) {
    // Enable text mode for JSON data
    ws_setopt(wsFd, WS_SO_TEXT);

    // Send JSON string (will use WS_OP_TEXT)
    int sent = WSWrite(wsFd, jsonStr, strlen(jsonStr));
    if (sent > 0) {
        iprintf("Sent JSON: %s\r\n", jsonStr);
    }
}
```

Example 2: Switching Between Text and Binary

```
void SendMixedData(int wsFd) {
    // Send text message
    ws_setopt(wsFd, WS_SO_TEXT);
    const char *textMsg = "{\"type\":\"status\",\"value\":\"ready\"}";
    WSWrite(wsFd, textMsg, strlen(textMsg));

    // Switch to binary mode for sensor data
    ws_clroption(wsFd, WS_SO_TEXT);
    struct {
        uint32_t timestamp;
        float temperature;
    } sensorData = {Secs, 25.5f};
    WSWrite(wsFd, (char*)&sensorData, sizeof(sensorData));

    // Back to text mode for acknowledgment
    ws_setopt(wsFd, WS_SO_TEXT);
    const char *ack = "{\"type\":\"ack\"}";
    WSWrite(wsFd, ack, strlen(ack));
}
```

Example 3: Checking Current Mode

```
void DisplayWebSocketMode(int wsFd) {
    int options = ws_getoption(wsFd);

    if (options & WS_SO_TEXT) {
        iprintf("WebSocket fd %d: TEXT mode (UTF-8)\r\n", wsFd);
    } else {
        iprintf("WebSocket fd %d: BINARY mode\r\n", wsFd);
    }
}
```

Example 4: Application-Specific Protocol

```
// Define message types
enum MessageType {
    MSG_TEXT_JSON,
    MSG_BINARY_PROTOBUF,
    MSG_BINARY_IMAGE
};

void SendMessage(int wsFd, MessageType type, const void *data, int len) {
    switch (type) {
        case MSG_TEXT_JSON:
            // Use text frames for JSON
            ws_setoption(wsFd, WS_SO_TEXT);
            iprintf("Sending as TEXT frame\r\n");
            break;

        case MSG_BINARY_PROTOBUF:
        case MSG_BINARY_IMAGE:
            // Use binary frames for protobuf and images
            ws_cloption(wsFd, WS_SO_TEXT);
            iprintf("Sending as BINARY frame\r\n");
            break;
    }

    int sent = WSWrite(wsFd, (const char*)data, len);
    iprintf("Sent %d bytes\r\n", sent);
}
```

Example 5: WebSocket Mode Configuration Helper

```
// Helper function to configure WebSocket for JSON API
void ConfigureJSONWebSocket(int wsFd) {
    // Set text mode
    ws_setoption(wsFd, WS_SO_TEXT);

    // Verify setting
    int opts = ws_getoption(wsFd);
    if (opts & WS_SO_TEXT) {
        iprintf("WebSocket configured for JSON (text mode)\r\n");
    } else {
        iprintf("ERROR: Failed to set text mode\r\n");
    }
}

// Helper function to configure WebSocket for binary protocol
void ConfigureBinaryWebSocket(int wsFd) {
    // Clear text mode (enable binary)
    ws_cloption(wsFd, WS_SO_TEXT);

    // Verify setting
    int opts = ws_getoption(wsFd);
    if (!(opts & WS_SO_TEXT)) {
        iprintf("WebSocket configured for binary protocol\r\n");
    } else {
        iprintf("ERROR: Failed to clear text mode\r\n");
    }
}
```

Chapter 11

System Diagnostics

11.1 Introduction

System Diagnostics can be viewed from a device's Configuration Server web page (port 20034) by calling the [EnableSystemDiagnostics\(\)](#) function in your application. Diagnostic information includes items such as system definitions, variables, TCP stack status, and RTOS state information.

11.2 Server Page with Diagnostics Enabled

When system diagnostics are enabled, an additional button will appear on the configuration server web page in the lower right corner:

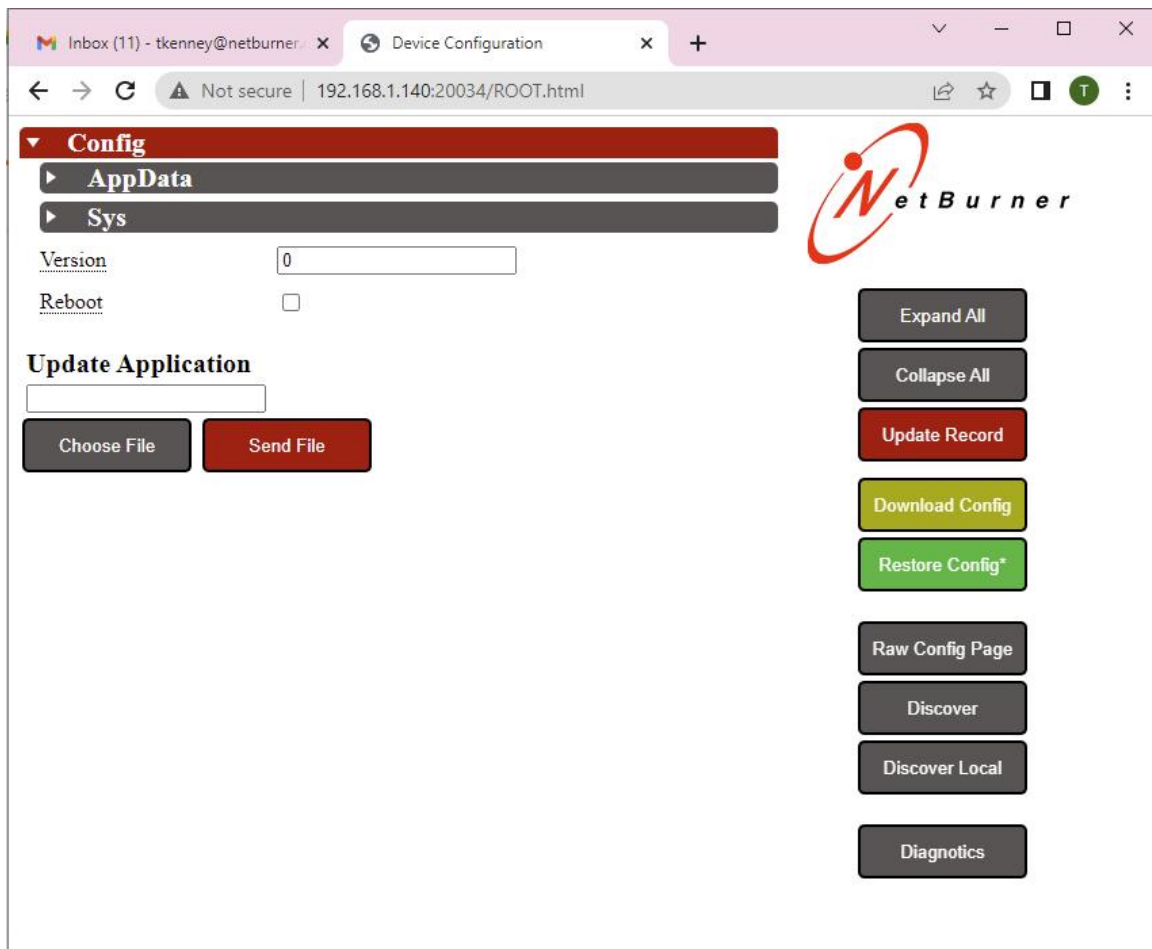


Figure 11.1 Configuration Server With Diagnostics Button

11.3 Configuration Web Page Display

Clicking on the Diagnostics button will display system information. An example of the raw data is shown below:

```
{
  "Active":0 day 01:18:04,
  "Buffers":401,
  "Config":{"ConfigUsed":749,
  "ConfigMaxSpaceAvailable":10240
}
,
"EXtraIO":120,
"FreeMem":7577316,
"My Int":4677,
"My String":"My String",
"NNDKVer":"3.3.8",
"System Macros":{"
  "_DEBUG":"not defined",
  "TICKS_PER_SECOND":20,
  "TICK_IRQ_LEVEL":5,
  "SERIAL_IRQ_LEVEL":3,
  "TCP_SOCKET_OFFSET":5,
  "TCP_SOCKET_STRUCTS":128,
  "OS_MAX_IRQ":"not defined",
  "NBRTOS_STACKCHECK":"not defined",
  "NBRTOS_STACKOVERFLOW":"not defined",
  "NBRTOS_TASKLIST":"not defined",
  "NBRTOS_TASK_LOG":"not defined",
  "NBRTOS_TIME":"not defined",
  "BUFFER_DIAG":"not defined",
  "BUFFER_DIAG_LOG":"not defined",
```

```

"BUFFER_SANITY":"not defined",
"_DEBUG_PRINT":"not defined",
"ENABLE_SMARTTRAP":"defined",
"MULTIHOME":"not defined",
"IPV6":"defined",
"IPV6_COUNTERS":"defined",
"AUTOIP":"defined",
"ALLOW_CUSTOM_NET_DO_RX":"not defined",
"TCP_NOCOPY_TX":"defined",
"UDP_FRAGMENTS":"not defined",
"GATHER_RANDOM":"defined",
"NB_SSL_SUPPORTED":"defined",
"SSL_V3_DISABLED":"not defined",
"SSL_TLS_SUPPORT":"defined",
"WEB_CLIENT_SSL_SUPPORT":"defined",
"SSL_DEFAULT_MAX_SESSION_AGE_TICKS":"defined",
"TLS_CACHE_PEER_CERT_VALIDATIONS":"defined",
"NB_SSL_CLIENT_CERTIFICATE_CHECKING_ENABLED":"not defined",
"NB_SSH_SUPPORTED":"not defined",
"NB_ENABLE_USER_QSPI":"defined",
"SUPPORT_LEGACY_FIND":"defined",
"NO_SYMMETRIC_ROUTING":"not defined",
"FEC_ISR_ERROR_COUNTERS":"not defined"
},
"TCP":{"FreeSockets":124,
"Sockets":{"Socket129":{"State":"ESTABLISHED","Flags":"0x81","myIP":"192.168.1.140","theirIP":"192.168.1.26","myPort":20034,"theirPort":20034,"myWindow":871968320,"TxBuffDepth":0,"RxBuffDepth":0,"lastRxTime":93681},
"Socket130":{"State":"ESTABLISHED","Flags":"0x80","myIP":"192.168.1.140","theirIP":"192.168.1.26","myPort":20034,"theirPort":20034,"myWindow":2744612803,"TxBuffDepth":0,"RxBuffDepth":0,"lastRxTime":93681},
"Socket131":{"State":"LISTEN","Flags":"0x00","myIP":"","theirIP":"","myPort":80,"theirPort":0,"max_listen":5,"cur_listen":0,"myWindow":0,"TxBuffDepth":0,"RxBuffDepth":0,"lastRxTime":0},
"Socket132":{"State":"LISTEN","Flags":"0x00","myIP":"","theirIP":"","myPort":20034,"theirPort":0,"max_listen":5,"cur_listen":0,"myWindow":0,"TxBuffDepth":0,"RxBuffDepth":0,"lastRxTime":93681}}}
"Tasks":{
"Task38":{"Name":"Enet", "Prio":38,"State":"Fifo","Time":40,
"Stack":["70005b26","7002d6d4","70005f30","7000ed1c","000004a4"]},
"Task44":{"Name":"Config Server", "Prio":44,"State":"Running","Time":600,
"Stack":["7002d54a","700061be","70023f82","700249ec","70020de4","7002178e","700218e8","7003143e","700213e6","000004a4"]},
"Task45":{"Name":"HTTP", "Prio":45,"State":"Semaphore","Time":19,
"Stack":["70005b26","7002d6d4","70005e48","7000465a","70004694","700245e4","000004a4"]},
"Task50":{"Name":"Main", "Prio":50,"State":"Timer","Time":20,
"Stack":["70005b26","7002d6d4","70005b96","7001e05a","000004a4"]},
"Task63":{"Name":"Idle", "Prio":63,"State":"Ready","Time":"FOREVER", "Stack":["7002c424","000004a4"]}
},
"Ticks":93681,
"UpTime":4684
}

```

If you are using Firefox or have the Pretty Print extension in Chrome, the JSON data will be formatted and much easier to read. If the output you see looks identical to the previous raw information, it means the browser you are using does not support JSON data formatting. The image below shows a snippet of a formatted file displayed in Firefox:

JSON	Raw Data	Headers
Save	Copy	Collapse All Expand All Filter JSON
Active:	"0 day 00:00:34"	
Buffers:	401	
▼ Config:		
ConfigUsed:	785	
ConfigMaxSpaceAvailable:	10240	
ExtraIO:	120	
FreeMem:	7567588	
My Int:	27	
NNDKVer:	"Release_3_X-git"	
▼ System Macros:		
_DEBUG:	"not defined"	
TICKS_PER_SECOND:	20	
TICK_IRQ_LEVEL:	5	
SERIAL_IRQ_LEVEL:	3	
TCP_SOCKET_OFFSET:	5	
TCP_SOCKET_STRUCTS:	128	
OS_MAX_IRQ:	"not defined"	
NBRTOS_STACKCHECK:	"not defined"	
NBRTOS_STACKOVERFLOW:	"not defined"	
NBRTOS_TASKLIST:	"not defined"	
NBRTOS_TASK_LOG:	"not defined"	
NBRTOS_TIME:	"not defined"	
BUFFER_DIAG:	"not defined"	
BUFFER_DIAG_LOG:	"not defined"	
BUFFER_SANITY:	"not defined"	
_DEBUG_PRINT:	"not defined"	
ENABLE_SMARTTRAP:	"defined"	
MULTIHOME:	"not defined"	
IPV6:	"defined"	
IPV6_COUNTERS:	"defined"	
AUTOIP:	"defined"	
ALLOW_CUSTOM_NET_DO_RX:	"not defined"	
TCP_NOCOPY_TX:	"defined"	
UDP_FRAGMENTS:	"not defined"	
GATHER_RANDOM:	"defined"	
NB_SSL_SUPPORTED:	"defined"	
SSL_V3_DISABLED:	"not defined"	
SSL_TLS_SUPPORT:	"defined"	
WEB_CLIENT_SSL_SUPPORT:	"defined"	
SSL_DEFAULT_MAX_SESSION_AGE_TICKS:	"defined"	
TLS_CACHE_PEER_CERT_VALIDATIONS:	"defined"	
NB_SSL_CLIENT_CERTIFICATE_CHECKING_ENABLED:	"not defined"	
NB_SSH_SUPPORTED:	"not defined"	
NB_ENABLE_USER_QSPI:	"defined"	
SUPPORT_LEGACY_FIND:	"defined"	
NO_SYMMETRIC_ROUTING:	"not defined"	
FEC_ISR_ERROR_COUNTERS:	"not defined"	
▼ TCP:		
FreeSockets:	125	

Figure 11.2 Formatted JSON displayed with Firefox

11.4 Adding Your Own Diagnostic Information

You can add your own diagnostic information using the `MyVarMon()` functions. Please refer to the System Diagnostics example for more details. In the example output in the previous section, the variables below were added to the standard system output:

```
"My Int":4677,  
"My String":"My String",
```

The application code to implement the display from the example is:

```
// Variables to monitor  
int myInt;  
char myString[80];  
  
// Declarations to add variables to Diagnostic Web page  
static DiagVarMon MyIntMon("My Int", myInt);  
static DiagVarMon MyStrMon("My String", myString);
```


Chapter 12

Example Applications

12.1 Example Applications Main Page

Included in your NNDK software installation folder are over 200 examples to aid you in your product development and learning the NetBurner platform, demonstrating functionality such as:

- Web pages, Web servers and Web clients
- TCP client and server
- TLS client and server, certificate, auto cert generation and management
- UDP client and server
- Device configuration
- Real-Time operating system such as semaphore, tasks, queues and critical sections
- JSON
- File system
- VLAN
- Web sockets
- Hardware board locking to secure application code
- Hardware peripherals such as [I2C](#), SPI, UART, USART, 1-Wire, GPIO, ADC, DAC, PWM, Timers
- PPP
- FTP
- SSH
- [MQTT](#)
- ACME certificate support

These are just some of the extensive topics covered. There are a few ways to browse the examples:

1. The HTML and PDF documents
2. Browsing the folder structure in `nburn\examples`
3. When using NBEclipse, you can browse and import examples

There are two main categories of examples:

1. The `\nburn\examples` folder contains examples that can be run on all platforms.

2. The `\nburn\examples\PlatformSpecific` folder contains examples that are specific to a NetBurner product, such as hardware peripherals: [I2C](#), SPI, 1-Wire, ADC, DAC, etc.

Links to the example sections are below, or you can use the tree control structure in the online docs.

- [AES](#)
- [Application Image Security](#)
- [Clear Configuration Flash](#)
- [Configuration](#)
- [DNS Device Name](#)
- [DHCP](#)
- [DNS Client](#)
- [Embedded Flash File System \(EFFS\)](#)
- [Ethernet](#)
- [Exception Try/Catch](#)
- [External IRQ](#)
- [Extra FD Circular Buffer](#)
- [FileDescriptor](#)
- [FTP](#)
- [GDB Debugger](#)
- [General Purpose I/O](#)
- [I2C](#)
- [IP Address Object \(IPADDR\)](#)
- [IPv6](#)
- [JSON](#)
- [malloc](#)
- [MQTT Examples](#)
- [Multicast](#)
- [Multihome](#)
- [NB Approve Shutdown](#)
- [NBString Class](#)
- [Network Time Server \(NTP\) Client](#)
- [NTP & Real-Time Clock](#)
- [Overload Directory & System Files](#)
- [Ping](#)
- [PLATFORM SPECIFIC](#)
- [PPP](#)
- [Profiler](#)

- [RTOS](#)
- [Serial](#)
- [Serial Webserver](#)
- [SHA1 Digest](#)
- [Show Network Interfaces](#)
- [SOCKS5 Client](#)
- [SPI](#)
- [SSH](#)
- [SSL/TLS](#)
- [Stack Protection](#)
- [Syslog](#)
- [System Diagnostics](#)
- [TCP](#)
- [Telnet Command](#)
- [TFTP Client](#)
- [TFTP Server](#)
- [Time Functions](#)
- [Timers](#)
- [UDP](#)
- [VLAN](#)
- [Web Server](#)
- [Web Client](#)
- [Web Sockets](#)
- [WiFi Examples](#)

12.2 AES

A basic description of AES can be obtained from Wikipedia, and if you are new to AES, you should definitely do some research to understand the implementation. The description below is from Wikipedia:

In cryptography, a block cipher is a symmetric key cipher which operates on fixed-length groups of bits, termed blocks, with an unvarying transformation. When encrypting, a block cipher might take a (for example) 16-bit block of plaintext as input, and output a corresponding 16-bit block of ciphertext. The exact transformation is controlled using a second input - the secret key. Decryption is similar: the decryption algorithm takes, in this example, a 16-bit block of ciphertext together with the secret key, and yields the original 16-bit block of plaintext.

To encrypt messages longer than the block size, a mode of operation is used. Block ciphers can be contrasted with stream ciphers; a stream cipher operates on individual digits one at a time, and the transformation varies during the encryption. The distinction between the two types is not always clear-cut: a block cipher, when used in certain modes of operation, acts effectively as a stream cipher.

This example demonstrates how to use the AES key, encryption, and decryption function calls on a 16 byte block of data. Since AES is a block cipher, all data must be encrypted in 16 byte blocks. If you have more than 16 bytes to transfer, you must break the data into 16 byte blocks, encrypt them (you can use the same key for all blocks), send them, then decrypt them in 16 byte blocks on the other end.

12.3 Application Image Security

12.3.1 Examples for application image security, including:

- Application image Checksum
- Locking an application to a specific board
- Signing an application with a private key so that a new application cannot be programmed without also being signed.

12.3.1.1 Examples

- [Application Image Checksum](#)
- [Board Lock](#)
- [Signed Application](#)

12.3.2 Application Image Checksum

12.3.2.1 Application Image Checksum

12.3.2.1.1 Overview

This application demonstrates how to read and display the flash application image checksum at runtime. The checksum provides a way to verify the integrity of the application image stored in flash memory.

12.3.2.1.2 Features

- Reads the application image checksum from the running application header
- Displays application build information including name and NNDK revision
- Starts an HTTP server for web-based access to checksum information
- Provides both console output and web interface variables

12.3.2.1.3 How It Works

The application uses the NetBurner RTOS system to access the running application image header structure. The checksum is retrieved from the `dwImgSum` field of the application header and displayed in hexadecimal format.

12.3.2.1.3.1 Key Components

- **main.cpp**: Main application code that initializes the system and retrieves the checksum
- **htmlvar.h**: Header file declaring web interface variables

12.3.2.1.3.2 Checksum Access

The application checksum can be accessed at runtime by including the `startup.h` header and reading from the `RunningAppImage` structure:

```
#include <startup.h>
printf("Application Checksum: %#08lx\r\n", RunningAppImage->appHdr.dwImgSum);
```

12.3.2.1.4 Build Process Verification

When building the application, the `flashpack` utility displays the image checksum in its output. This allows you to verify that the runtime checksum matches the build-time checksum.

Example flashpack output:

```
App Header:
Image Address: 0x0040801c
Exec Address: 0x70000001
pExtra: 0x0040801c
Image Length: 0x0007adb0
```

```
Image Checksum: 0xc585db80
Header Checksum: 0x0c46c8e5
Raw image size: 646688 bytes
Compressed size: 503106 bytes
```

The Image Checksum line shows the checksum value that should match the runtime value.

12.3.2.1.5 Application Output

The application provides the following information:

1. **Console Output:** Displays app name, NNDK revision, and checksum via printf statements
2. **Web Variables:** Makes checksum and app info available through HTML variables:
 - `strAppInfo`: Contains app name and NNDK revision
 - `strAppChecksum`: Contains the hexadecimal checksum value

12.3.2.1.6 System Requirements

- NetBurner RTOS system
- Network connectivity for HTTP server functionality
- Flash memory with valid application image

12.3.2.1.7 Usage

1. Build and flash the application to your NetBurner device
2. Connect to the device via serial console to see the output
3. Access the HTTP server to view checksum information via web interface
4. Compare the runtime checksum with the build-time checksum from flashpack output

12.3.2.1.8 Security Considerations

The image checksum provides a basic integrity check for the application image. It can help detect:

- Corruption of the flash memory
- Incomplete or failed firmware updates
- Tampering with the application image

For production systems, consider removing the `EnableSystemDiagnostics()` call as noted in the code comments.

12.3.2.1.9 Technical Details

- The application runs in an infinite loop with 1-second delays
- Network initialization waits up to 5 seconds for active network
- HTTP server is started automatically for web interface access
- Checksum is displayed as an 8-digit hexadecimal value with 0x prefix

12.3.3.2.1.2 Purpose

This application implements a hardware-based authorization system that:

- Prevents unauthorized devices from running the application
- Uses the network interface MAC address as a unique device identifier
- Stores authorization data in the device's UserParam flash space
- Provides runtime verification of device authorization

12.3.3.2.1.3 How It Works

Board Lock Verification Process

1. **Initialize Context:** The application uses a pre-configured MD5 context with specific state values that serve as a company secret
2. **Get MAC Address:** Retrieves the MAC address from the first network interface
3. **Generate Hash:** Updates the MD5 context with the 6-byte MAC address and finalizes the hash
4. **Compare Values:** Compares the generated 16-byte digest with the stored authorization value in flash memory
5. **Return Result:** Returns true if all 16 bytes match, false otherwise

Key Components

YourCompanySecret

A pre-initialized MD5 context containing:

- **ctx_state:** 4x 32-bit state values (2106921824, 3945495657, 2391356351, 2780313164)
- **ctx_count:** 2x 32-bit count values (512, 0)
- **buf:** 64-byte buffer for MD5 operations

CheckBoardLock Function

The core authorization function that:

- Copies the company secret MD5 context
- Retrieves the device MAC address
- Updates MD5 hash with MAC address
- Compares result with stored authorization data

12.3.3.2.1.4 Application Flow

1. **Initialization:** Network stack initialization and system diagnostics enablement
2. **Network Wait:** Waits up to 5 seconds for active network ([DHCP](#))
3. **Authorization Check:** Calls `CheckBoardLock()` to verify device authorization
4. **Status Report:** Prints success or failure message
5. **Main Loop:** Enters infinite loop with 1-second delays

12.3.3.2.1.5 Security Features

- **Unique Device Binding:** Each authorization is tied to a specific MAC address
- **Flash Storage:** Authorization data stored in non-volatile UserParam flash space
- **Company Secret:** Uses proprietary MD5 context as cryptographic seed
- **Runtime Verification:** Checks authorization every time the application starts

12.3.3.2.1.6 Output Messages

- `"Application started"` - Application initialization complete
- `"Board checks ok"` - Device is authorized to run
- `"Board fails lock check"` - Device authorization failed

12.3.3.2.1.7 Technical Requirements

- Network interface support for MAC address retrieval
- MD5 hashing capability
- UserParam flash space for storing authorization data
- Real-time operating system (RTOS) support

12.3.3.2.1.8 Security Considerations

- The application includes system diagnostics which should be removed for production builds
- The company secret values are hardcoded and should be unique per deployment
- Authorization data in flash must be properly programmed during device provisioning

12.3.3.2.1.9 Usage

This application is typically used in embedded systems where device authorization is required before allowing normal operation. The board lock mechanism ensures that only properly authorized hardware can run the application software.

12.3.3.2.1.10 Development Notes

- Remove `EnableSystemDiagnostics()` call for production builds

12.3.3.3 Key Blob

12.3.3.3.1 Key Blob Application

12.3.3.3.1.1 Overview

The Key Blob application is a device-locking security utility that generates a unique MD5-based key blob to bind an application to a specific device. This creates a hardware-specific signature that can be used for software licensing and device authentication purposes.

12.3.3.3.1.2 Purpose

This application creates an MD5 context that can be used to lock an application to a particular device. The generated key blob serves as a unique identifier that ties software to specific hardware, providing a form of copy protection and device authentication.

12.3.3.3.1.3 How It Works

1. **Initialization:** The application initializes the network stack and waits for an active network connection
2. **MD5 Context Creation:** Creates an MD5 context using a secret signing text
3. **Data Processing:** Processes the secret text in 64-byte chunks through the MD5 algorithm
4. **Padding:** When the secret text is shorter than 64 bytes, it pads the remaining space with incremental values
5. **Key Blob Generation:** Outputs the MD5 context state as C code that can be compiled into other applications

12.3.3.3.1.4 Key Components

Secret Signing Text

```
const char *YourSecretSigningText = "This should be your company secret message";
```

This should be replaced with your company's unique secret message for generating device-specific keys.

MD5 Processing

- Processes the secret text in 64-byte blocks
- Pads remaining space with incremental byte values (0, 1, 2, ...)
- Updates the MD5 context with the processed data

Output Format

The application generates C code output containing:

- `ctx_state[4]`: MD5 context state array
- `ctx_count[2]`: MD5 context count array
- Buffer declarations for SSL/TLS support (conditional)
- Complete MD5_CTX constructor call

12.3.3.3.1.5 Usage

1. **Customize Secret Text:** Replace `YourSecretSigningText` with your company's secret message
2. **Compile and Run:** Build the application for your target device
3. **Capture Output:** The application will print the key blob as C code to the console
4. **Integration:** Copy the generated code into your main application for device locking

12.3.3.3.1.6 Example Output

```
/*Your keyblob is:*/
uint32_t ctx_state[4]={1234567890u,987654321u,555666777u,111222333u};
uint32_t ctx_count[2]={128u,0u};
unsigned char buf[64];
#ifdef SSL_TLS_SUPPORT
unsigned char inbuf[64];
unsigned char outbuf[64];
#endif
MD5_CTX YourCompanySecret (ctx_state,ctx_count,buf
#ifdef SSL_TLS_SUPPORT
,inbuf,outbuf
#endif
);
```

12.3.3.3.1.7 Security Considerations

- The secret signing text should be kept confidential
- Each device will generate a unique key blob due to hardware differences
- The generated key blob should be integrated into your application's authentication system
- Consider additional security measures for production deployments

12.3.3.3.1.8 Dependencies

- Network stack initialization
- MD5 hashing library
- System diagnostics (optional, recommended to disable in production)

12.3.3.3.1.9 Network Requirements

- The application waits for [DHCP](#) address assignment
- Network connectivity is required during initialization
- Timeout is set to 5 seconds for network activation

12.3.3.3.1.10 Production Notes

- Consider disabling `EnableSystemDiagnostics()` for production builds
- Ensure the secret signing text is properly secured
- Test the key blob integration thoroughly before deployment
- The application runs in an infinite loop after generating the key blob

12.3.3.3.1.11 Application Flow

1. Initialize network stack
2. Enable system diagnostics (development only)
3. Wait for active network connection
4. Initialize MD5 context
5. Process secret signing text through MD5
6. Generate and output key blob as C code
7. Enter infinite loop

This key blob system provides a foundation for device-specific software licensing and authentication mechanisms.

12.3.3.4 Sign Board

12.3.3.4.1 Sign Board Lock Application

12.3.3.4.1.1 Overview

The Sign Board Lock application is a C++ embedded system program designed to create a unique digital signature that locks an application to a specific hardware device (board). This signature is generated using the device's MAC address and a company-specific secret key, providing a hardware-based authentication mechanism.

12.3.3.4.1.2 Purpose

This application serves as a device binding security measure by:

- Creating a unique signature based on the device's MAC address
- Using MD5 hashing with a pre-configured secret key
- Storing the resulting signature in the device's flash memory
- Preventing the application from running on unauthorized hardware

12.3.3.4.1.3 How It Works

Core Components

1. **Secret Key Configuration:** The application uses a predefined MD5 context (`YourCompanySecret`) containing:
 - State values: {2106921824u, 3945495657u, 2391356351u, 2780313164u}
 - Count values: {512u, 0u}
 - Buffer arrays for processing
2. **MAC Address Retrieval:** The program obtains the MAC address from the first available network interface on the device.
3. **Signature Generation:**
 - Combines the secret key context with the device's MAC address
 - Processes this combination through MD5 hashing
 - Produces a 16-byte unique digest
4. **Storage:** The generated signature is stored in the device's UserParam flash memory space.

Program Flow

1. **Initialization:** Network stack and system diagnostics are initialized
2. **Network Setup:** Waits up to 5 seconds for an active network connection ([DHCP](#))
3. **Board Signing:** Executes the `SignBoardLock()` function to generate and store the signature
4. **Continuous Operation:** Enters an infinite loop with 1-second delays

12.3.3.4.1.4 Key Functions

`SignBoardLock()`

- Creates a copy of the company secret MD5 context
- Retrieves the MAC address from the first network interface
- Updates the MD5 context with the MAC address data
- Finalizes the hash to produce a 16-byte digest
- Saves the digest to flash memory using `SaveUserParameters()`

`UserMain()`

- Main application entry point
- Handles system initialization and network setup
- Calls the board signing function
- Maintains the application in a running state

12.3.3.4.1.5 Technical Details

- **Hash Algorithm:** MD5
- **Signature Length:** 16 bytes
- **MAC Address Length:** 6 bytes

12.3.3.4.1.6 Security Considerations

- The secret key (`YourCompanySecret`) should be unique to your company/application
- The secret key values in the code should be replaced with your own company-specific values

12.3.3.4.1.7 Build Requirements

- Network interface support
- MD5 hashing library
- Flash memory access functions (`SaveUserParameters`)
- RTOS with timing functions (`OSTimeDly`, `TICKS_PER_SECOND`)

12.3.3.4.1.8 Usage Notes

- The application automatically signs the board upon startup
- No user interaction is required during the signing process
- The signature is permanently stored in flash memory
- System diagnostics are enabled by default (should be disabled for production)

12.3.3.4.1.9 Customization

To adapt this application for your use:

1. Replace the `ctx_state` and `ctx_count` values with your company-specific secret
2. Disable system diagnostics for production deployment

12.3.3.4.1.10 Output

The application provides console output indicating:

- "Application started" - when initialization is complete
- "Board signed" - when the signature has been successfully generated and stored

12.3.4 Signed Application

12.3.4.1 Signed Application Example

12.3.4.1.1 Overview

This is a NetBurner embedded application that demonstrates the signed application update capability. The application creates a web server that can receive and validate signed firmware updates using RSA public/private key cryptography.

12.3.4.1.2 Key Features

- **Signed Application Updates:** Validates firmware updates using RSA 1024-bit key pairs
- **Web Server:** Runs an HTTP server on port 80 with a simple web interface
- **Network Stack:** Full TCP/IP networking capabilities
- **Security:** Public key verification to ensure only authorized updates can be installed

12.3.4.1.3 How It Works

The application initializes the network stack, starts an HTTP web server, and registers a public key for validating signed application updates. After 20 seconds of runtime, it registers the signing public key with the system to enable secure firmware updates.

12.3.4.1.3.1 Main Application Flow

1. Initialize the network stack
2. Start HTTP web server on port 80
3. Wait for active network connection (5 second timeout)
4. Display system information
5. After 20 seconds, register the application signing public key
6. Continue running with 1-second heartbeat messages

12.3.4.1.4 Security Model

CRITICAL SECURITY NOTE:** The private key (`signkey.pem`) must NEVER be stored on the target device. It should be kept in a secure location separate from the device. Loss of the private key means you will never be able to update the device firmware again.

12.3.4.1.4.1 Key Requirements

- **Public Key:** Stored in the application binary and used to verify update signatures
- **Private Key:** Kept secure and separate from the device, used only for signing updates
- **Key Isolation:** Signing keys must be different from any other certificates/keys used on the device (e.g., HTTPS certificates)

12.3.4.1.5 Setup Instructions

12.3.4.1.5.1 1. Generate Key Pair

You can use the provided `makekey.bat` script or run these commands manually:

```
openssl genrsa -out signkey.pem 1024
openssl rsa -in signkey.pem -pubout -out src/public.key
set NB_SIGN_KEY=%cd%\signkey.pem
```

12.3.4.1.5.2 2. Command Line Build

For command line compilation and deployment:

```
make sign      # Create signed application
make loadsign  # Load signed application to device
```

12.3.4.1.5.3 3. NBEclipse IDE Setup

If using NetBurner's NBEclipse IDE:

1. Generate the key pair as described above
2. In the directory with your public key, run:
`compile public.key codekey_array codekey_len codekey.cpp`
3. Include the generated `codekey.cpp` file in your NBEclipse project
4. Build your project normally

External Tool Configuration for Signing

After building your project in NBEclipse:

1. Go to **Run > External Tools > External Tools Configurations**
2. Create a new configuration with these settings:
 - **Name:** Sign
 - **Location:** `${env_var:NNDK_ROOT}\pcbin\nbsign.exe`
 - **Working Directory:** Browse to your project workspace
 - **Arguments:** `-k <full_path_to_private_key.pem> -in Release\${project_name}.bin -o Release\${project_name}.signed.bin`
3. Run the "Sign" external tool to create signed binaries

12.3.4.1.6 File Structure

- `main.cpp` - Main application source code
- `makekey.bat` - Script to generate RSA key pair
- `signkey.pem` - RSA private key (1024-bit) - **KEEP SECURE**
- `public.key` - RSA public key for signature verification
- `index.html` - Simple web page served by the HTTP server
- `htmlvar.h` - Header file for HTML variables
- `ReadMe.txt` - Original documentation

12.3.4.1.7 Web Interface

The application serves a simple web page at the device's IP address that displays:

- NetBurner logo
- Device IP address
- "Thank you for NetBurning!" message

12.3.4.1.8 Runtime Output

The application provides console output showing:

- Application name and NNDK revision
- Network status
- 1-second heartbeat with timestamp
- Confirmation when signing key is registered

12.3.4.1.9 Important Security Considerations

1. **Private Key Security:** Never store the private key on the target device
2. **Key Backup:** Securely backup your private key - loss means permanent inability to update
3. **Key Uniqueness:** Use different keys for signing than for other device functions
4. **Secure Key Storage:** Store private keys in a hardware security module or other secure location
5. **Update Validation:** All firmware updates must be properly signed with the corresponding private key

12.3.4.1.10 Dependencies

- NetBurner NNDK (NetBurner Network Development Kit)
- OpenSSL (for key generation)
- Standard NetBurner libraries: init, nbrtos, system, [config_obj](#), nbupdate, iosys

12.3.4.1.11 Compilation Requirements

- NetBurner toolchain
- Access to NetBurner system libraries
- Properly generated public key compiled into the application

This example demonstrates the foundation for a secure firmware update system suitable for production IoT and embedded applications where remote update capability with cryptographic verification is essential.

12.4 Clear Configuration Flash

12.4.1 NetBurner Clear Configuration Flash

12.4.1.1 Overview

This is a simple utility program designed to clear the configuration flash area on NetBurner 3.x devices. The application provides a safe way to reset device configuration parameters to their default values by erasing the configuration flash memory.

12.4.1.2 Purpose

The primary purpose of this application is to:

- Clear the NetBurner 3.x configuration flash area
- Force re-initialization of configuration parameters after device reset
- Provide a clean slate for device configuration

12.4.1.3 Important Limitations

Note

For MOD5441x, NANO54415, and SB800EX devices, This program will NOT clear the configuration parameters used by the Alternate Boot Monitor.

12.4.1.4 Reboot Requirement

Note

After running this program successfully, you must reboot the NetBurner device for the configuration changes to take effect. The cleared flash will cause the device to reinitialize with default configuration parameters upon restart.

12.4.1.5 How It Works

1. **Initialization:** The program initializes the NetBurner system and enables system diagnostics
2. **Network Setup:** Configures discovery server and waits for active network connection (up to 5 seconds)
3. **User Confirmation:** Prompts the user to press any key before proceeding with the flash erase operation
4. **Flash Erase:** Calls `EraseWholeConfigRecord()` to clear the configuration flash area
5. **Completion:** Notifies the user that changes will take effect after reboot
6. **Idle Loop:** Enters an infinite loop, waiting for user to reboot the device

12.4.1.6 Usage Instructions

1. Flash this program to your NetBurner device
2. Run the program
3. When prompted, press any key to confirm the configuration flash erase
4. Wait for the "complete" message
5. Reboot the device to activate the cleared configuration

12.4.1.7 Technical Details

12.4.1.7.1 Dependencies

- NetBurner SDK libraries (`init.h`, `netinterface.h`)
- External function: `EraseWholeConfigRecord()`

12.4.1.7.2 Key Features

- System diagnostics enabled for debugging
- Network discovery server configuration
- User confirmation before destructive operation
- Clear feedback on operation status

12.4.1.7.3 Program Flow

```
- Initialize System
- Setup Network (5 second timeout)
- Wait for User Input
- Erase Configuration Flash
- Display Completion Message
- Wait for Reboot (Infinite Loop)
```

12.4.1.8 Safety Considerations

- The program requires user confirmation before erasing flash
- Configuration changes only take effect after a device reboot
- System diagnostics are enabled (should be disabled for production use)
- The program includes appropriate wait states and feedback messages

12.4.1.9 Development Notes

- Contains system diagnostics that should be removed for production code
- Uses standard NetBurner SDK functions and conventions
- Implements proper error handling and user feedback

12.5 Configuration

Examples to demonstrate how to:

- Customize the configuration server
- Set configuration parameters via web page and through the application code

Examples:

- [Application Data](#)
- [Config and System Params](#)
- [Config Web Interfaces](#)
- [Detached Config Object](#)
- [Change Notify](#)
- [Config Class Receive Notification](#)

12.5.1 Application Data

The examples in this directory show how to use the configuration system to create, modify, and store persistent data values that can be used in user applications.

- [BasicConfigVariables](#) - Shows how to create, modify and store basic config variable types with the config system
- [ConfigClass](#) - Expands on [BasicConfigVariables](#), and shows how to create new class types that inherit from base config objects, and how they can be combined together to create complex objects
- [CustomRangeLimit](#) - Shows how to extend NetBurner's config objects with custom behaviors like min/max checking.

See also

[config_int_limit](#)

- [NestingConfigObjects](#) - Expands on [ConfigClass](#), and shows how config objects can be grouped together to create a nested hierarchy of data

Examples:

- [Basic Config Variables](#)
- [Config Class](#)
- [Custom Range Limit Config Variables](#)
- [Nesting Config Objects](#)
- [FS_BulkStore](#)

12.5.1.1 Basic Config Variables

12.5.1.1.1 BasicConfigVariable - Config Tree Demo

A NetBurner embedded systems application demonstrating the use of persistent configuration variables through an interactive serial interface.

12.5.1.1.1.1 Overview

This application showcases the NetBurner config system, which provides persistent storage for both system-defined boot parameters and user-defined variables. The config system maintains values across power cycles, making it ideal for storing device settings, calibration data, and application parameters.

12.5.1.1.1.2 Features

- **Persistent Configuration Storage:** Variables maintain their values after power cycling
- **Interactive Serial Interface:** Menu-driven system for viewing and modifying config values
- **Dynamic Variable Management:** Create and delete config variables at runtime
- **Config Tree Visualization:** View the complete configuration hierarchy
- **Multiple Data Types:** Support for strings, integers, and unsigned integers

12.5.1.1.1.3 Configuration Variable Types

The application demonstrates usage of several config object types:

- `config_string` - String values
- `config_int` - Signed integer values
- `config_uint` - Unsigned integer values
- `config_bool` - Boolean values
- `config_double` - Double precision floating point
- `config_chooser` - Multiple choice options
- `config_pass` - Password fields
- `config_IPADDR4` - IPv4 addresses
- `config_IPADDR6` - IPv6 addresses
- `config_MACADR` - MAC addresses

12.5.1.1.1.4 Application Structure

Global Configuration Variables

The application defines three global configuration variables:

```
static config_string gSingleString(appdata, "MyString", "SName");
config_int gMyOwnVal(appdata, 199, "MyOwnValue");
config_uint gMyOwnuVal(appdata, 299, "MyOwnuValue");
```

Each variable is initialized with:

- Parent container (appdata)
- Default value
- Unique identifier name

Serial Commands

The application provides an interactive menu system accessible via serial interface:

Command	Description
C	Show Config Tree - Display complete configuration hierarchy
S	Show and Set gSingleString - View/modify string variable
V	Show and Set gMyOwnVal - View/modify signed integer variable
U	Show and Set gMyOwnuVal - View/modify unsigned integer variable
N	New/Create dynamic variable - Create new config variable at runtime
D	Delete dynamic variable - Remove dynamically created variable
Q	Query dynamic variable - Check if dynamic variable exists and show value
?	Show Command List - Display available commands

12.5.1.1.1.5 Build Requirements

Dependencies

- NetBurner SDK
- Standard C++ compiler with C++11 support
- NetBurner libraries:
 - `arp.h` - ARP protocol support
 - `http.h` - HTTP server functionality
 - `init.h` - System initialization
 - `netinterface.h` - Network interface management
 - `serial.h` - Serial communication
 - `config_server.h` - Configuration system

Compilation

Compile using the NetBurner development environment with standard build tools. Ensure all NetBurner libraries are properly linked.

12.5.1.1.1.6 Usage

Initial Setup

1. Flash the application to your NetBurner device
2. Connect via serial interface (typically 115200 baud)
3. Wait for network initialization (up to 5 seconds)

Startup Information

Upon boot, the application displays:

- Build timestamp
- Current IP address
- Auto-IP address
- Gateway address

- Config storage usage statistics

Interactive Operation

1. **View Configuration Tree:** Use command `C` to see the complete hierarchy
2. **Modify Variables:** Use commands `S`, `V`, or `U` to change values
3. **Dynamic Variables:** Use `N` to create, `D` to delete, and `Q` to query runtime variables
4. **Help:** Use `?` to display the command menu

Example Session

```
Config Tree Demo built at 12:34:56 on Jan 15 2025
'?' for commands
IP:          192.168.1.100
AutoIP :     169.254.1.100
Gateway:     192.168.1.1
```

Config space used: 1024 of 8192 bytes (max)

```
Command List
-----
C) Show Config Tree
S) Show and Set gSingleString
V) Show and Set gMyOwnVal
U) Show and Set gMyOwnuVal
N) New/Create dynamic variable
D) Delete dynamic variable
Q) Query dynamic variable
?) Show Command List
```

```
Read char S
SName old value: MyString
Enter new value: Hello World

SName new value: Hello World
```

12.5.1.1.1.7 Key Features

Persistent Storage

All configuration changes are automatically saved to non-volatile storage using `SaveConfigToStorage()`. Values persist across power cycles and system resets.

Dynamic Variable Management

The application demonstrates creating config variables at runtime using the `new` operator and properly registering them with `FixNonStaticObject()`.

Network Integration

The application initializes network interfaces and displays connection information, showing integration with NetBurner's networking stack.

System Diagnostics

Built-in system diagnostics are enabled for development and debugging purposes.

12.5.1.1.1.8 Important Notes

Variable Naming

- Each config variable must have a unique identifier name
- Duplicate names can cause issues within the config tree
- Use descriptive names for better organization

Memory Management

- Static config variables are automatically managed
- Dynamic variables require explicit cleanup using `delete`
- Call `FixNonStaticObject()` for dynamically created variables

Thread Safety

The application runs in a single-threaded context with periodic delays using `OSTimeDly()`.

12.5.1.1.1.9 Troubleshooting

Common Issues

1. **Network Connection:** Ensure proper network configuration if IP addresses show as 0.0.0.0
2. **Serial Interface:** Verify baud rate and connection settings
3. **Config Storage:** Monitor storage usage to avoid exceeding maximum capacity
4. **Variable Names:** Ensure unique naming to prevent config tree conflicts

Development Tips

- Remove `EnableSystemDiagnostics()` for production builds
- Adjust `TICKS_PER_SECOND` delays based on application requirements
- Use config tree display (C command) to verify variable organization

12.5.1.2 Config Class

12.5.1.2.1 Config Class Application

12.5.1.2.1.1 Overview

This NetBurner application demonstrates the use of the config system for creating and managing persistent configuration variables. The config system provides a way to store both system-defined settings (like network configuration) and user-defined custom variables that persist across power cycles.

12.5.1.2.1.2 Features

- **Custom Config Classes:** Demonstrates inheritance from `config_obj` to create structured configuration objects
- **Persistent Storage:** All configuration values are automatically saved to non-volatile storage
- **Interactive Menu System:** Serial port interface for viewing and modifying configuration values
- **Config Tree Visualization:** Display the complete configuration hierarchy
- **Network Integration:** Shows network interface information on startup

12.5.1.2.1.3 Architecture

Config Object Types

The application uses several NetBurner config object types:

- `config_obj` - Base container object for grouping other config objects
- `config_bool` - Boolean values
- `config_int` - Integer values
- `config_double` - Double precision floating point values
- `config_string` - String values
- `config_chooser` - Multiple choice selections
- `config_pass` - Password fields
- `config_IPADDR4` - IPv4 addresses
- `config_IPADDR6` - IPv6 addresses
- `config_MACADR` - MAC addresses

Class Hierarchy

The application defines a hierarchy of config classes:

```
config_obj
- TempRange
  - m_maxTemp (default: 100)
  - m_setTemp (default: 30)
  - m_minTemp (default: 10)
- Thermostat (inherits from TempRange)
  - m_tempRec1 (default: 101)
  - m_tempRec2 (default: 102)
  - m_active (default: true)
  - m_location (default: "Warehouse 1")
  - m_tempScale (default: "Fahrenheit")
```

Key Classes

TempRange

Base class for temperature-related configuration:

- `m_maxTemp`: Maximum temperature setting
- `m_setTemp`: Target temperature setting
- `m_minTemp`: Minimum temperature setting

Thermostat

Extended thermostat configuration inheriting from TempRange:

- `m_tempRec1` and `m_tempRec2`: Temperature recording values
- `m_active`: Enable/disable flag
- `m_location`: Physical location description
- `m_tempScale`: Temperature scale selection (Fahrenheit/Celsius/Kelvin)

cfg_class_slot and cfg_class_slots

Example classes for slot-based configuration:

- `cfg_class_slot`: Individual slot with enabled flag and name
- `cfg_class_slots`: Container for multiple slot configurations

12.5.1.2.1.4 Serial Interface Commands

The application provides an interactive menu system accessible via serial port:

Command	Description
C	Show complete config tree
L	Show and set thermostat location
M	Show and set thermostat max temperature
N	Show and set thermostat min temperature
S	Show and set thermostat set temperature
T	Show thermostat recorded temperatures
?	Show command list

12.5.1.2.1.5 Usage

1. **Power On**: The application initializes the network interface and displays network information
2. **Menu Access**: Connect to the serial port and press ? to see available commands
3. **Configuration**: Use the menu commands to view and modify configuration values
4. **Persistence**: All changes are automatically saved to non-volatile storage

12.5.1.2.1.6 Network Information

On startup, the application displays:

- Current IP address
- AutoIP address
- Gateway address
- Build timestamp

12.5.1.2.1.7 Important Notes

Config Object Rules

1. **Unique Names:** Each config object must have a unique name to avoid conflicts in the config tree
2. **Global Scope:** Config objects should be declared at global scope for proper initialization
3. **ConfigEndMarker:** Must be placed after all config data members in each class
4. **Default Values:** All config members should be given appropriate default values

12.5.1.2.1.8 Build Information

- **Application Name:** ConfigClass
- **Build Date/Time:** Displayed on startup
- **Platform:** NetBurner embedded system

12.5.1.2.1.9 Dependencies

The application requires the following NetBurner libraries:

- [arp.h](#) - ARP protocol support
- [http.h](#) - HTTP server functionality
- [init.h](#) - System initialization
- [netinterface.h](#) - Network interface management
- [serial.h](#) - Serial port communication
- [config_server.h](#) - Configuration system

12.5.1.2.1.10 Development Notes

This example builds upon the BasicConfigVariable example and demonstrates more advanced config system usage with custom classes and inheritance. It serves as a foundation for applications requiring persistent configuration management with structured data organization.

12.5.1.3 Custom Range Limit Config Variables

12.5.1.3.1 Custom Range Limit Configuration Demo

A NetBurner embedded application demonstrating advanced configuration system capabilities with custom validation objects and range-limited configuration variables.

12.5.1.3.1.1 Overview

This application showcases the NetBurner Config system by implementing custom configuration object classes that extend the base [config_obj](#) functionality. The demo specifically focuses on creating configuration variables with custom validation logic, including a novel color brightness validator.

12.5.1.3.1.2 Key Features

Built-in Configuration Objects

The application demonstrates several built-in NetBurner configuration object types:

- `config_string_limit` - String with length constraints (1-10 characters)
- `config_int_limit` - Integer with range validation (-200 to 200)
- `config_uint_limit` - Unsigned integer with range validation (0 to 300)
- `config_double_limit` - Double with range and precision validation (-10.0 to 10.0, 0.01 step)
- `config_pass_limit` - Password field with length constraints (5-128 characters)

Custom Configuration Object: Color Brightness Validator

The centerpiece of this demo is the `config_color_brightlimit` class, which validates HTML color strings based on perceived brightness:

- Accepts hex color strings in format `#rrggbb` (e.g., `#00ccff`)
- Calculates perceived brightness using ITU-R Recommendation BT.601 coefficients
- Enforces minimum and maximum brightness thresholds
- Provides meaningful error messages for invalid inputs

Brightness Calculation

The luminance calculation uses the ITU standard coefficients:

$$\text{Luminance} = (0.2582 * R + 0.66 * G + 0.0818 * B) * 100 / 255$$

Configuration Variables

The application defines several global configuration variables:

Variable	Type	Default	Range/Limit	Description
<code>gSingleString</code>	<code>config_string_limit</code>	"SNameLimit"	1-10 chars	String length demo
<code>gMyOwnVal</code>	<code>config_int_limit</code>	199	-200 to 200	Integer range demo
<code>gMyOwnuVal</code>	<code>config_uint_limit</code>	299	0 to 300	Unsigned int range demo
<code>gMyOwndVal</code>	<code>config_double_limit</code>	9.0	-10.0 to 10.0	Double precision demo
<code>gMyOwnpVal</code>	<code>config_pass_limit</code>	""	5-128 chars	Password field demo
<code>gMyDarkColor</code>	<code>config_color← _brightlimit</code>	"#331122"	0-50% brightness	Dark color validator
<code>gMyLightColor</code>	<code>config_color← _brightlimit</code>	"#cceeef"	50-100% brightness	Light color validator

12.5.1.3.1.3 Interactive Commands

The application provides a serial console interface with the following commands:

- **C** - Show Config Tree: Display the complete configuration hierarchy
- **S** - Show and Set `gSingleString`: Modify the string limit example
- **V** - Show and Set `gMyOwnVal`: Modify the integer limit example
- **U** - Show and Set `gMyOwnuVal`: Modify the unsigned integer limit example

- **N** - New/Create dynamic variable: Create a new config variable at runtime
- **D** - Delete dynamic variable: Remove the dynamically created variable
- **Q** - Query dynamic variable: Display the value of the dynamic variable
- **?** - Show Command List: Display available commands

12.5.1.3.1.4 Technical Implementation

Custom Validation Logic

The `config_color_brightlimit` class demonstrates several advanced concepts:

1. **Server-side Validation:** The `TestNewValue()` method provides robust server-side validation with custom error messages
2. **Client-side Schema:** The `ExtendedSchema()` method adds HTML pattern validation for web-based configuration
3. **Operator Overloading:** Multiple assignment operators handle different input types while maintaining validation
4. **Error Reporting:** Uses `LogParseError()` to provide user-friendly error messages

Dynamic Configuration Management

The application shows how to:

- Create configuration objects dynamically at runtime
- Properly manage memory for dynamic config objects
- Use `FixNonStaticObject()` for runtime-created config variables

12.5.1.3.1.5 Web Interface Integration

The configuration system automatically generates a web-based configuration interface at the device's IP address. The custom validation rules are enforced both in the web UI (client-side) and on the device (server-side).

12.5.1.3.1.6 Network Information

Upon startup, the application displays:

- Device IP address
- AutoIP address
- Gateway address
- Configuration space usage statistics

12.5.1.3.1.7 Building and Running

This application is designed for NetBurner embedded systems. Ensure you have:

- NetBurner development environment installed
- Compatible NetBurner hardware platform
- Network connectivity for web-based configuration access

12.5.1.3.1.8 Configuration Storage

All configuration changes are automatically persisted to non-volatile storage using the NetBurner configuration system. Changes survive device reboots and power cycles.

12.5.1.3.1.9 Use Cases

This demo is particularly useful for:

- Learning NetBurner configuration system concepts
- Implementing custom validation logic for configuration parameters
- Creating domain-specific configuration objects
- Understanding client-server validation patterns
- Building robust embedded configuration systems

12.5.1.3.1.10 Error Handling

The application includes comprehensive error handling for:

- Invalid color format strings
- Out-of-range brightness values
- Network connectivity issues
- Memory management for dynamic objects

12.5.1.3.1.11 Extension Points

The framework demonstrated here can be extended to create custom validators for:

- IP address ranges
- Time/date validation
- File path validation
- Custom protocol parameters
- Hardware-specific constraints

12.5.1.4 Nesting Config Objects

12.5.1.4.1 NetBurner Config System - Nesting Config Objects

12.5.1.4.1.1 Overview

This application demonstrates the NetBurner configuration system, specifically showcasing how configuration objects can be nested to create hierarchical data structures. The config system is used to store both boot/interface information and user-defined persistent variables in flash memory as JSON data.

12.5.1.4.1.2 Application Description

The **NestingConfigObjects** example builds upon the basic ConfigClass example by creating nested configuration classes that demonstrate a real-world thermostat control system. The application models a group of thermostats, each with their own temperature settings, location information, and operational parameters.

12.5.1.4.1.3 Class Hierarchy

The application uses a three-level inheritance and nesting structure:

1. TempRange (Base Class)

- **Purpose:** Defines basic temperature range parameters
- **Members:**
 - `m_maxTemp`: Maximum temperature (default: 100)
 - `m_setTemp`: Set temperature (default: 30)
 - `m_minTemp`: Minimum temperature (default: 10)

2. Thermostat (Extends TempRange)

- **Purpose:** Represents a complete thermostat with additional features
- **Members:**
 - Inherits all TempRange members
 - `m_tempRec1`: Temperature record 1 (default: 101)
 - `m_tempRec2`: Temperature record 2 (default: 102)
 - `m_active`: Active status (default: true)
 - `m_location`: Location string (default: "Warehouse 1")
 - `m_tempScale`: Temperature scale chooser (Fahrenheit/Celsius/Kelvin)

3. ThermoGroup (Container Class)

- **Purpose:** Groups multiple thermostats together
- **Members:**
 - `thermo_1`: First thermostat instance
 - `thermo_2`: Second thermostat instance

12.5.1.4.1.4 Available Config Object Types

The NetBurner config system supports the following object types:

- `config_obj`: Base container object for other objects
- `config_bool`: Boolean values
- `config_int`: Integer values
- `config_double`: Double precision floating point
- `config_string`: String values
- `config_chooser`: Multiple choice options
- `config_pass`: Password fields
- `config_IPADDR4`: IPv4 addresses
- `config_IPADDR6`: IPv6 addresses
- `config_MACADR`: MAC addresses

12.5.1.4.1.5 Interactive Commands

The application provides a serial console interface with the following commands:

- **C**: Show complete configuration tree
- **L**: Show and set thermostat locations
- **M**: Show and set maximum temperatures
- **N**: Show and set minimum temperatures
- **S**: Show and set target temperatures
- **T**: Display recorded temperature values
- **?**: Show command list

12.5.1.4.1.6 Key Features

Persistent Storage

- All configuration changes are automatically saved to flash memory
- Data persists across power cycles and system resets
- Storage format is JSON for human readability

Hierarchical Organization

- Nested structure allows logical grouping of related parameters
- Each object has a unique name for identification
- Clear parent-child relationships in the configuration tree

Real-time Updates

- Configuration changes take effect immediately
- Interactive console allows runtime modification
- No need to restart the application after changes

12.5.1.4.1.7 Usage Instructions

1. **Compile and Deploy:** Build the application and deploy to your NetBurner device
2. **Connect Serial Console:** Use a terminal program to connect at 115200 baud
3. **View Network Info:** The application displays IP address, AutoIP, and gateway on startup
4. **Access Commands:** Type '?' to see available commands
5. **Modify Settings:** Use the letter commands to update thermostat parameters
6. **View Configuration:** Use 'C' to display the complete configuration tree

12.5.1.4.1.8 Important Notes

- **Unique Names:** Each config object must have a unique name to avoid conflicts
- **Global Scope:** Config objects should be declared at global scope
- **ConfigEndMarker:** Always include this marker after the last data member in each class
- **Automatic Persistence:** Changes are automatically saved to flash storage

12.5.1.4.1.9 Development Guidelines

When extending this example:

1. Always use unique names for config objects
2. Place `ConfigEndMarker` after the last data member in each class
3. Attach config objects to the `appdata` config object for proper integration
4. Use appropriate default values that make sense for your application
5. Consider the hierarchical structure when designing your config classes

12.5.1.5 FS_BulkStore

12.5.1.5.1 Config FS BulkStore Application

12.5.1.5.1.1 Overview

The Config FS BulkStore application demonstrates how to use the `config_fs_bulkstore` class to store large binary blobs in a filesystem while managing their contents through the NetBurner configuration system. This example shows how to handle persistent storage of large data objects that might exceed the limitations of the standard configuration space.

12.5.1.5.1.2 Features

- **Bulk Storage Management:** Store large binary data blobs in the filesystem
- **Configuration Integration:** Manage bulk storage objects via the NetBurner config system
- **Flexible Storage Options:** Support for both fixed and movable storage paths
- **File System Operations:** Built-in commands for file manipulation (`cat`, `rm`, `fsls`)
- **Command-Line Interface:** Interactive serial console with argument parsing
- **Network Support:** Full network stack initialization with diagnostics

12.5.1.5.1.3 Architecture

Core Components

1. **config_fs_bulkstore:** Main class for managing large binary blobs
 - Stores data in filesystem while providing config system interface
 - Supports both fixed and movable storage paths
 - Handles runtime path changes when configured as movable
2. **Argument Parser:** Robust command-line argument parsing system
 - Supports various data types (`int`, `uint`, `string`, `bool`, `flags`)
 - Handles quoted arguments and escape sequences
 - Provides detailed error reporting
3. **Console Commands:** File system management commands
 - `cat`: Display file contents
 - `rm`: Delete files
 - `fsls`: List filesystem contents with tree structure

Configuration Objects

The application creates two bulk storage objects:

```
// Fixed path bulk store
static config_fs_bulkstore gMyBulkStore{
    appdata, "/MyBulkStore.bin", false, "MyBulkStore"
};

// Movable path bulk store
static config_fs_bulkstore gMyMovableBulkStore{
    appdata, "/MyMoveableBulkStore.bin", true, "MyMovableBulkStore", NULL, false
};
```

12.5.1.5.1.4 Usage

Starting the Application

1. The application initializes the filesystem (if not already available)
2. Sets up the network stack with diagnostics enabled
3. Waits for network connectivity (up to 5 seconds)
4. Drops to serial configuration mode for interactive use

Available Commands

- **cat** <file_path>: Display the contents of a file
- **rm** <file_path>: Delete a file from the filesystem
- **fsls**: List all files and directories in a tree structure
- **help**: Show available commands (inherited from config system)

Command Line Features

The argument parser supports:

- **Flags**: Single character flags (e.g., -a, -v)
- **Quoted Arguments**: Single and double quotes with escape sequences
- **Multiple Data Types**: Integers, unsigned integers, strings, booleans
- **Optional/Required Parameters**: Flexible parameter definitions

12.5.1.5.1.5 Build Requirements

- NetBurner development environment
- Platform with filesystem support (or EDFS Std filesystem)
- Standard C++ compiler support

12.5.1.5.1.6 Configuration Space

The application reports configuration space usage on startup:

- Shows current usage vs. maximum available space
- Helps monitor config system utilization

12.5.1.5.1.7 Error Handling

The argument parser provides detailed error messages:

- Unknown flag identification
- Too few/many arguments detection
- Invalid argument format reporting
- Clear error location indication

12.5.1.5.1.8 Network Features

- Full network stack initialization
- [ARP](#) and HTTP protocol support
- System diagnostics enabled for debugging
- 5-second network connectivity timeout

12.5.1.5.1.9 File System Integration

The application integrates with the NetBurner filesystem:

- Automatic filesystem initialization on platforms without native support
- Full file manipulation capabilities
- Recursive directory traversal
- File size reporting

12.5.1.5.1.10 Example Usage

```
Config Tree Demo built at 12:34:56 on Jan 01 2024
'?' for commands

Config space used: 1024 of 65536 bytes (max)
Aborted to Serial Config (BOOT) to continue boot
type help for commands
> cat /MyBulkStore.bin
> fsls
> rm /temporary_file.txt
```

12.5.1.5.1.11 Technical Details

Memory Management

- Uses NetBurner's SMPool buffer system for efficient memory usage
- Configurable buffer sizes for different operations
- Automatic buffer cleanup and management

Threading

- Main application runs in continuous loop with 1-second delays
- Uses NetBurner's OSTimeDly for cooperative multitasking
- Thread-safe configuration system integration

Platform Support

- Conditional filesystem initialization based on platform capabilities
- Portable across different NetBurner hardware platforms
- Configurable for different storage backend implementations

12.5.1.5.1.12 Development Notes

This application serves as a reference implementation for:

- Integrating large data storage with NetBurner's config system
- Building interactive command-line interfaces
- Implementing argument parsing
- Managing filesystem operations in embedded systems

12.5.2 Config and System Params

The examples in this directory show how to access and modify system values through the configuration system.

- ShowConfig - Shows how to view the entire JSON config blob
- ModifyBoot - Shows how to view and modify the Boot system values through the global config variable `monitor_config`
- ModifyEthernet - Shows how to view and modify the interface system values (default is typically Ethernet0) through the config system
- Disable system features and network ports

Examples

- [Disable System Features](#)
- [Modify Boot](#)
- [Modify Ethernet Interface](#)
- [Show Configuration](#)

12.5.2.1 Disable System Features

12.5.2.1.1 Disable System Features

A NetBurner example application that demonstrates how to disable various system networking features for enhanced security. This application allows users to disable features like AutoIP, Discovery Server, PING/Echo responses, and web services to reduce the attack surface and prevent these services from appearing in network security scans.

12.5.2.1.1.1 Overview

The DisableSysFeatures application provides an interactive command-line interface to control system networking features that may be undesirable in production environments. By disabling these features, administrators can:

- Close unnecessary network ports
- Reduce visibility in network security scans
- Minimize attack vectors
- Implement security best practices for embedded devices

12.5.2.1.1.2 Features

System Settings Control

The application allows runtime configuration of the following networking features:

- **AutoIP:** Automatic IP address assignment when [DHCP](#) is unavailable
- **Discovery Server:** NetBurner's `discover.netburner.com` service registration
- **PING/Echo Responses:** ICMP ping and UDP echo port responses
- **Configuration Persistence:** Save changes to flash memory

Interactive Menu System

The application provides a simple command-line interface with the following options:

1. Show current system settings
2. Disable networking features
3. Re-enable networking features
4. Save configuration to flash memory

12.5.2.1.1.3 Configuration Options

AutoIP Disabling

```
ib->ip4.autoip = false;
```

Disables automatic IP address assignment, preventing the device from self-assigning an IP when DHCP is unavailable.

Discovery Server Disabling

```
ib->discovery_server = "";
```

Prevents the device from registering with NetBurner's discovery service at discover.netburner.com.

PING and Echo Response Disabling

```
ib->supress_default_responses = true;
```

Disables responses to ICMP ping requests and UDP echo port communications.

Web Server Control

Web services can be disabled by not calling `StartHttp()` and `StartHttps()` functions in `UserMain()`. For HTTPS-only operation:

```
StartHttps(443, 0); // HTTPS on 443, HTTP disabled
```

Configuration Server Control

The device configuration web interface can be disabled by removing the `EnableSecureConfigServer()` call, or configured for HTTPS-only access:

```
EnableSecureConfigServer(true); // HTTPS only  
EnableSecureConfigServer(false); // Both HTTP and HTTPS
```

12.5.2.1.1.4 Usage

Runtime Operation

1. Compile and load the application onto your NetBurner device
2. Connect via serial console
3. Use the interactive menu to view and modify system settings
4. Save changes to flash memory using the 'S' command

Commands

- **1**: Display current system settings
- **2**: Disable AutoIP, Discovery Server, and PING/Echo responses
- **3**: Re-enable all networking features
- **S**: Save current configuration to flash memory
- **?**: Show command menu

12.5.2.1.1.5 Security Considerations

Network Hardening

This application is particularly useful for:

- Production deployments requiring minimal network exposure
- Environments with strict security scanning requirements
- IoT devices that need to minimize their network footprint
- Systems requiring compliance with security standards

Alternative Configuration Methods

These settings can also be configured through:

- The device's Configuration Server web page (if enabled)
- Direct modification of configuration parameters
- Factory default settings

12.5.2.1.1.6 Application Structure

Key Components

- **showSystemSettings()**: Displays current networking feature states
- **disableSystemSettings()**: Disables AutoIP, Discovery Server, and PING/Echo
- **enableSystemSettings()**: Re-enables all networking features
- **Interactive Menu Loop**: Provides user interface for runtime control

Network Interface Management

The application uses the first available network interface (`GetFirstInterface()`) for configuration. For multi-interface systems, additional logic would be needed to select and configure specific interfaces.

12.5.2.1.1.7 Use Cases

Production Environments

- Embedded systems requiring minimal network services
- Industrial control systems with security requirements
- IoT devices deployed in sensitive network environments

Security Testing

- Verification of disabled services
- Network security scan preparation
- Compliance testing and validation

Development and Testing

- Feature testing during development
- Network behavior verification
- Security configuration validation

12.5.2.1.1.8 Notes

- Changes made through the application interface are runtime modifications
- Use the 'S' command to persist changes to flash memory
- Configuration Server access may be required for certain changes if the application interface is unavailable
- Network security scans should be performed after disabling features to verify effectiveness

12.5.2.2 Modify Boot

12.5.2.2.1 NetBurner Configuration System - Boot Parameter Modification

12.5.2.2.1.1 Overview

This application demonstrates how to view and modify system boot configuration parameters on NetBurner embedded devices. The program provides an interactive command-line interface for managing persistent system settings such as baud rates, boot delays, authentication credentials, and other boot-related parameters.

12.5.2.2.1.2 Application Name

`SystemParamsModifyBoot`

12.5.2.2.1.3 Purpose

The NetBurner config system stores both system-defined boot and interface information (like default baud rates and current IP settings) as well as user-defined persistent variables. This example specifically focuses on accessing and modifying the "Boot" configuration record through a menu-driven interface.

12.5.2.2.1.4 Key Features

- **Interactive Menu System:** Command-line interface with single-character commands
- **Real-time Configuration Updates:** Changes are immediately saved to persistent storage
- **Web Interface Integration:** Configuration changes can be viewed via web browser
- **Multiple Data Types:** Supports string, integer, and boolean configuration parameters
- **Network Information Display:** Shows current IP configuration on startup

12.5.2.2.1.5 Supported Configuration Parameters

Boot Configuration Options

- **Abort Command (A):** Set the command string used to abort boot process
- **Baud Rate (B):** Configure serial communication baud rate
- **Boot Delay (D):** Set delay time before boot process begins
- **User Credentials (P):** Set system username and password
- **Quiet Boot (Q):** Enable/disable quiet boot mode (boolean)

System Information

- **Config Tree (C):** Display the complete configuration tree structure
- **Help (?):** Show available commands

12.5.2.2.1.6 Technical Implementation

Core Components

1. Configuration Data Types:

- `config_string`: For text-based settings
- `config_int`: For numeric values
- `config_bool`: For true/false settings

2. Key Functions:

- `UpdateConfigString()`: Handle string parameter updates

- `UpdateConfigInt()`: Handle integer parameter updates
- `UpdateConfigBool()`: Handle boolean parameter updates
- `ShowTree()`: Display configuration tree structure
- `SaveConfigToStorage()`: Persist changes to storage

3. External Dependencies:

- `MonitorRecord monitor_config`: Global variable exposing boot config values
- NetBurner system libraries for networking, serial communication, and configuration management

12.5.2.2.1.7 Usage Instructions

Getting Started

1. **Build and Deploy**: Compile the application and deploy it to your NetBurner device
2. **Serial Connection**: Connect to the device via serial port (default: 115200 baud)
3. **Network Setup**: Ensure the device is connected to the network (5-second timeout for network initialization)

Interactive Commands

Once the application starts, you'll see network information and a command prompt. Enter any of the following single characters:

- `A` - Modify abort boot command string
- `B` - Change baud rate setting
- `C` - Display configuration tree
- `D` - Set boot delay value
- `P` - Update username and password
- `Q` - Toggle quiet boot mode
- `?` - Show command list

Web Interface

The application also provides a web interface for viewing configuration changes:

- Navigate to `http://[device-ip]:20034` in your web browser
- Refresh the page to see updated values after making changes via the command line

12.5.2.2.1.8 Configuration Flow

1. **Display Current Value**: Shows the existing configuration value
2. **User Input**: Prompts for new value entry
3. **Validation and Assignment**: Processes and assigns the new value
4. **Automatic Save**: Calls `SaveConfigToStorage()` to persist changes
5. **Confirmation**: Displays the updated value

12.5.2.2.1.9 Network Information Display

On startup, the application displays:

- Current IP Address
- Auto-assigned IP Address
- Gateway Address

12.5.2.2.1.10 Build Information

The application displays build timestamp information showing when it was compiled, helping with version tracking and debugging.

12.5.2.2.1.11 Important Notes

- **Serial Port Management:** The application closes and reopens the serial port to ensure proper communication
- **System Diagnostics:** Includes diagnostic capabilities (should be disabled for production)
- **Persistent Storage:** All configuration changes are immediately saved to non-volatile storage
- **Network Dependency:** Waits up to 5 seconds for network activation before proceeding

12.5.2.2.1.12 Error Handling

The application includes basic input validation and handles:

- String input with automatic null termination
- Integer conversion from string input
- Boolean parsing (accepts 'T'/t' for true, anything else for false)

12.5.2.3 Modify Ethernet Interface

12.5.2.3.1 ModifyEthernet Configuration Example

12.5.2.3.1.1 Overview

This NetBurner example demonstrates how to view and modify Ethernet interface configuration parameters using the embedded system configuration system. The application provides an interactive command-line interface for managing network settings that are stored persistently in the device's configuration record.

12.5.2.3.1.2 Features

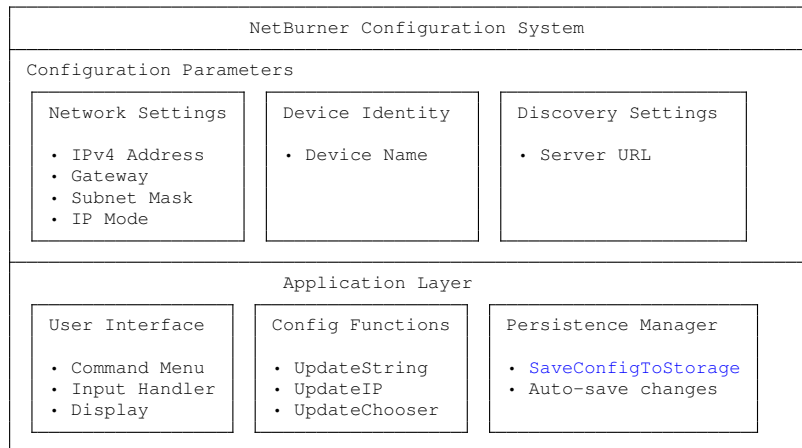
- Interactive command-line interface for network configuration
- Persistent storage of configuration changes
- Real-time modification of network parameters
- Configuration tree visualization
- Support for multiple IP configuration modes
- Automatic configuration validation and saving

12.5.2.3.1.3 Project Structure

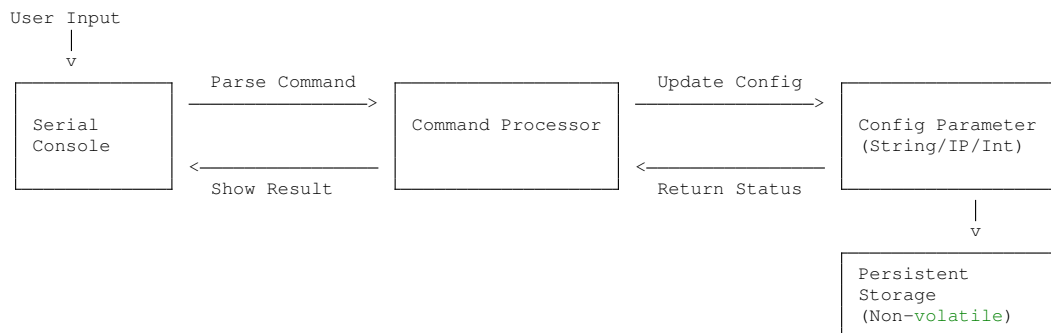
```
ModifyEthernet/  
├── src/  
│   ├── main.cpp      # Main application with interactive interface  
│   └── makefile       # Build configuration
```

12.5.2.3.1.4 Architecture

System Components



Configuration Data Flow



12.5.2.3.1.5 Configurable Parameters

Network Settings

The application allows modification of the following network parameters:

IPv4 Static IP Address

- **Command:** A
- **Type:** IP Address ([config_IPADDR4](#))
- **Description:** Device's static IP address
- **Example:** 192.168.1.100

IPv4 Static IP Gateway

- **Command:** G
- **Type:** IP Address ([config_IPADDR4](#))
- **Description:** Network gateway address
- **Example:** 192.168.1.1

IPv4 Static IP Mask

- **Command:** M
- **Type:** IP Address ([config_IPADDR4](#))
- **Description:** Subnet mask
- **Example:** 255.255.255.0

IPv4 Mode

- **Command:** O
- **Type:** Chooser ([config_chooser](#))
- **Description:** IP configuration mode
- **Options:** Static, [DHCP](#), AutoIP

Device Identity**Device Name**

- **Command:** N
- **Type:** String ([config_string](#))
- **Description:** Human-readable device identifier
- **Example:** "Production_Controller_01"

Discovery Settings**Discover Server URL**

- **Command:** U
- **Type:** String ([config_string](#))
- **Description:** Discovery server endpoint
- **Example:** "http://discovery.company.com/devices"

12.5.2.3.1.6 Command Interface**Available Commands**Command List

```

A) Show and Set IPv4 Static IP Address
C) Show Config Tree
G) Show and Set IPv4 Static IP Gateway
M) Show and Set IPv4 Static IP Mask
N) Show and Set Device Name
O) Show and Set IPv4 Mode
U) Show and Set Discover Server URL
?) Show Commands

```

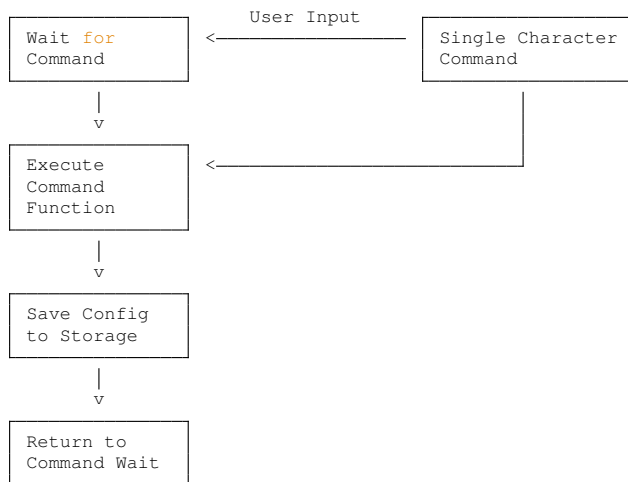
Command Usage Flow

Application Startup

↓
v

Display Info
 • Current IP
 • AutoIP
 • Gateway

↓
v



12.5.2.3.1.7 Function Reference

Core Update Functions

UpdateConfigString(config_string &confStr)

Purpose: Updates string-based configuration parameters

- Displays current value
- Prompts for new value via serial input
- Updates configuration parameter
- Saves changes to persistent storage

UpdateConfigInt(config_int &confInt)

Purpose: Updates integer-based configuration parameters

- Displays current value
- Prompts for new numeric value
- Converts string input to integer
- Saves changes to persistent storage

UpdateConfigIpaddr(config_IPADDR4 &confIp)

Purpose: Updates IP address configuration parameters

- Displays current IP address in dotted decimal notation
- Prompts for new IP address string
- Converts string to IP address format
- Validates and saves changes

UpdateConfigChooser(config_chooser &confChooser)

Purpose: Updates chooser-based configuration parameters

- Displays current selection
- Shows available choices
- Validates user input against allowed options
- Saves valid selection

Utility Functions

ShowCommandList ()

Purpose: Displays available command menu

- Lists all interactive commands
- Shows command letters and descriptions
- Called on startup and when '?' is pressed

ShowTree ()

Purpose: Displays configuration tree structure

- Shows hierarchical configuration layout
- Useful for debugging and understanding config structure
- External function (implementation not shown in main.cpp)

Main Application Function

UserMain (void *pd)

Main application entry point that:

- Initializes network stack with `init ()`
- Enables system diagnostics
- Waits for active network connection
- Configures serial port for console interaction
- Displays initial network information
- Implements main command processing loop

12.5.2.3.1.8 Technical Implementation

Serial Console Setup

Serial Configuration:

Baud Rate: 115200
Data Bits: 8
Stop Bits: 1
Parity: None
Flow Control: No

The application reconfigures the serial port during initialization:

1. Closes existing serial connection
2. Reopens with 115200 baud rate
3. Uses standard input/output for user interaction

Configuration Persistence

All configuration changes are automatically saved using:

`SaveConfigToStorage () ;`

This function:

- Writes changes to non-volatile storage
- Ensures persistence across device reboots
- Provides immediate effect of configuration changes

Network Interface Access

The application uses the first available network interface:

```
InterfaceBlock *firstInterface = GetInterfaceBlock(GetFirstInterface());
```

This approach:

- Automatically selects the primary network interface (typically Ethernet0)
- Provides access to all interface configuration parameters
- Works with different NetBurner hardware platforms

12.5.2.3.1.9 Usage Examples

Setting IP Address

```
Read char A
IPv4Address old value: 192.168.1.100
Enter new value: 192.168.1.200

IPv4Address new value: 192.168.1.200
```

Changing IP Mode

```
Read char O
IPv4Mode old value: Static
Enter new value. Choices are {Static,DHCP,AutoIP}: DHCP

IPv4Mode new value: DHCP
```

Setting Device Name

```
Read char N
DeviceName old value: NetBurner_Device
Enter new value: Production_Controller

DeviceName new value: Production_Controller
```

12.5.2.3.1.10 Build Instructions

1. Prerequisites:

- NetBurner development environment installed
- Target NetBurner hardware platform
- Serial console application (115200 baud)

2. Building:

```
make clean
make
```

3. Deployment:

```
make load DEVIP=<device_ip_address>
```

4. Console Connection:

- Connect serial cable to device
- Open terminal at 115200 baud
- Reset device to see startup messages

12.5.2.3.1.11 Development Notes

Dependencies

Required NetBurner system libraries:

- [arp.h](#) - ARP protocol support
- [http.h](#) - HTTP protocol support

- `init.h` - System initialization
- `netinterface.h` - Network interface management
- `serial.h` - Serial communication
- `config_server.h` - Configuration system
- `stdlib.h` - Standard C library functions

Build Configuration

- **Application Name:** SystemParamsModifyEthernet
- **Source Files:** src/main.cpp
- **Target Platform:** All NetBurner platforms
- **Build System:** Standard NetBurner makefile

Memory Considerations

- Configuration parameters are stored in device flash memory
- Changes are persistent across power cycles
- No additional RAM allocation required for configuration storage
- Serial input buffers are statically allocated (80 bytes for strings, 25 bytes for integers)

Error Handling

The application includes basic error handling:

- Input validation for chooser parameters
- Automatic null termination of string inputs
- Configuration save verification
- Network initialization checks

12.5.2.3.1.12 Security Considerations

Important Security Notes**:

- Serial console access provides full configuration control
- No authentication mechanism implemented
- Consider adding access control for production environments
- Configuration changes take effect immediately without confirmation

12.5.2.3.1.13 Compatibility

- Compatible with all NetBurner embedded platforms
- Requires NetBurner RTOS version 3.x or later
- Works with Ethernet and Wi-Fi interfaces
- Supports IPv4 configuration (IPv6 not included in this example)

12.5.2.4 Show Configuration

12.5.2.4.1 System Configuration Display Application

12.5.2.4.1.1 Overview

The System Configuration Display Application is an embedded C++ program designed to demonstrate how to read and display system configuration settings stored in JSON format. This application provides a simple command-line interface for examining device configuration data and memory usage statistics.

12.5.2.4.1.2 Features

- **Configuration Tree Display:** Read and display the complete system configuration structure in JSON format
- **Memory Usage Monitoring:** Show how much memory is being used by configuration settings versus total available space
- **Network Information:** Display current network interface settings including IP address, AutoIP, and Gateway
- **Interactive Command Interface:** Simple character-based menu system for user interaction

12.5.2.4.1.3 Application Details

Main Components

- **Application Name:** ShowSystemConfiguration
- **Primary Function:** Configuration system exploration and diagnostics
- **Interface:** Serial port communication with command-driven menu system

System Configuration

The configuration system stores critical device settings including:

- Device boot parameters
- Network interface settings
- Serial interface configuration
- All settings stored in JSON format for easy parsing and modification

Network Features

The application automatically:

- Waits for active network connection (5-second timeout)
- Displays current network configuration on startup
- Shows IP address, AutoIP, and Gateway settings for the first network interface

12.5.2.4.1.4 User Interface

Available Commands

Command	Description
C	Show Configuration Tree (displays complete JSON structure)
S	Show Configuration Memory Usage (current usage vs. available space)
?	Show Command List (displays this help menu)

Command Operation

1. Launch the application

2. Wait for network initialization
3. Review displayed network information
4. Use single-character commands to interact with the system
5. Commands are case-insensitive

12.5.2.4.1.5 Technical Implementation

Dependencies

The application includes several system libraries:

- `http.h` - HTTP protocol functionality
- `init.h` - System initialization
- `netinterface.h` - Network interface management
- `serial.h` - Serial communication
- `stdlib.h` - Standard library functions
- `config_server.h` - Configuration management system

Key Functions

- `ShowTree()` - Displays the configuration tree structure in JSON format
- `ShowConfigMemoryUsage()` - Reports memory usage statistics
- `ShowCommandList()` - Displays available user commands
- `UserMain()` - Main application loop and initialization

System Diagnostics

The application enables system diagnostics by default, which should be disabled for production deployments to optimize performance and reduce debug output.

12.5.2.4.1.6 Memory Management

The application provides real-time monitoring of configuration memory usage through the `ConfigSize()` and `ConfigMaxSize()` functions, allowing administrators to:

- Monitor current configuration storage usage
- Understand available configuration space
- Plan for configuration expansion
- Identify potential memory constraints

12.5.2.4.1.7 Network Initialization

The application implements a robust network startup sequence:

1. Initialize core system components
2. Enable system diagnostics (development mode)
3. Wait for network to become active (5-second timeout)
4. Display network configuration information
5. Enter interactive command mode

12.5.2.4.1.8 Usage Example

```
Config Tree Demo built at 12:34:56 on Jan 01 2025
'?' for commands

IP:           192.168.1.100
AutoIP :      169.254.1.1
Gateway:      192.168.1.1

Command List
-----
C = Show Config Tree
S = Show Config Memory Usage
? = Show Commands

Configuration Memory Used: 1024 bytes, out of 8192 bytes
```

12.5.2.4.1.9 Development Notes

- The application runs in a continuous loop, checking for user input every 250ms
- All user commands are processed immediately when characters are available
- System diagnostics are enabled for development and debugging purposes
- The application is designed for embedded systems with limited resources

12.5.2.4.1.10 Production Considerations

Before deploying to production:

1. Remove or disable `EnableSystemDiagnostics()` call
2. Consider adjusting the network timeout based on your environment
3. Implement proper error handling for network initialization failures
4. Add logging mechanisms for configuration changes if needed

12.5.2.4.1.11 Build Information

The application displays build timestamp information on startup, showing both compilation time and date for version tracking and debugging purposes.

12.5.3 Config Web Interfaces

The system configuration web page interface is located on network port number 20034. These example demonstrate how to provide the same configuration interface as part of the application web server (default port numbers are 80 or 443 for SSL/TLS), so that the configuration web pages are consistent with the rest of the application web interface.

Examples:

- [Basic Web Config](#)
- [Basic Web Config with SSL/TLS](#)
- [Config Tags](#)
- [Custom Web Config](#)
- [Custom Web System Config](#)
- [HTML Only Configuration](#)

12.5.3.1 Basic Web Config

The system configuration web page interface is located on network port number 20034. This example demonstrates how to provide the same configuration interface as part of your application web server on port 80 (or 443 for SSL/TLS), so that you can customize it to be consistent with the rest of your application web pages.

The web page is dynamically generated, and provides access to view and/or modify boot and interface settings, such as default serial port baud rates and IP addresses, as well as application specific data.

The code used to generate and process the web form is in `html/index.html`. This example does not use any external JavaScript libraries, and all of the form generation code is located at the bottom of `index.html`. For a more advanced example that utilized Bootstrap and jQuery, please see the CustomWebConfig example.

Currently available config objects that can be used are as follows:

- `config_obj` // base object, used as a container for other objects
- `config_bool` // boolean
- `config_int` // integer
- `config_double` // double float
- `config_string` // string
- `config_chooser` // multiple choice option
- `config_pass` // password
- `config_IPADDR4` // IP4 Address
- `config_IPADDR6` // IP6 Address
- `config_MACADR` // MAC Address

Note that because we are only processing the config object, we are able to avoid writing a custom POST handler that is used in the other examples. If you include more to your web interface than is provided here, that will need to be incorporated as well.

12.5.3.2 Basic Web Config with SSL/TLS

This example is identical to the BasicWebConfig example with the addition of SSL/TLS. Please refer to that example documentation for a description of how the configuration server mirror operates.

Certificates can be handled a number of ways:

- Automatically with ACME (preferred)
- Auto generation feature of the NetBurner ssl/tls library
- Self-signed and either compiled into the application or uploaded to a file system.

To simplify things, this example provides files for a compiled certificate and key, named `ServerCert.cpp` and `ServerKey.cpp`.

To generate your own self-signed certificates, use the script files in the `\nburn\CreateCerts` folder:

- `makeca.bat` to generate a Certificate Authority certificate and key
- `makeserver.bat` to generate the HTTPS server certificate and key

For the common name (CN), use the IP address of your device, then copy the generated `.cpp` files to your project's `src` folder.

Please refer to the `\nburn\examples\ssl` folder for more advanced SSL/TLS certificate handling.

12.5.3.3 Config Tags

12.5.3.3.1 HTML Configuration Web Application

12.5.3.3.1.1 Overview

This application demonstrates how to create a web-based configuration interface for embedded systems using dynamic HTML content tags. It provides a complete example of displaying and modifying configuration variables through a web browser interface.

The example demonstrates how to obtain just the raw data from the board, as well as how it can be styled using Bootstrap and CSS. While VARIABLE and CPPCALL HTML tags can be used to present the same information, the CONFIG tags simplify things by combining the data with HTML code to create input fields, tables and value displays.

12.5.3.3.1.2 Features

Dynamic Web Content Tags

The application showcases three main configuration tags:

- **CONFIGVALUE** - Displays configuration variable values
- **CONFIGINPUT** - Creates input fields for modifying variables
- **CONFIGTABLE** - Generates HTML tables with configuration data

Web Interface Options

- **Raw Configuration View** - Shows configuration data as it comes directly from the module
- **Styled Configuration View** - Presents the same data with Bootstrap CSS styling and tabbed interface

Configuration Variable Types

- **config_uint** - Unsigned integer configuration values
- **config_string** - String configuration values
- System configuration variables (IP addresses, network settings, etc.)

12.5.3.3.1.3 File Structure

```
/
- main.cpp           # Main application code
- htmlvars.h         # Configuration variable declarations
- index.html         # Main landing page
- raw_config.html    # Raw configuration display
- styled_config.html # Bootstrap-styled configuration
```

12.5.3.3.1.4 Configuration Variables

The application defines two custom configuration variables:

1. gMyOwnuVal (config_uint)

- Default value: 299
- Name: "MyOwnuValue"
- Description: "This is the hint for my value"

2. gMyOwnStrVal (config_string)

- Default value: "My Test String"
- Name: "MyOwnString"
- Description: "This is the hint for my string"

12.5.3.3.1.5 Web Interface

Main Page (index.html)

The entry point provides navigation to two different views of the configuration system:

- Link to styled configuration page (Bootstrap-enhanced)
- Link to raw configuration page (unstyled, raw data)

Raw Configuration Page (raw_config.html)

Demonstrates basic usage of configuration tags:

- Display values using CONFIGVALUE tags
- Create input forms using CONFIGINPUT tags
- Generate tables using CONFIGTABLE tags
- Show system variables like IP addresses

Styled Configuration Page (styled_config.html)

Features a modern Bootstrap-based interface with:

- Tabbed navigation (Display Values, Input Fields, Tables)
- Responsive design with Bootstrap CSS
- Enhanced styling and user experience
- Code examples with syntax highlighting

12.5.3.3.1.6 Form Handling

The application provides two approaches for handling form submissions:

Simple Handler (Recommended)

```
HttpPostConfigVariableHandler post1Form("post1.html", "index.html");
HttpPostConfigVariableHandler post2Form("post2.html", "index.html");
```

Custom Callback Handler

```
HttpPostVariableListCallback postForm1("post1.html", ConfigPostCallBack);
```

The custom callback approach allows for more complex processing, including:

- Handling non-configuration variables
- Custom reboot logic
- Advanced form validation

12.5.3.3.1.7 Configuration Tag Usage

Display Values

```
<!-- Using configuration path -->
<!--CONFIGVALUE "Config/AppData/MyOwnuValue" -->

<!-- Using variable reference -->
<!--CONFIGVALUE gMyOwnuVal -->

<!-- System variables with formatting -->
<!--CONFIGVALUE "Config/Sys/NetIf/Ethernet0/IPv4/ActiveAddr",16 -->
```

Input Fields

```
<!-- Individual input fields -->
<!--CONFIGINPUT gMyOwnuVal -->
<!--CONFIGINPUT gMyOwnStrVal -->
```

Tables

```
<!-- Individual variable tables -->
<!--CONFIGTABLE gMyOwnStrVal -->
<!--CONFIGTABLE gMyOwnuVal -->

<!-- System configuration sections -->
<!--CONFIGTABLE enet0.ip4 -->
<!--CONFIGTABLE "Config/Reboot" -->
```

12.5.3.3.1.8 Key Features

Read-Only Variables

Some system variables are flagged as read-only (fConfigReadOnly flag). These variables:

- Display their current values
- Cannot be modified through HTML input fields
- Are shown in tables but without editable controls

Reboot Handling

The application includes automatic reboot functionality:

- Uses the system-defined `rebooter` variable
- Displays a "Rebooting" message before restart
- Handles graceful connection closure

Network Integration

- Waits for active network before starting
- Supports HTTP server functionality
- Handles TCP connection management

12.5.3.3.1.9 Dependencies

Required Headers

- `<init.h>` - System initialization
- `<iosys.h>` - I/O system functions
- `<netinterface.h>` - Network interface management
- `<config_obj.h>` - Configuration object system
- `<httppost.h>` - HTTP POST handling
- `<tcp.h>` - TCP connection management

Web Assets

- Bootstrap CSS (`bootstrap.min.css`)
- Bootstrap JavaScript (`bootstrap.bundle.min.js`)
- jQuery (`jquery-3.3.1.min.js`)

12.5.3.3.1.10 Usage

1. **Compile and Deploy** the application to your embedded system
2. **Connect to Network** - The application waits for network connectivity
3. **Access Web Interface** - Navigate to the device's IP address in a web browser
4. **Choose Interface Style:**
 - Click "Config Elements Styled Using Bootstrap and CSS" for modern interface
 - Click "Config Data As It Comes From the Module" for raw data view
5. **Modify Configuration** - Use the forms to change variable values
6. **Save Changes** - Click "SaveConfig" to persist changes

12.5.3.3.1.11 Technical Notes

Form Processing

- POST requests are handled by registered callback functions
- Configuration variables are automatically parsed and validated
- Changes are saved to persistent storage
- Optional reboot functionality for system-level changes

Error Handling

- Invalid configuration values are rejected
- Network connectivity is verified before operation
- TCP connections are properly managed and closed

Performance

- Minimal memory footprint
- Efficient HTML generation
- Responsive web interface

12.5.3.3.1.12 Example Configuration Paths

Application Data

- `Config/AppData/MyOwnValue` - Custom integer value
- `Config/AppData/MyOwnString` - Custom string value

System Configuration

- `Config/Sys/NetIf/Ethernet0/IPv4/ActiveAddr` - Current IP address
- `Config/Reboot` - System reboot flag
- `enet0.ip4` - IPv4 configuration section

12.5.3.4 Custom Web Config

12.5.3.4.1 Custom Web Config System

12.5.3.4.1.1 Overview

This is an advanced embedded web configuration system built for NetBurner devices. It provides a modern, responsive web interface for managing device configuration objects, combining C++ backend configuration management with a JavaScript/jQuery frontend. The system allows users to configure thermostat settings, email notifications, and other application-specific parameters through a web browser.

This example is an advanced version BasicWebConfig example that implements the content with JavaScript and jQuery. It also demonstrates how to store and display custom application data in the AppData section.

Configuration objects are created when the device first boots, then they are stored so they are persist after the device has been power cycled. The web interface adds custom styling beyond the basic server presentation to view and set the values. The

Note

Because we are only processing the config object, we are able to avoid writing a custom POST handler that is used in the other examples. If you include more to your web interface than is provided here, a post handler may be required.

12.5.3.4.1.2 Features

- **Modern Web Interface:** Bootstrap-based responsive design with tabbed navigation
- **Persistent Configuration:** Configuration objects are stored and persist after power cycles
- **Real-time Updates:** Dynamic form generation from JSON configuration data
- **Multiple Configuration Types:** Support for various data types including integers, strings, booleans, IP addresses, and dropdown selections
- **Serial Port Interface:** Command-line interface for local configuration management
- **Automatic Network Setup:** [DHCP](#) support with automatic IP configuration

12.5.3.4.1.3 Architecture

Backend (C++)

The application is built around a hierarchical configuration system using C++ classes:

Configuration Object Types

- [config_obj](#) - Base container object for organizing other configuration objects
- [config_bool](#) - Boolean values
- [config_int](#) - Integer values
- [config_double](#) - Double precision floating point values
- [config_string](#) - String values
- [config_chooser](#) - Multiple choice dropdown options
- [config_pass](#) - Password fields
- [config_IPADDR4](#) - IPv4 addresses
- [config_IPADDR6](#) - IPv6 addresses
- [config_MACADR](#) - MAC addresses

Application Classes

Thermostat1 Class**

- `m_maxTemp` - Maximum temperature setting (default: 100)
- `m_setTemp` - Target temperature setting (default: 30)
- `m_minTemp` - Minimum temperature setting (default: 10)
- `m_powerMax` - Maximum power setting (default: 200)
- `m_active` - Thermostat active status (default: true)

Thermostat2 Class** (extends Thermostat1)

- Inherits all Thermostat1 properties
- `m_tempRec1` - Temperature record 1 (default: 101)
- `m_tempRec2` - Temperature record 2 (default: 102)

EmailNotify Class**

- `m_normal` - Normal notification email address
- `m_emergency` - Emergency notification email address
- `m_choose` - Action chooser (Alert, TurnOff, Crash options)

Frontend (HTML/JavaScript)

The web interface features:

- **Responsive Design:** Bootstrap-based layout that adapts to different screen sizes
- **Tabbed Interface:** Separate tabs for Application Data, Boot Config, and Network Config
- **Dynamic Form Generation:** Forms are automatically generated from JSON configuration data
- **Real-time Updates:** Changes are immediately reflected in the configuration system

12.5.3.4.1.4 Serial Port Commands

The application provides a command-line interface through the serial port:

- `C` - Show complete configuration tree
- `E` - Show email notification settings
- `F` - Show and modify email normal contact address
- `N` - Show IP address for first network interface
- `S` - Show and modify global string setting
- `T` - Show thermostat 1 settings
- `V` - Show and modify global integer value
- `2` - Show thermostat 2 settings
- `6` - Show all interface IP addresses and system time
- `?` - Display command list

12.5.3.4.1.5 Web Interface Usage

1. **Access the Interface:** Navigate to the device's IP address in a web browser
2. **Application Data Tab:** Configure thermostat settings, email notifications, and custom application parameters
3. **Boot Config Tab:** Modify system boot configuration parameters
4. **Network Config Tab:** View and configure network interface settings (some fields are read-only)
5. **Update Records:** Click "Update Record" button to save changes to persistent storage

12.5.3.4.1.6 Configuration Data Flow

1. **Initialization:** Configuration objects are created with default values on first boot
2. **Storage:** Configuration data is automatically saved to persistent storage
3. **Web Access:** HTTP server provides JSON endpoints for configuration data
4. **Form Generation:** JavaScript dynamically creates forms based on JSON structure
5. **Updates:** Form submissions update the JSON data and save to persistent storage

12.5.3.4.1.7 Technical Requirements

- NetBurner embedded device with network capability
- HTTP server support
- Serial port interface (115200 baud)
- Bootstrap CSS framework
- jQuery JavaScript library

12.5.3.4.1.8 Network Configuration

The system automatically configures network settings:

- [DHCP](#) client for automatic IP assignment
- HTTP server on port 80
- AutoIP support for link-local addressing
- Gateway and DNS configuration

12.5.3.4.1.9 Security Considerations

- Password fields are automatically detected and masked in the web interface
- Some network configuration fields are read-only to prevent accidental network disconnection
- Configuration changes require explicit user action (Update Record button)

12.5.3.4.1.10 Customization

To extend the system:

1. Create new configuration classes inheriting from appropriate base types
2. Add objects to the global configuration tree
3. Ensure each object has a unique name
4. The web interface will automatically generate forms for new configuration objects

12.5.3.4.1.11 Build Information

The application displays build timestamp and provides network status information including:

- Primary IP address
- AutoIP address
- Gateway address
- System uptime

12.5.3.5 Custom Web System Config

12.5.3.5.1 Custom System Config Server

A NetBurner RTOS-based embedded web application that demonstrates how to customize the device's system configuration server interface.

12.5.3.5.1.1 Overview

This application shows how to create a custom system configuration interface that runs on NetBurner's network port 20034, separate from the main application web interface on port 80. Unlike typical configuration examples that integrate configuration pages into the application's web interface, this example demonstrates how to customize the device's built-in system configuration server with custom branding, logos, and web page content.

Note

You can have both a custom System Configuration Page on port 20034 and a custom Web Configuration Page on port 80 active at the same time.

12.5.3.5.1.2 Features

- **Dual Web Interfaces:**
 - Main application web server on port 80 (based on SimpleHtml)
 - Custom system configuration server on port 20034
- **Custom Branding:** Ability to customize logos and web page content
- **System Configuration:** Access to device configuration through customized interface
- **File Override System:** Uses NetBurner's overload procedure for customization
- **System Diagnostics:** Integrated diagnostic capabilities for system monitoring
- **Network Management:** Automatic network initialization and connection handling
- **Real-Time Operation:** Built on NetBurner RTOS for reliable real-time performance

12.5.3.5.1.3 Hardware Requirements

- NetBurner embedded module/board
- Network connectivity (Ethernet)
- Compatible with NetBurner NNDK (NetBurner Network Development Kit)

12.5.3.5.1.4 Software Dependencies

- NetBurner RTOS
- NetBurner Network Development Kit (NNDK)
- Required NetBurner libraries:
 - `init.h` - System initialization
 - `nbrtos.h` - RTOS functions
 - `system.h` - System utilities

12.5.3.5.1.5 Quick Start

1. **Run as Demo:** The application includes a sample LOGO.gif file and can be built and run immediately to see the customization in action
2. **View Results:**
 - Browse to port 80 for the main application
 - Browse to port 20034 to see the custom system configuration interface
3. **Add Your Customizations:** Replace the sample files with your own branding elements

12.5.3.5.1.6 Application Flow

1. **Initialization:** System and network stack initialization
2. **Diagnostics:** Enables system diagnostic features
3. **HTTP Server:** Starts the web server for configuration access
4. **Network Wait:** Waits up to 5 seconds for active network connection
5. **Status Display:** Shows application name and NNDK revision information
6. **Main Loop:** Continuous operation with 1-second intervals

12.5.3.5.1.7 Customizable System Files

The system configuration interface can be customized by overriding these files located in `\nburn\nbrtos\source`:

- **LOGO.gif** - Logo displayed on the configuration page
- **ROOT.html** - JavaScript root configuration page
- **RAW.html** - Low-level HTML page with links to configuration and UI.html that displays the entire JSON object
- **UI.html** - Displays the entire JSON configuration object

12.5.3.5.1.8 Customization Process

This application uses NetBurner's **overload procedure** for customization:

1. Create the override directory structure: `override\nbrtos\source`
2. Copy the files you want to customize into this directory
3. Modify the files as needed
4. Build the application - your custom files will be compiled into the application

Example: Custom Logo

A sample LOGO.gif file is provided in the application's `src` folder. To use it:

1. Copy the LOGO.gif to `override\nbrtos\source\` folder
2. Build and run the application
3. View the custom logo on port 20034

12.5.3.5.1.9 Usage

1. **Build and Deploy:** Build and deploy the application to your NetBurner device
2. **Main Application:** Access the basic web page on port 80
3. **System Configuration:** Access the custom configuration interface on port 20034
4. **Customization:** The system configuration page will display your custom modifications

12.5.3.5.1.10 Port Configuration

- **Port 80:** Main application web interface (SimpleHtml-based)
- **Port 20034:** Custom system configuration server

12.5.3.5.1.11 Configuration

The application automatically:

- Initializes network connectivity
- Starts HTTP services on default ports
- Enables diagnostic capabilities

12.5.3.5.1.12 Development Notes

- **Production Deployment:** Consider removing `EnableSystemDiagnostics()` for production builds
- **Network Timeout:** 5-second network connection timeout is configured
- **Main Loop:** 1-second delay in main loop for efficient resource usage
- **File Compilation:** Logo and HTML files are converted to .cpp files and compiled into the application
- **Branding Strategy:** You can combine both custom application and custom system configuration for complete product branding
- **Demo Ready:** The application includes a sample LOGO.gif and can be run as-is to demonstrate customization

12.5.3.5.1.13 Build Information

The application displays:

- Application name: "Custom System Config Server"
- NNDK revision information at startup

12.5.3.6 HTML Only Configuration

12.5.3.6.1 HTML Only Configuration Application

12.5.3.6.1.1 Overview

This is a simple web-based configuration application that provides an HTML interface for configuring network settings and system parameters. The application runs on port 80 and allows users to modify configuration values through a web form interface.

12.5.3.6.1.2 Features

- **Web-based Configuration:** Simple HTML form interface for configuration management
- **Network Configuration:** Supports IPv4 network settings (enet0.ip4)
- **Reboot Control:** Checkbox option to reboot system after updating configuration values
- **Styled Interface:** Clean, professional styling with a modern table design
- **Remote Console:** Enables remote console access for debugging and monitoring

12.5.3.6.1.3 File Structure

```
main.cpp      # Main application entry point
index.html    # Configuration form interface
style.css     # Styling for the web interface
```

12.5.3.6.1.4 Application Components

main.cpp

The main application file that:

- Initializes the network stack
- Enables system diagnostics
- Starts the HTTP server on port 80
- Waits for active network connection
- Enables remote console access
- Implements the HTML post configuration handler

index.html

The web interface that provides:

- Configuration form with POST method to `postcfg.html`
- Table-based layout for configuration parameters
- Network IP configuration section (enet0.ip4)
- Reboot option checkbox
- Submit button to save configuration

style.css

Styling that includes:

- Clean, modern table design with shadow effects
- Alternating row colors for better readability
- Green accent color (#009879) for headers and borders
- Responsive font sizing and spacing
- Active row highlighting capability

12.5.3.6.1.5 Configuration Options

The application currently supports:

1. **Network Configuration** (`enet0.ip4`)
 - IPv4 network settings for ethernet interface
2. **System Reboot** (`rebooter`)
 - Checkbox to enable automatic reboot after configuration changes

12.5.3.6.1.6 Technical Details

- **Framework:** NetBurner NNDK (NetBurner Network Development Kit)
- **Web Server:** Built-in HTTP server on port 80
- **Configuration Handler:** `HtmlPostConfigVariableHandler` class
- **Network Stack:** NetBurner real-time network stack
- **System:** Real-time operating system (NBRTOS)

12.5.3.6.1.7 Usage

1. Connect to the device's IP address via web browser
2. Navigate to the configuration page (`index.html`)
3. Modify the desired configuration parameters
4. Check the reboot option if system restart is needed
5. Click "SaveConfig" to apply changes

12.5.3.6.1.8 Development Notes

- The application uses NetBurner's configuration system with `<!--CONFIGTABLE-->` directives
- Additional configuration objects can be added by including appropriate header files
- System diagnostics are enabled (should be removed for production)
- The application includes a 5-second timeout for network initialization

12.5.3.6.1.9 Customization

To add additional configuration parameters:

1. Include the appropriate header files in `index.html`
2. Add new `<!--CONFIGTABLE-->` entries for your configuration objects
3. Rebuild and deploy the application

12.5.3.6.1.10 Dependencies

- NetBurner NNDK framework
- Standard NetBurner libraries ([init.h](#), [nbrtos.h](#), [system.h](#), etc.)
- HTTP POST handling library
- Remote console library

12.5.4 Detached Config Object

This example demonstrates how to create JSON configuration objects (data or structures) in a "detached" mode, meaning they do not appear in the web page configuration screen, nor are they saved, but are exposed via individual web pages. In Detached mode they are created at boot time and can be modified and used by the application. The example will:

- Create a number of configuration object classes
- Use the classes to create Detached configuration objects
- For reference, shows how they would be exposed in the normal configuration tree
- Expose those objects as web pages containing the JSON information

For example, if your device IP address is 10.1.1.10, opening a web browser and typing "10.1.1.10/set.json" will display that object and it's data:

```
ctrl_set
Hset
  D 0
  I 0.0001
  P 1
Vset
  D 0
  I 0.0001
  P 1
rev 0
```

12.5.4.1 Detached Config Example

12.5.4.1.1 Overview

This NetBurner application demonstrates how to create JSON configuration objects in "detached" mode. In detached mode, configuration objects are created at boot time and can be modified and used by the application, but they do not appear in the Configuration Server web interface and may or may not be saved to persistent storage depending on the application requirements. However, they can, and usually are, used by individual web pages.

12.5.4.1.2 Features

- Creates multiple configuration object classes with different data types
- Demonstrates both detached and normal configuration modes
- Exposes configuration objects as individual JSON web pages
- Includes examples of inheritance between configuration classes

12.5.4.1.3 Configuration Classes

12.5.4.1.3.1 TempRange

Base class for temperature-related configuration:

- `MaxTemp`: Maximum temperature (integer, default: 100)
- `SetTemp`: Set temperature (integer, default: 30)
- `MinTemp`: Minimum temperature (integer, default: 10)

12.5.4.1.3.2 Thermostat

Extends `TempRange` with additional thermostat-specific settings:

- `Active`: Enable/disable thermostat (boolean, default: true)
- `Location`: Physical location (string, default: "Warehouse 1")
- `TempScale`: Temperature scale chooser (Fahrenheit/Celsius/Kelvin, default: Fahrenheit)

12.5.4.1.3.3 PIDSet

PID controller configuration:

- **P**: Proportional term (double, default: 1.0)
- **I**: Integral term (double, default: 0.0001)
- **D**: Differential term (double, default: 0.0)

12.5.4.1.3.4 ControlSet

Container for PID control settings:

- **Hset**: Horizontal PID settings
- **Vset**: Vertical PID settings
- **rev**: Settings revision number (integer, default: 0)

12.5.4.1.4 Configuration Modes

The application supports two modes controlled by the `DETACHED_MODE` preprocessor definition:

12.5.4.1.4.1 Detached Mode (Default)

- Configuration objects are created with the `detached` parameter
- Objects are not exposed in the configuration tree
- Objects are not saved to persistent storage
- Objects are accessible via individual web pages

12.5.4.1.4.2 Normal Mode

- Configuration objects are created under the `appdata` parent
- Objects appear in the standard configuration tree
- Objects are saved to persistent storage

12.5.4.1.5 Web Interface

The application exposes three JSON endpoints:

- `<device_ip>/set.json` - Displays the `ControlSet` object
- `<device_ip>/thermo.json` - Displays the `Thermostat` object
- `<device_ip>/anint.json` - Displays a simple integer configuration

12.5.4.1.5.1 Example JSON Output

Accessing `<device_ip>/set.json` displays:

```
ctrl_set
  Hset
    D    0
    I    0.0001
    P    1
  Vset
    D    0
    I    0.0001
    P    1
  rev   0
```

12.5.4.1.6 Usage

1. Build and deploy the application to your NetBurner device
2. Wait for network initialization (up to 5 seconds)
3. Open a web browser and navigate to:
 - http://<device_ip>/set.json
 - http://<device_ip>/thermo.json
 - http://<device_ip>/anint.json

12.5.4.1.7 Application Structure

main.cpp	- Main application code with configuration classes
ReadMe.txt	- Original documentation
index.html	- Basic web page with NetBurner branding

12.5.4.1.8 Build Requirements

- NetBurner development environment
- Standard NetBurner libraries:
 - [nbrtos.h](#)
 - [http.h](#)
 - [init.h](#)
 - [netinterface.h](#)

12.5.4.1.9 Key Concepts Demonstrated

1. **Configuration Object Classes:** How to create custom configuration classes that inherit from [config_obj](#)
2. **Detached Configuration:** Creating configuration objects that exist only in memory
3. **Web Exposure:** Using `HtmlConfigExposer` to make configuration objects accessible via HTTP
4. **Inheritance:** How configuration classes can extend other configuration classes
5. **Multiple Data Types:** Support for integers, doubles, booleans, strings, and choosers

12.5.4.1.10 Notes

- The `ConfigEndMarker` must be placed after all configuration members in each class
- Configuration objects can be nested within other configuration objects
- The application name is set to "Detached Config Example"
- Network initialization waits up to 5 seconds before starting the HTTP server

12.5.5 Change Notify

12.5.5.1 Config Notify Application

12.5.5.1.1 Overview

This application demonstrates how to use configuration objects with change notification callbacks in an embedded system. The program creates basic configuration variables and registers callbacks to be notified when these variables change.

12.5.5.1.2 Features

- **Configuration Management:** Creates and manages different types of configuration variables (string, integer, unsigned integer)
- **Change Notifications:** Implements callback notifications when configuration values are modified
- **Interactive Menu System:** Provides a command-line interface for viewing and modifying configuration values
- **Dynamic Configuration:** Supports creating and deleting configuration variables at runtime
- **Network Integration:** Displays network interface information on startup

12.5.5.1.3 Configuration Variables

The application creates three main configuration objects:

1. **gSingleString** - A string configuration variable with default value "SName"
2. **gMyOwnVal** - An integer configuration variable with default value 199
3. **gMyOwnuVal** - An unsigned integer configuration variable with default value 299

12.5.5.1.4 Command Interface

The application provides an interactive menu system with the following commands:

- **C** - Show Config Tree: Displays the complete configuration tree
- **S** - Show and Set gSingleString: View and modify the string configuration
- **V** - Show and Set gMyOwnVal: View and modify the integer configuration
- **U** - Show and Set gMyOwnuVal: View and modify the unsigned integer configuration
- **N** - New/Create dynamic variable: Creates a new dynamic configuration variable
- **D** - Delete dynamic variable: Removes the dynamic configuration variable
- **Q** - Query dynamic variable: Displays the value of the dynamic variable
- **?** - Show Command List: Displays all available commands

12.5.5.1.5 Notification System

The application implements a notification callback system that:

- Uses a semaphore-based notification mechanism
- Registers callbacks for all configuration variables
- Provides real-time notification when any monitored configuration value changes
- Displays a message when a configuration change is detected

12.5.5.1.6 Technical Implementation

12.5.5.1.6.1 Key Components

- **Configuration Objects:** Uses `config_string`, `config_int`, and `config_uint` types
- **Callback System:** Implements `ConfigNotificationCallBack` for change notifications
- **Semaphore Synchronization:** Uses `OS_SEM` for thread-safe notification handling
- **Storage Management:** Automatically saves configuration changes to persistent storage

12.5.5.1.6.2 Memory Management

- Static configuration objects are created at global scope
- Dynamic configuration objects can be created and destroyed at runtime
- Non-static objects require calling `FixNonStaticObject()` method

12.5.5.1.6.3 Network Information

On startup, the application displays:

- IP address of the first network interface
- AutoIP address
- Gateway address
- Configuration space usage statistics

12.5.5.1.7 Usage

1. **Startup:** The application initializes network interfaces and displays system information
2. **Command Entry:** Enter single character commands to interact with the configuration system
3. **Value Modification:** Follow prompts to enter new values for configuration variables
4. **Notification Monitoring:** The system automatically notifies when values change

12.5.5.1.8 Build Information

The application displays build timestamp information on startup, showing when the code was compiled.

12.5.5.1.9 Example Workflow

1. Start application and view initial configuration
2. Use 'C' command to see the complete config tree
3. Use 'S', 'V', or 'U' commands to modify configuration values
4. Observe notification messages when values change
5. Use 'N' to create dynamic variables and 'D' to delete them
6. Use 'Q' to query dynamic variable values

12.5.6 Config Class Receive Notification

12.5.6.1 Config Notify Class Example

12.5.6.1.1 Overview

This NetBurner application demonstrates the use of the config system with notification capabilities. The config system stores both system-defined boot and interface information (such as default BaudRate and current IP settings) and provides a framework for users to define and store their own persistent variables.

This example extends the basic ConfigClass functionality by implementing a notification system that alerts when any configuration variables change.

12.5.6.1.2 Features

- **Persistent Configuration Storage:** Variables are automatically saved to non-volatile storage
- **Change Notifications:** Automatic notifications when configuration values are modified
- **Interactive Command Interface:** User-friendly commands to view and modify settings
- **Hierarchical Configuration:** Demonstrates inheritance with TempRange and Thermostat classes

12.5.6.1.3 Application Structure

12.5.6.1.3.1 Classes

TempRange

A base configuration class that defines temperature-related settings:

- `m_maxTemp`: Maximum temperature threshold (default: 100)
- `m_setTemp`: Set point temperature (default: 30)
- `m_minTemp`: Minimum temperature threshold (default: 10)

Thermostat

Extends `TempRange` and implements `ConfigNotificationObject`:

- Inherits all `TempRange` variables
- `m_tempRec1`: Temperature record 1 (default: 101)
- `m_tempRec2`: Temperature record 2 (default: 102)
- `m_active`: Active status flag (default: true)
- `m_location`: Thermostat location string (default: "Warehouse 1")
- `m_tempScale`: Temperature scale chooser (Fahrenheit/Celsius/Kelvin)

12.5.6.1.3.2 Key Features

Notification System

The `Thermostat` class implements a `Notify()` method that is automatically called whenever any of its configuration variables change. This provides real-time awareness of configuration modifications.

Interactive Commands

The application provides a command-line interface with the following options:

- `C` - Show complete configuration tree
- `L` - View and set thermostat location
- `M` - View and set maximum temperature
- `N` - View and set minimum temperature
- `S` - View and set target temperature
- `T` - Display recorded temperatures
- `?` - Show command list

12.5.6.1.4 Usage

1. **Startup**: The application initializes the network interface and displays current IP settings
2. **Interactive Mode**: Enter commands to view or modify configuration values
3. **Automatic Saving**: All changes are automatically saved to persistent storage
4. **Notifications**: The system prints a message whenever configuration values change

12.5.6.1.5 Network Information

On startup, the application displays:

- Current IP address
- AutoIP address
- Gateway address

12.5.6.1.6 Technical Implementation

12.5.6.1.6.1 Configuration Storage

- Uses NetBurner's config system for persistent storage
- Automatic serialization/deserialization of configuration objects
- Thread-safe access to configuration variables

12.5.6.1.6.2 Notification Mechanism

- Implements `ConfigNotificationObject` interface
- Automatic registration for change notifications
- Custom `Notify()` method for handling configuration changes

12.5.6.1.6.3 Input Processing

- Non-blocking character input using `charavail()`
- User-friendly prompts for value modification
- Automatic type conversion and validation

12.5.6.1.7 Building and Running

This application is designed for NetBurner embedded systems and requires:

- NetBurner SDK and toolchain
- Network-capable NetBurner hardware platform
- Serial terminal for user interaction

The application runs continuously, processing user commands and maintaining configuration state across power cycles.

12.5.6.1.8 Example Output

```
Config Tree Demo built at 12:34:56 on Jan 01 2025
'?' for commands
IP:          192.168.1.100
AutoIP :     169.254.1.100
Gateway:     192.168.1.1

Command List
-----
C) Show Config Tree
L) Show and Set Thermostat Location
M) Show and Set Thermostat Max Temp
N) Show and Set Thermostat Min Temp
S) Show and Set Thermostat Set Temp
T) Show Thermostat Recorded Temps
?) Show Command List
```

12.5.6.1.9 Notes

- All configuration objects should be created at global scope
- Each configuration member must have a unique name within the config tree
- The `ConfigEndMarker` is required after the last data member in each config class
- Changes trigger both the notification system and automatic storage saves

12.6 DNS Device Name

12.6.1 DNS Device Name Application

12.6.1.1 Overview

This NetBurner application demonstrates how to programmatically set a device name that can be used for DNS resolution on a network. The device name is automatically generated using the device's MAC address and stored in the flash configuration.

The device name is stored in the device's Flash System Configuration. There are two ways to give your device a name:

1. Use the configuration web interface, which is your device's `<device ip address>:20034`.
2. You can change the configuration flash setting in your application. You will then need to reboot the device for the name to take effect.

This example demonstrates how to change the device name in an application.

Note

This is not the same as a .local or WINS device name. The network must have a DNS and [DHCP](#) server.

12.6.1.2 Features

- Automatically generates a unique device name based on MAC address
- Stores device name in flash configuration for persistence
- Provides network identification for [DHCP](#) and DNS services
- Simple web interface access for manual configuration

12.6.1.3 How It Works

The application performs the following operations:

1. **Device Initialization:** Sets up the network interface and enables HTTP services
2. **MAC Address Retrieval:** Gets the MAC address from the first available network interface (typically Ethernet0)
3. **Name Generation:** Creates a device name using the format `NetBurner-XXXX` where XXXX represents the last two octets of the MAC address in hexadecimal
4. **Configuration Storage:** Compares the generated name with the current flash configuration and updates it if different
5. **Persistence:** Saves the new configuration to storage and notifies that a reboot is required

12.6.1.4 Device Naming

12.6.1.4.1 Automatic Naming Convention

- Format: `NetBurner-XXXX`
- Example: `NetBurner-848F` (where 848F are the last two bytes of the MAC address)

12.6.1.4.2 Manual Configuration Options

You can also set the device name manually through:

1. **Web Interface:** Access `<device_ip_address>:20034` in your browser
2. **Application Code:** Modify the flash configuration programmatically (as demonstrated in this example)

12.6.1.5 Network Integration

12.6.1.5.1 DHCP Integration

When the device requests an IP address from a [DHCP](#) server, it sends:

- MAC address
- Device name (optional, but supported)

The [DHCP](#) server maintains a table with MAC address, IP address, and device name for each device.

12.6.1.5.2 DNS Resolution

For the device name to be resolvable via DNS:

- **Windows Server:** Configure [DHCP](#) and DNS server integration through dialog settings
- **Linux/Unix:** May require additional configuration for DHCP-DNS integration
- **Without DNS Server:** Manually add entries to the `lmhosts` file on each PC

12.6.1.5.3 Windows WINS Protocol

Windows machines can use the proprietary WINS protocol for name resolution, where one Windows PC maintains the table of MAC addresses, IP addresses, and names.

12.6.1.6 Sample Output

```
Interface 1, Ethernet0, MAC:00:03:f4:0b:84:8f
Current device name from flash configuration: "MODM7AE70SX-848F"
Assigning device name: NetBurner-848F
Reboot device for new device name to take effect

Primary network interface configured for DHCP
MAC Address = 00:03:f4:0b:84:8f
Type "A" to abort boot

Interface 1, Ethernet0, MAC:00:03:f4:0b:84:8f
Current device name from flash configuration: "NetBurner-848F"
New name already matches configuration
```

12.6.1.7 Key Functions

12.6.1.7.1 CreateDnsDeviceName(char *deviceName, uint32_t maxLen)

- Retrieves the first network interface information
- Displays current configuration
- Generates new device name from MAC address
- Updates flash configuration if name has changed
- Saves configuration to storage

12.6.1.7.2 UserMain(void *pd)

- Initializes the system and network
- Enables HTTP services
- Waits for network connectivity
- Calls the device naming function
- Enters main application loop

12.6.1.8 Important Notes

- **Reboot Required:** After changing the device name, a reboot is necessary for the new name to take effect
- **Name Length:** Maximum device name length is 32 characters
- **Network Dependencies:** Requires active network connection and [DHCP](#) configuration
- **Persistence:** Device name is stored in flash memory and survives power cycles

12.6.1.9 Building and Deployment

This application is designed for NetBurner embedded systems and requires:

- NetBurner development environment
- Compatible NetBurner hardware module
- Network connection with [DHCP](#) server

12.6.1.10 Use Cases

- Network device identification and management
- Simplified device access by name instead of IP address
- Integration with existing DNS infrastructure
- Automated device provisioning in larger networks

12.6.1.11 Additional Information on DNS Naming

When a network device (such as the NetBurner module) requests an IP address from a [DHCP](#) Server, the device sends it MAC address and an optional device name (this is supported by your NetBurner device). The [DHCP](#) Server supplies the dynamic IP address and maintains a table containing the MAC address, IP address and device name for each device that has accepted a [DHCP](#) lease.

The purpose of a DNS server is to convert a name into an IP address. To do this it must get the information from your [DHCP](#) Server. On windows server platforms that run both a [DHCP](#) server and DNS server, this can be configured in a dialog box. On Linux it may be more complicated. Basically, every time the [DHCP](#) server gives out a new dynamic IP address it updates the DNS server. If you are not using [DHCP](#), and are assigning static IP addresses, then you must update the DNS server with the entries manually (i.e. name and IP address).

If you do not have a DNS server, you can modify the `lmhosts` file on your PC to add the IP address and name of each network device you want to address by name. You would need to do this on every PC that wants to use the name.

Windows has a proprietary protocol called WINS (only Microsoft products can use it). Windows machines can have a name, and the protocol enables windows machines to talk to one another. This is achieved by having one windows PC maintain the table of MAC address, IP address and names.

12.7 DHCP

DHCP Examples:

- [DHCP Client - Change IP Address](#)
- [DHCP Client - Test DHCP](#)
- [DHCP Server \(DHCPD\)](#)
- [DHCP Client - Change IP Address Via Webpage](#)

12.7.1 DHCP Client - Change IP Address

Demonstrate how to view IP address and change between [DHCP](#) and static IPv4 IP settings.

12.7.1.1 DHCP Client - Change IP Address Application

12.7.1.1.1 Overview

This application demonstrates how to dynamically switch between [DHCP](#) and static IPv4 IP address configurations on a NetBurner embedded device. It provides an interactive menu system through the serial port that allows users to view current network settings and change between addressing modes without requiring a system reboot.

12.7.1.1.2 Features

- **Interactive Serial Menu:** Control the application through a simple command-line interface
- **Real-time IP Configuration:** Switch between [DHCP](#) and static IP modes during runtime
- **DHCP State Monitoring:** Display current [DHCP](#) client state and lease information
- **Network Status Display:** View comprehensive IPv4 settings including IP address, subnet mask, gateway, and DNS servers
- **Optional WiFi Support:** Configurable WiFi functionality (disabled by default)
- **Web Server Integration:** Built-in HTTP server for web-based management

12.7.1.1.3 Supported Address Modes

The application supports the following IPv4 addressing modes:

- **DHCP:** Automatic IP address assignment from [DHCP](#) server
- **DHCP w/ Fallback:** [DHCP](#) with fallback to static configuration if [DHCP](#) fails
- **Static:** Manual IP address configuration
- **Disabled:** Network interface disabled

12.7.1.1.4 Menu Options

The application provides a simple menu system with the following options:

1. **Display Runtime Settings:** Shows current IPv4 configuration, [DHCP](#) state, and network interface information
2. **Change to Static Address Mode:** Switches to static IP configuration using settings from the configuration record
3. **Change to [DHCP](#) Address Mode:** Enables [DHCP](#) client and obtains IP address automatically

12.7.1.1.5 Configuration Methods

12.7.1.1.5.1 Method 1: Using Configuration Record (Default)

The application reads static IP settings from the device's configuration record stored in flash memory. This is the recommended approach as it allows settings to persist across reboots.

12.7.1.1.5.2 Method 2: Application-Defined Settings (Optional)

Alternatively, you can define static IP settings directly in the application code by:

1. Uncommenting the `#define` statements for static IP values
2. Uncommenting the `AssignStaticIpv4Values()` function call
3. Modifying the IP addresses to match your network environment

```
#define STATIC_IP "10.1.1.222"  
#define STATIC_MASK "255.255.255.0"  
#define STATIC_GATE "10.1.1.1"  
#define STATIC_DNS1 "10.1.1.1"  
#define STATIC_DNS2 "8.8.8.8"
```

12.7.1.1.6 Switching Between Modes

12.7.1.1.6.1 Changing from DHCP to Static

1. Static settings are read from the configuration record
2. Address mode is set to "Static"
3. Configuration is saved to flash memory
4. Runtime settings are updated immediately (no reboot required)

12.7.1.1.6.2 Changing from Static to DHCP

1. Static settings can remain intact (they won't affect [DHCP](#) operation)
2. Address mode is set to "DHCP" or "DHCP w/ Fallback"
3. Configuration is saved to flash memory
4. [DHCP](#) client is restarted to obtain new lease

12.7.1.1.7 DHCP State Information

The application monitors and displays the following [DHCP](#) client states:

- **DHCP not started:** System initialization incomplete
- **DHCP Discover:** Searching for [DHCP](#) servers
- **DHCP Offer:** Received offer from [DHCP](#) server
- **DHCP Acknowledge:** Accepted [DHCP](#) offer
- **DHCP Complete:** Successfully obtained valid lease
- **DHCP Renew/Rebind:** Lease renewal process
- **DHCP Failed:** Could not obtain [DHCP](#) lease

12.7.1.1.8 Optional Features

12.7.1.1.8.1 WiFi Support

WiFi functionality can be enabled by uncommenting the `#define USE_WIFI (1)` directive. This requires:

- WiFi driver initialization
- QSPI interface configuration
- Appropriate WiFi hardware support

12.7.1.1.8.2 System Diagnostics

The application includes system diagnostics for debugging purposes. For production deployments, consider removing the `EnableSystemDiagnostics()` call.

12.7.1.1.9 Network Interface Information

The application displays comprehensive network information including:

- Device name and application name
- Current addressing mode (DHCP/Static)
- IPv4 addresses (current and configured)
- Subnet mask
- Default gateway
- Primary and secondary DNS servers
- Link status and connectivity information

12.7.1.1.10 Usage Instructions

1. **Build and Deploy:** Compile the application and deploy to your NetBurner device
2. **Connect Serial Terminal:** Connect to the device's serial port (typically 115200 baud)
3. **Wait for Network:** The application waits up to 5 seconds for network initialization
4. **Use Menu System:** Enter menu options (1, 2, or 3) to interact with the application
5. **Monitor Changes:** View real-time network configuration changes

12.7.1.1.11 Technical Details

- **NNDK Compatibility:** Built for NetBurner Network Development Kit
- **Memory Management:** Uses configuration server for persistent settings storage
- **Network Stack:** Integrates with NetBurner's TCP/IP stack
- **Threading:** Runs in NetBurner's multi-tasking environment
- **Error Handling:** Includes proper error checking and state management

12.7.1.1.12 Development Notes

- The application uses NetBurner-specific APIs and data structures
- Format warnings for `HI` and `I` parameters are intentionally suppressed
- The code follows NetBurner's naming conventions and coding standards
- IPv6 support is conditionally compiled based on `IPV6` preprocessor definition

12.7.1.1.13 Prerequisites

- NetBurner development board with Ethernet interface
- NetBurner Network Development Kit (NNDK)
- Serial terminal application for user interaction
- Network infrastructure with [DHCP](#) server (for [DHCP](#) mode testing)

This application serves as both a practical network management tool and a comprehensive example of NetBurner's networking capabilities for embedded developers.

12.7.2 DHCP Client - Test DHCP

12.7.2.1 DHCP Test Application

12.7.2.1.1 Overview

This application demonstrates advanced [DHCP](#) (Dynamic Host Configuration Protocol) client functionality by providing direct control and monitoring of [DHCP](#) services. Unlike simple applications that use the `GetDhcpAddress()` function, this example creates and manages a [DHCP](#) object directly, giving you fine-grained control over the [DHCP](#) process.

12.7.2.1.2 Purpose

The application serves both educational and testing purposes:

- **Educational:** Shows internal [DHCP](#) workings by displaying UDP packet information and [DHCP](#) variables
- **Testing:** Provides interactive control over [DHCP](#) services for debugging and validation
- **Demonstration:** Illustrates how to access and manipulate [DHCP](#) object properties programmatically

12.7.2.1.3 Key Features

12.7.2.1.3.1 DHCP State Monitoring

The application displays comprehensive [DHCP](#) state information including:

- `DHCP_NOTSTARTED` - System not initialized
- `DHCP_DISCOVER` - Discovering [DHCP](#) servers
- `DHCP_OFFER` - Responding to server offers
- `DHCP_ACK` - Acknowledging offers
- `DHCP_INIT` - Reinitializing
- `DHCP_CMPL` - Valid lease obtained
- `DHCP_RENEW` - Lease renewal in progress
- `DHCP_REBIND` - Rebinding after failed renewal
- `DHCP_RELEASE` - Releasing lease
- `SBOOTP_TRANSMITTING/DONE` - BOOTP fallback states
- `DHCP_FAILED` - Failed to obtain lease

12.7.2.1.3.2 Interactive Menu System

The application provides a command-line interface with the following options:

- `**t**` - Display remaining lease time and timing information
- `**s**` - Show detailed [DHCP](#) status (debug mode only)
- `**v**` - Validate current [DHCP](#) lease
- `**1**` - Set system timer to 5 seconds before renew time
- `**2**` - Set system timer to 5 seconds before rebind time
- `**3**` - Set system timer to 5 seconds before lease expiration
- `**4**` - Force [DHCP](#) renewal
- `**5**` - Force [DHCP](#) rebinding
- `**a**` - Show [ARP](#) table

12.7.2.1.3.3 Advanced Timing Control

The application allows manipulation of [DHCP](#) timing for testing scenarios:

- Simulate approaching renewal deadlines
- Test rebinding behavior
- Verify lease expiration handling
- Force state transitions for validation

12.7.2.1.4 Technical Implementation

12.7.2.1.4.1 Architecture

- Creates a global [DHCP](#) object variable
- Uses semaphore monitoring for [DHCP](#) completion
- Accesses private class variables for detailed debugging (educational purposes only)
- Integrates with the NetBurner network stack

12.7.2.1.4.2 Debug Information

When debug mode is enabled, the application displays:

- UDP packet details
- Internal [DHCP](#) variables
- Low-level state information
- Timing and lease parameters

12.7.2.1.5 Setup and Configuration

12.7.2.1.5.1 Debug Mode Setup

To enable detailed debugging information:

1. Edit `\nburn\system\dhcpc.cpp`
2. Uncomment the line: `#define DHCP_DEBUG 1`
3. Rebuild the application

Warning

Comment out `#define DHCP_DEBUG 1` when finished with testing. This should not be enabled in production applications.

12.7.2.1.5.2 Build Configuration

- The application uses conditional compilation with `DHCPTTEST_DEBUG`
- System diagnostics are enabled for development (consider removing for production)
- Serial output is directed to UART0 by default (viewable with MTTTY)

12.7.2.1.6 Usage

1. Flash the application to your NetBurner device
2. Connect via serial terminal (MTTTY recommended)
3. The application will automatically start [DHCP](#) and display the menu
4. Use the menu commands to interact with and test [DHCP](#) functionality
5. Monitor the output for state changes and debugging information

12.7.2.1.7 Educational Value

This application is designed to teach:

- [DHCP](#) protocol states and transitions
- Lease timing and renewal processes
- Error handling and fallback mechanisms
- Network debugging techniques
- Embedded network programming patterns

For comprehensive [DHCP](#) protocol information, refer to RFC 2131.

12.7.2.1.8 Development Notes

- Application name appears as "DHCPTTest" in IPSetup
- Uses NetBurner-specific APIs and classes
- Demonstrates both high-level and low-level [DHCP](#) access patterns
- Shows proper integration with the NetBurner timeout management system

12.7.2.1.9 Warning

This example accesses private class variables of the [DHCP](#) object for educational purposes. This approach is **NOT recommended** for production applications and should only be used for learning and debugging.

12.7.2.1.10 Dependencies

- NetBurner RTOS and network stack
- Serial communication support
- [DHCP](#) client library
- [ARP](#) functionality modify the system timer, and display dhcp status.

12.7.3 DHCP Server (DHCPD)

12.7.3.1 DHCP Server Application (DHCPD)

12.7.3.1.1 Overview

This NetBurner application demonstrates how to create and deploy a [DHCP](#) (Dynamic Host Configuration Protocol) Server on your NetBurner device. The application provides automatic IP address assignment to devices on your network, making it useful for creating standalone networks or testing environments.

12.7.3.1.2 Features

- **DHCP Server Implementation:** Creates a fully functional [DHCP](#) server that can assign IP addresses to network clients
- **Conflict Detection:** Automatically detects existing [DHCP](#) servers on the network before starting to prevent conflicts
- **Web Interface:** Includes an HTTP server for potential configuration and monitoring
- **Network Diagnostics:** Built-in system diagnostics for troubleshooting
- **Multi-Interface Support:** Can operate on different network interfaces

12.7.3.1.3 How It Works

12.7.3.1.3.1 Startup Sequence

1. **Network Initialization:** The application initializes the network stack and waits for an active network connection
2. **DHCP Server Detection:** Scans the network for existing [DHCP](#) servers to avoid conflicts
3. **User Intervention:** If another [DHCP](#) server is detected, the application prompts the user to resolve the conflict
4. **Server Launch:** Once no conflicts are detected, launches the [DHCP](#) server on the specified interface
5. **Continuous Operation:** Runs indefinitely, serving [DHCP](#) requests

12.7.3.1.3.2 Key Components

- **FindOtherDHCPServer():** Implements network scanning to detect existing [DHCP](#) servers
 - Sends [DHCP](#) DISCOVER messages to probe for active servers
 - Uses a 5-second timeout to wait for responses
 - Returns true if another server is found, false otherwise
- **AddMyDHCPServer():** Initializes and starts the [DHCP](#) server functionality
 - Operates on the specified network interface
 - Handles [DHCP](#) client requests and IP address allocation
- **DhcpServCheck():** Callback function for processing [DHCP](#) responses during server detection

12.7.3.1.4 Usage

12.7.3.1.4.1 Compilation and Deployment

1. Ensure you have the NetBurner development environment set up
2. Include the necessary NetBurner libraries:
 - `dhcpd.h` - DHCP server functionality
 - `dhcpinternals.h` - Internal DHCP structures
 - `netinterface.h` - Network interface management
 - Standard NetBurner system libraries
3. Compile and flash to your NetBurner device

12.7.3.1.4.2 Operation

1. **Power on** your NetBurner device
2. The application will automatically:
 - Initialize network services
 - Check for existing DHCP servers
 - Prompt you if conflicts are detected
3. **If conflicts exist:** Disable other DHCP servers on the network and press any key to retry
4. **Once running:** The device will serve DHCP requests to network clients

12.7.3.1.4.3 Network Configuration

- The DHCP server will operate on the first available network interface by default
- Client devices connecting to the network will automatically receive IP addresses
- The web server runs on port 80 for potential configuration access

12.7.3.1.5 Important Considerations

12.7.3.1.5.1 Production Deployment

- **Remove diagnostics:** The code includes `EnableSystemDiagnostics()` which should be removed for production use
- **Security:** Consider implementing access controls if deploying in production environments
- **Network planning:** Ensure the IP address pool doesn't conflict with other network infrastructure

12.7.3.1.5.2 Network Conflicts

- **Multiple DHCP servers** on the same network can cause IP address conflicts
- Always verify no other DHCP servers are active before deployment
- Common sources of DHCP servers include routers, other embedded devices, and server systems

12.7.3.1.6 Dependencies

- NetBurner NNDK (NetBurner Network Development Kit)
- Custom memory allocation module (`MyAlloc.h`)
- Standard NetBurner networking libraries

12.7.3.1.7 Technical Details

- **Language:** C++
- **Platform:** NetBurner embedded devices
- **Threading:** Uses NetBurner RTOS (Real-Time Operating System)
- **Network Stack:** NetBurner TCP/IP stack
- **Memory Management:** Custom buffer management for DHCP packets

12.7.3.1.8 Troubleshooting

12.7.3.1.8.1 Common Issues

1. **There is an active DHCP server on this net**
 - Another DHCP server is detected on the network
 - Disable other DHCP servers or move to an isolated network segment
2. **Network timeout during startup**
 - Check network connectivity and cable connections
 - Verify network interface configuration
3. **Clients not receiving IP addresses**
 - Confirm DHCP server started successfully
 - Check network segmentation and VLAN configuration
 - Verify firewall settings aren't blocking DHCP traffic

12.7.3.1.8.2 Debug Information

The application outputs status messages to the console including:

- Application name and NNDK revision information
- DHCP server conflict detection results
- Startup progress indicators

12.7.4 DHCP Client - Change IP Address Via Webpage

12.7.4.1 DHCP Client - Change IP Address Via Webpage

This application demonstrates how to view and change IPv4 IP settings between DHCP and static configurations using both a web interface and serial console interface.

12.7.4.1.1 Overview

The application allows users to:

- View current runtime IP settings and DHCP status
- Switch between DHCP and static IP address modes
- Configure IP settings through a web-based interface
- Monitor network interface status and configuration
- Handle multiple network interfaces (Ethernet, WiFi)

12.7.4.1.2 Key Features

12.7.4.1.2.1 Web Interface

- **Interactive Configuration:** Web-based form (`changeip.html`) for modifying IP settings
- **Real-time Status:** Display current IP addresses, [DHCP](#) state, and interface information
- **Automatic Redirection:** After configuration changes, the web interface automatically redirects to the new IP address
- **Multi-interface Support:** Handles multiple network interfaces simultaneously

12.7.4.1.2.2 Serial Console Interface

- **Runtime Settings Display:** View current IPv4 settings for all interfaces
- **[DHCP](#) State Monitoring:** Real-time [DHCP](#) client status information
- **Manual Mode Switching:** Change between [DHCP](#) and static modes via console commands
- **Network Link Status:** Monitor physical network connection status

12.7.4.1.2.3 DHCP Client Features

- **Full [DHCP](#) State Machine:** Complete implementation with all [DHCP](#) states
- **[DHCP](#) with Fallback:** Option to fall back to static IP if [DHCP](#) fails
- **Lease Management:** Automatic [DHCP](#) lease renewal and rebinding
- **BOOTP Support:** Fallback to BOOTP protocol when [DHCP](#) is unavailable

12.7.4.1.3 File Structure

- `main.cpp`: Main application with serial console interface and core networking logic
- `web.cpp`: Web server implementation with form handling and IP configuration management
- `ip_util.h`: Utility functions for IP address display and network operations

12.7.4.1.4 Configuration Modes

12.7.4.1.4.1 DHCP Mode

- Automatically obtains IP address from [DHCP](#) server
- Handles lease renewal and network changes
- Supports [DHCP](#) with fallback to static configuration

12.7.4.1.4.2 Static Mode

- Uses pre-configured static IP settings
- Settings stored in configuration flash memory
- Can be configured via web interface or application code

12.7.4.1.4.3 Runtime vs. Persistent Settings

- **Runtime Changes:** Immediate effect without reboot
- **Persistent Changes:** Saved to flash memory and survive reboots
- **Configuration Storage:** Uses built-in configuration server for settings persistence

12.7.4.1.5 Usage Instructions

12.7.4.1.5.1 Changing from DHCP to Static:

1. Access the web interface or serial console
2. Read current static settings from configuration record
3. Set address mode to "Static"
4. Save configuration to flash memory
5. Choose to reboot or apply runtime settings immediately

12.7.4.1.5.2 Changing from Static to DHCP:

1. Access the web interface or serial console
2. Set address mode to "DHCP" or "DHCP w Fallback"
3. Save configuration to flash memory
4. Choose to reboot or restart [DHCP](#) client immediately

12.7.4.1.5.3 Serial Console Commands:

- **1:** Display current runtime IPv4 settings
- **2:** Change to static address mode (without reboot)
- **3:** Change to [DHCP](#) address mode (without reboot)

12.7.4.1.6 Technical Implementation

12.7.4.1.6.1 DHCP State Machine

The application implements a complete [DHCP](#) client state machine with the following states:

- `SDHCP_NOTSTARTED`: System not initialized
- `SDHCP_DISCOVER`: Discovering [DHCP](#) servers
- `SDHCP_OFFER`: Processing server offers
- `SDHCP_ACK`: Acknowledging lease
- `SDHCP_CMPL`: [DHCP](#) lease complete
- `SDHCP_RENEW`: Renewing lease
- `SDHCP_REBIND`: Rebinding after renewal failure
- `SDHCP_RELEASE`: Releasing lease
- `SDHCP_FAILED`: [DHCP](#) process failed

12.7.4.1.6.2 Web Server Integration

- **HTTP Server:** Built-in web server on port 80
- **Form Processing:** POST request handling for configuration changes
- **Dynamic Content:** Real-time display of network interface status
- **Redirect Logic:** Automatic redirection after IP address changes

12.7.4.1.6.3 Network Interface Management

- **Multi-interface Support:** Handles ethernet and WiFi interfaces
- **Interface Discovery:** Automatic detection of available interfaces
- **Configuration Persistence:** Settings saved across reboots
- **Runtime Flexibility:** Changes can be applied with or without reboot

12.7.4.1.7 Optional Features

12.7.4.1.7.1 WiFi Support

- Uncomment `#define USE_WIFI (1)` to enable WiFi functionality
- Requires additional WiFi driver initialization
- Supports dual ethernet/WiFi configurations

12.7.4.1.7.2 Custom Static IP Assignment

- Option to define static IP values in application code
- Alternative to using configuration server settings
- Useful for embedded applications with fixed network requirements

12.7.4.1.8 Dependencies

- NetBurner NNDK (NetBurner Network Development Kit)
- Standard C++ libraries
- NetBurner networking stack
- Configuration server module
- HTTP server module

12.7.4.1.9 Build Configuration

The application uses pragma directives to suppress GCC warnings for NetBurner-specific format specifiers (`HI`, `I`) used in network address printing.

12.7.4.1.10 Network Diagnostics

The application includes comprehensive network diagnostics:

- Interface status monitoring
- Link state detection
- IP address validation
- **DHCP** lease status tracking
- Network connectivity verification

12.8 DNS Client

12.8.1 DNS Client Application

12.8.1.1 Overview

This is a DNS (Domain Name System) client example application for NetBurner embedded devices. The application demonstrates how to resolve domain names to IP addresses using both IPv4 and IPv6 protocols.

12.8.1.2 Features

- **Dual-stack DNS resolution:** Supports both IPv4 and IPv6 address resolution
- **Interactive interface:** Prompts user for domain names to resolve via serial terminal
- **Network interface validation:** Automatically checks network configuration before attempting DNS queries
- **Comprehensive error handling:** Provides detailed error messages for various DNS failure scenarios
- **Real-time network status:** Displays current network interface status and configuration

12.8.1.3 Application Functions

12.8.1.3.1 Core DNS Functions

The application uses three main NetBurner DNS functions:

1. **GetHostByName()** - General DNS resolution (returns first available IP address)
2. **GetHostByName() with IPv4** - Specific IPv4 address resolution
3. **GetHostByName() with IPv6** - Specific IPv6 address resolution (AAAA records)

12.8.1.3.2 Key Components

12.8.1.3.2.1 ExecuteDNSRequest(char *name)

Performs DNS lookups for a given domain name using all three resolution methods:

- General IP resolution
- IPv4-specific resolution
- IPv6-specific resolution (when not compiled with IPV4ONLY flag)

12.8.1.3.2.2 CanInterfaceDoDNS(int ifNumber)

Validates that a network interface has the required configuration for DNS:

- Active network link
- Configured DNS server IP address
- Configured network gateway

12.8.1.3.2.3 UserMain(void *pd)

Main application loop that:

- Initializes the network stack
- Waits for active network connection
- Validates network interface capabilities
- Provides interactive DNS lookup interface

12.8.1.4 Prerequisites

For the application to function properly, your NetBurner device must have:

1. **Active network connection** - Physical link must be established
2. **Valid IP configuration** - IP address, subnet mask, and gateway
3. **DNS server configuration** - Either from [DHCP](#) or static configuration
4. **Internet access** - Required to reach external DNS servers

12.8.1.5 Usage Instructions

12.8.1.5.1 Setup

1. Connect your NetBurner device to the network
2. Start MTTY (NetBurner's terminal program)
3. Connect to the debug serial port
4. Load and run the DNS client application

12.8.1.5.2 Operation

1. The application will display the network interface status
2. If the network is properly configured, you'll be prompted to enter a domain name
3. Type a domain name (e.g., `www.netburner.com`) and press Enter
4. The application will display the resolved IP addresses for IPv4 and IPv6 (if available)

12.8.1.5.3 Example Domain Names to Test

The application suggests several interesting test cases:

- **google.com** - Resolves both IPv4 and IPv6 addresses
- **ipv6.vm1.test-ipv6.com** - IPv6-only resolution
- **www.netburner.com** - NetBurner's website (recommended test)

12.8.1.6 Error Handling

The application provides specific error messages for common DNS failures:

- **DNS_TIMEOUT** - DNS server did not respond within 5 seconds
- **DNS_NOSUCHNAME** - Domain name does not exist
- **Other DNS errors** - Displays error code for debugging

12.8.1.7 Network Interface Diagnostics

The application performs comprehensive network diagnostics:

```
Interface[eth0]:  
Link UP  
Has a DNS address (8.8.8.8)  
Has a Gateway address (192.168.1.1)  
This interface is capable of doing DNS!
```

If any component is missing, it will be flagged as a problem.

12.8.1.8 Compilation Options

- **IPV4ONLY** - Define this flag to disable IPv6 functionality and compile for IPv4-only environments

12.8.1.9 Technical Details

- **DNS timeout:** 5 seconds per query
- **Buffer size:** 128 characters for domain name input
- **Network wait:** Up to 10 seconds for network interface activation
- **System diagnostics:** Enabled for debugging (should be disabled in production)

12.8.1.10 Troubleshooting

12.8.1.10.1 Common Issues

1. **Found no interface capable of doing DNS**

- Check physical network connection
- Verify IP configuration ([DHCP](#) or static)
- Ensure DNS server is configured
- Confirm gateway is set

2. **DNS_TIMEOUT errors**

- Check internet connectivity
- Verify DNS server is reachable
- Check firewall settings

3. **DNS_NOSUCHNAME errors**

- Verify domain name spelling
- Try a known good domain like `google.com`

12.8.1.10.2 Network Configuration

Ensure your NetBurner device has proper network configuration:

- Valid IP address
- Correct subnet mask
- Proper gateway configuration
- DNS server IP address (often provided by [DHCP](#))

12.8.1.11 Application Flow

1. Initialize network stack
2. Enable system diagnostics
3. Wait for active network (up to 10 seconds)
4. Check each network interface for DNS capability
5. If capable interface found, enter interactive mode
6. Accept domain names from user and perform DNS lookups
7. Display results and repeat

This application serves as both a functional DNS client and a diagnostic tool for network connectivity issues on NetBurner embedded devices.

12.9 Embedded Flash File System (EFFS)

Embedded Flash File System (EFFS) examples. There are two types of file systems:

- EFFS FAT: A FAT 32 file system used for flash memory cards
- EFFS STD: A power fail-safe file system that resides in the flash memory of the module itself

Example Catagories:

- [FAT File System \(EFFS-FAT\)](#)
- [Standard File System \(EFFS-STD\)](#)

12.9.1 FAT File System (EFFS-FAT)

EFFS FAT32 Examples:

- [Application Update](#)
- [Basic](#)
- [FTP](#)
- [HTTP](#)
- [HTML Variables](#)
- [Multiple Tasks](#)
- [Performance Tests](#)
- [RAM Drive](#)

12.9.1.1 Application Update

This application demonstrates how to update a NetBurner application in onboard flash memory from an external SD flash card. The application will look for a file on the flash card with the name specified by APPFILENAME. Use MTTTY to monitor the serial output of your NetBurner device. If this file is found, an application update can be performed by typing a character in response to the MTTTY prompt.

Note

All EFFS FAT examples require that you add the Embedded Flash File System File Allocation Table (EFFS FAT) library to your project: [Add a Library to a Project](#)

12.9.1.2 Basic

12.9.1.2.1 EFFS-FAT-Basic File System Example

12.9.1.2.1.1 Overview

This program demonstrates basic file system operations for SD/MMC and Compact Flash cards using the Embedded Flash File System with File Allocation Table (EFFS-FAT) library. The application provides a comprehensive example of external flash storage management on embedded systems.

Note

All EFFS FAT examples require that you add the Embedded Flash File System File Allocation Table (EFFS FAT) library to your project: [Add a Library to a Project](#)

12.9.1.2.1.2 Features

The application demonstrates the following file system operations:

- **Drive Mounting:** Initialize and mount external storage devices
- **Space Management:** Determine used and free file space on the storage device
- **File Creation:** Create new files on the external storage
- **Data Writing:** Write data to files on the storage device
- **Data Reading:** Read data from files on the storage device
- **Drive Unmounting:** Safely unmount the storage device

12.9.1.2.1.3 Supported Storage Types

The application supports multiple storage interfaces:

- **SD/MMC Cards:** Secure Digital and MultiMediaCard support
- **Compact Flash Cards:** CompactFlash card support
- **Multi-drive configurations:** For modules with onboard flash sockets

12.9.1.2.1.4 Hardware Compatibility

The application includes specific support for:

- MOD5441X modules
- SBE70LC modules
- Other NetBurner modules with MMC/SD or CFC interfaces

12.9.1.2.1.5 Program Flow

1. **Initialization:** Network stack initialization and system diagnostics
2. **File System Entry:** Register the current task for file system access
3. **External Flash Initialization:** Initialize the CFC or SD/MMC drive
4. **Status Display:** Show file space usage statistics
5. **Directory Listing:** Display existing files and directories
6. **Read/Write Operations:** Perform file system read and write tests
7. **Cleanup:** Unmount the file system and release resources

12.9.1.2.1.6 Important Notes

File System Access

- The `f_enterFS()` function must be called in every task that accesses the file system
- This function should only be called once per task before any other file system operations
- Up to 10 tasks can simultaneously access the file system (can be increased)
- Tasks can be removed from file system access using `f_releaseFS()`

Web Server Limitations

- For web server functionality with external storage, refer to the EDFS-HTTP example

Library Requirements

Note

All EDFS FAT examples require adding the Embedded Flash File System File Allocation Table (EDFS FAT) library to your project.

12.9.1.2.1.7 Output

The program displays status information through the debug serial port, including:

- Program startup messages
- File system statistics
- Directory contents
- Read/write operation results
- Program completion status

12.9.1.2.1.8 Usage

1. Ensure your NetBurner module has the appropriate external flash interface (SD/MMC or CFC)
2. Insert a compatible flash card into the module
3. Build and deploy the application to your NetBurner module
4. Monitor the debug serial output for program status and results
5. Reset the device to repeat the demonstration

12.9.1.2.1.9 Build Configuration

The application uses preprocessor definitions to determine the storage type:

- `USE_MMC`: Enable MMC/SD card support
- `USE_CFC`: Enable Compact Flash card support
- `MULTI_MMC`: Enable multi-drive MMC support (for specific modules)

12.9.1.2.1.10 File Structure

- `main.cpp`: Main application code with `UserMain()` function
- `ReadMe.txt`: Original documentation
- `FileSystemUtils.h`: Header file for file system utility functions

12.9.1.2.1.11 Error Handling

The application includes proper error handling for:

- File system mounting/unmounting operations
- File access operations
- Network initialization timeouts
- Resource cleanup

Note

After completing all operations, the program enters an infinite loop and must be reset to repeat the demonstration.

12.9.1.3 FTP

This program illustrates file system and FTP operations for SD Flash cards:

- Mounting a flash drive
- Determining amount of used and free file space
- FTP access. We recommend Filezilla or WinScp

Status messages will be sent out the debug serial port.

Modules with an onboard microSD flash socket should use the multi MMC header files and functions because the modules are capable of supporting both onboard and external flash cards (even if your application only uses one).

Note

All EDFS FAT examples require that you add the Embedded Flash File System File Allocation Table (EDFS FAT) library to your project: [Add a Library to a Project](#)

12.9.1.3.1 FTP File System Application

12.9.1.3.1.1 Overview

This application demonstrates file system and FTP operations for SD Flash cards on NetBurner embedded systems. It provides a complete example of mounting flash drives, managing file operations, and enabling FTP access for remote file management.

12.9.1.3.1.2 Features

File System Operations

- Mount and initialize external flash drive
- Display directory contents and file listings
- Read and display text files
- Format flash cards (with confirmation prompt)
- Monitor disk space usage (used/free space statistics)
- Support for both long filenames and 8.3 format

FTP Server

- Built-in FTP server running on port 21
- Remote file access via FTP clients (recommended: FileZilla or WinSCP)
- Concurrent file system access for local operations and FTP
- Task-based priority management for FTP operations

System Features

- Network initialization with [DHCP](#) support
- NTP time synchronization with manual fallback
- Timezone configuration support
- Interactive command-line interface
- System diagnostics and debug output

12.9.1.3.1.3 Supported Hardware

Flash Card Types

- **SD/MMC Cards:** Secure Digital and MultiMediaCard support

Module Compatibility

- **MOD5441X:** Onboard microSD flash socket
- **SBE70LC:** Onboard microSD flash socket
- Other NetBurner modules with external flash card support

12.9.1.3.1.4 Command Interface

The application provides an interactive menu system accessible via the debug serial port:

Command	Description
D	Display Directory - Shows all files and folders on the flash card
E	Display TestFile.txt - Shows contents of the test file
F	Format SD Flash card - Formats the card (WARNING: destroys all data)
S	Display Space usage - Shows used and free space statistics
T	Display system Time - Shows current system time and date
?	Display Menu - Shows the command menu

12.9.1.3.1.5 Time Configuration

The application includes timezone support with examples for US time zones:

```
// Eastern Time
tzsetchar("EST5EDT4,M3.2.0/02:00:00,M11.1.0/02:00:00");

// Central Time
tzsetchar("CST6CDT5,M3.2.0/02:00:00,M11.1.0/02:00:00");

// Mountain Time
tzsetchar("MST7MDT6,M3.2.0/02:00:00,M11.1.0/02:00:00");

// Pacific Time (default)
tzsetchar("PST8PDT7,M3.2.0/02:00:00,M11.1.0/02:00:00");
```

12.9.1.3.1.6 Setup Requirements

Library Dependencies

- **EFS FAT Library:** Embedded Flash File System with File Allocation Table support
- `FatFile` Must be added to your NetBurner Eclipse project before compilation (`libFatFile.a`)

Hardware Setup

1. Insert SD/MMC card into the appropriate socket
2. Ensure network connection for [DHCP](#) and NTP functionality
3. Connect debug serial cable for command interface

Network Configuration

- [DHCP](#) client automatically obtains IP address
- FTP server accessible on port 21
- 5-second timeout for network initialization

12.9.1.3.1.7 File System Architecture

Multi-Drive Support

Modules with onboard microSD sockets use multi-MMC drivers to support both:

- Onboard flash cards
- External flash cards
- Simultaneous access to multiple cards

Task Management

- Main application runs at `MAIN_PRIO`
- FTP server runs at `FTP_PRIO (MAIN_PRIO - 2)`
- Up to 10 tasks can access the file system concurrently
- Each task must call `f_enterFS()` before file operations

12.9.1.3.1.8 Usage Examples

Basic File Operations

1. Start the application
2. Wait for network initialization
3. Use 'D' command to view directory contents
4. Use 'S' command to check available space
5. Use 'E' command to view test file contents

FTP Access

1. Connect FTP client to the device's IP address on port 21
2. Browse and transfer files using standard FTP commands
3. Simultaneous local and remote access is supported

Formatting Flash Cards

1. Use 'F' command from the menu
2. Confirm with 'Y' when prompted
3. **WARNING:** This will erase all data on the card

12.9.1.3.1.9 Technical Notes

- All file system operations require prior call to `f_enterFS()`
- Debug output is sent to the serial port
- System diagnostics are enabled (disable for production)
- NTP synchronization attempts before falling back to manual time setting
- Long filename support depends on compilation flags

12.9.1.3.1.10 Recommended FTP Clients

- **FileZilla:** Free, cross-platform FTP client
- **WinSCP:** Windows-based secure file transfer client

12.9.1.3.1.11 Build Configuration

Compile-time options:

- `USE_MMC`: Enable SD/MMC support
- `USE_CFC`: Enable Compact Flash support
- `MOD5441X` or `SBE70LC`: Enable multi-drive support for onboard sockets
- `F_LONGFILENAME`: Enable long filename support

12.9.1.4 HTTP

12.9.1.4.1 Overview

This example demonstrates how to use the EFS-FAT file system with SD cards. This file system is different than the EFS-STD file system which uses onboard flash chip memory.

Note

All EFS FAT examples require that you add the Embedded Flash File System File Allocation Table (EFS FAT) library to your project: [Add a Library to a Project](#)

12.9.1.4.2 EFS HTTP Application

A NetBurner embedded application that provides HTTP web server and FTP server functionality with SD/MMC/CFC flash card file system support using the EFS-FAT (Embedded Flash File System - File Allocation Table) library.

12.9.1.4.2.1 Overview

This application demonstrates comprehensive file system operations on embedded NetBurner devices, combining local file system management with network-based file access through HTTP and FTP protocols. The application supports multiple flash card types and provides both command-line and web-based interfaces for file operations.

12.9.1.4.2.2 Supported Hardware

The application automatically detects and configures support for different flash card types:

- **MMC/SD Cards:** Standard MultiMediaCard and Secure Digital cards
- **Multi-MMC:** Multiple MMC support for MOD5441X and SBE70LC modules
- **CFC:** CompactFlash Card support
- **SDHC:** Secure Digital High Capacity cards (NANO54415, SB800EX)

12.9.1.4.2.3 Core Features

File System Operations

- **File Management:** Create, read, write, and display text files
- **Directory Operations:** List directory contents and navigate file structure
- **Format Support:** Format flash cards (with data loss warning)
- **Space Monitoring:** Display file system usage statistics
- **File I/O Testing:** Built-in tests for `fgets()`, `fputs()`, and `fprintf()` functions

Network Services

- **HTTP Web Server:** Serves files from flash card with MIME type support
- **FTP Server:** Full FTP access for file upload/download operations
- **NTP Time Sync:** Automatic time synchronization for accurate file timestamps
- **DHCP Support:** Automatic network configuration

Web Interface Features

- **Directory Browsing:** Web-based file and directory listing
- **File Serving:** Direct file access through HTTP URLs
- **MIME Type Support:** Configurable MIME types via `MIME.txt` file or built-in defaults
- **Index Page Priority:** Automatic detection of `index.html` files

12.9.1.4.2.4 Application Structure

Main Components

1. **main.cpp:** Core application logic and command-line interface
2. **web.cpp:** HTTP server callbacks and web interface implementation
3. **Supporting Libraries:** `FileSystemUtils`, `effs_time`, `http_f` modules

Command-Line Interface

The application provides an interactive serial menu with the following options:

Command	Function
D	Display directory contents
E	Display <code>TestFile.txt</code> contents
F	Format SD flash card (destructive operation)
G	Run <code>fgets()</code> test on <code>TestFile.txt</code>
H	Run <code>fputs()</code> test on <code>TestFile.txt</code>
P	Run <code>fprintf()</code> test
S	Display file system space usage
T	Display current system time
?	Display command menu

12.9.1.4.2.5 Network Configuration

HTTP Server

- **Default Port:** 80
- **File Serving:** Direct access to files on flash card
- **Directory Listing:** Append `/DIR` to any URL for directory view
- **MIME Support:** Configurable content types

FTP Server

- **Default Port:** 21
- **Full Access:** Upload, download, and manage files
- **Long Filename Support:** Automatic detection and notification

Time Synchronization

- **Primary:** NTP server synchronization
- **Fallback:** Manual time setting if NTP fails
- **Time Zones:** Configurable (defaults to PST8PDT7)

12.9.1.4.2.6 File System Details

EFS-FAT Integration

- **Multi-Task Support:** Up to 10 tasks can access the file system
- **Thread Safety:** Proper file system entry/exit for each task
- **Drive Management:** Automatic drive selection based on hardware type

MIME Type Handling

The application supports MIME types through two mechanisms:

1. **MIME.txt File:** User-configurable MIME type definitions on the flash card
2. **Built-in Types:** Hard-coded support for common file types:
 - `jpg > image/jpeg`
 - `gif > image/gif`
 - `htm/html > text/html`
 - `xml > text/xml`
 - `css > text/css`
 - `mp4 > video/mp4`

12.9.1.4.2.7 Building and Configuration

Prerequisites

- NetBurner NNDK (NetBurner Network Development Kit)
- EFS-FAT library (FatFile.a)
- Compatible NetBurner hardware platform

Build Configuration

The flash card type is configured in `cardtype.h`. Modify the following defines:

- `USE_MMC`: For MMC/SD card support
- `USE_CFC`: For CompactFlash support
- `USE_SDHC`: For SDHC support
- `MULTI_MMC`: For multiple MMC support

Library Dependencies

Add to your makefile:

```
XTRALIB = $(NNDK_ROOT)/platform/$(PLATFORM)/original/lib/libFatFile.a
DBXTRALIB = $(NNDK_ROOT)/platform/$(PLATFORM)/original/lib/libFatFile.a
```

Or in NBEclipse, add "FatFile" to the GCC Linker Libraries.

12.9.1.4.2.8 Usage Examples

Web Interface Access

- **Default Page:** `http://[device-ip]/`
- **Directory Listing:** `http://[device-ip]/DIR`
- **Specific File:** `http://[device-ip]/filename.ext`
- **Subdirectory:** `http://[device-ip]/subdir/DIR`

FTP Access

Connect using any FTP client (e.g., FileZilla):

- **Host:** Device IP address
- **Port:** 21
- **Mode:** Standard FTP

Serial Interface

Connect via terminal emulator (MTTTY, PuTTY, etc.):

- **Baud Rate:** As configured for your NetBurner device
- **Interactive Menu:** Use single-letter commands

12.9.1.4.2.9 File Priority and Web Pages

When accessing the web interface, pages are served in the following priority order:

1. `index.html` file on flash card
2. First available `*.html` file on flash card
3. Generated directory listing page

12.9.1.4.2.10 Technical Specifications

Memory Usage

- **HTTP Buffer:** 32KB aligned buffer for file transfers
- **Temporary Buffer:** 256-byte buffer for line reading operations

Task Priorities

- **HTTP Server:** `HTTP_PRIO`
- **FTP Server:** `FTP_PRIO (MAIN_PRIO - 2)`
- **Main Application:** `MAIN_PRIO`

Performance Considerations

- File system searches occur for each HTTP request when serving from flash card
- For better performance, consider caching file system structure in memory
- `MIME.txt` lookups are not cached; keep the file small and organized by frequency

12.9.1.4.2.11 Error Handling

The application includes comprehensive error handling for:

- File system initialization failures
- Network service startup issues
- File access permissions and availability
- Flash card formatting operations

12.9.1.4.2.12 Security Notes

- FTP server provides full file system access
- Web server can serve any file accessible on the flash card
- Format operation permanently deletes all data
- No authentication mechanisms are implemented in this example

12.9.1.4.2.13 Troubleshooting

Common Issues

1. **File System Not Mounting:** Check flash card insertion and format
2. **NTP Sync Failure:** Verify network connectivity and DNS resolution
3. **FTP Connection Issues:** Check firewall settings and network configuration
4. **Web Pages Not Loading:** Verify file presence and MIME type configuration

Debug Information

The application provides extensive debug output through the serial interface, including:

- URL parsing details
- File system operation results
- Network service status
- MIME type resolution information

Note

If *.html files exist on the flash card, then the program will always access the first available web page file by default.

12.9.1.5 HTML Variables

This example program sets up HTTP access to the FAT file system on SD Flash cards.
To run this example:

1. Compile the example for your chosen flash card in cardtype.h. The NetBurner development board supports SD Flash cards.
2. Copy index.html and MIME.txt from the example's root directory onto your flashcard, or put your own there.
3. When the program runs, status messages will be displayed on the module debug serial port.

4. You can view the files on the flash card with any web browser by typing in the IP address of the module at the "/DIR" folder Example: "http://10.1.1.57/DIR").
5. When accessing the module via browser, what web pages will be displayed depends on the following order and their availability on the flash card:
 - index.html files
 - The first available .html file
 - A web page list of all directories and files that exist on the flash card

Note

If .html files exist on the flash card, then the program will always access the first available web page file by default. If you want to access the web page list of all existing directories and files on the flash card, then you type in the URL of the module, followed by "/dir" (Example: "http://10.1.1.57/dir").

12.9.1.6 Multiple Tasks

This program illustrates basic file system operations for SD Flash cards for multiple tasks:

- Mounting a drive
- Determining amount of used and free file space
- Creating files
- Writing data
- Reading data
- Unmounting a drive

When the program executes it will display program status information through the debug serial port.

Note

All EDFS FAT examples require that you add the Embedded Flash File System File Allocation Table (EDFS FAT) library to your project: [Add a Library to a Project](#)

12.9.1.7 Performance Tests

This example is used to test the speed of the EDFS FAT file system for operations such as file creation and data storage speed. All testing was done on a MOD54415.

Hardware:

- MOD54415-100IR on MOD-DEV-70 development board
- Kingston microSDHC 32GB Class 4, with MidroSC Adapter

Software: NetBurner tools revision 2.8.6 Date: February 7, 2018

Test results for file creation:

- 1 file: 15ms
- 100 files: 32ms
- 1,000 files: 93ms

Test results for data transfer, using 1MB of data:

- Write speed: 0.61 MBytes per second
- Read speed: 2.18 MBytes per second

12.9.1.8 RAM Drive

This program illustrates basic file system operations for RAM drives:

- Mounting a drive
- Determining amount of used and free file space
- Creating files
- Writing data
- Reading data
- Un-mounting a drive

When the program executes it will display program status information through the debug serial port.

This example is similar to the EFFECTS-BASIC example, but uses a RAM drive for the file system. The RAM drive configuration is located in the 'ramdrvMcf.cpp' source file.

To build any of the EFFECTS- examples to use the RAM drive, add the 'ramdrvMcf.cpp' file to the build list and modify 'cardType.h' to define USE_RAM instead of USE_MMC and set the EXT_FLASH_DRV_NUM to the correct RAM drive number.

The RAM drive supports 12, 16 and 32 bit file systems. The most commonly used size is 12-bit, which supports drive sizes from 50k up to 2.8MB. If you change the FAT size, you must make the corresponding change in the function: `int fstat = f_format(RAM_DRV_NUM, F_FAT12_MEDIA);`

To change the RAM drive file size, edit `ramdrv_mcf.cpp`.

Minimum RAM drive file system sizes: FAT12: 50KB (most commonly used) FAT16: 2.8MB FAT32: 32MB

Note

All EFFECTS FAT examples require that you add the Embedded Flash File System File Allocation Table (EFFECTS FAT) library to your project: [Add a Library to a Project](#)

12.9.2 Standard File System (EFFECTS-STD)

Examples for on-chip flash file system:

- [EFFECTS-STD HTTP](#)

12.9.2.1 EFFECTS-STD HTTP

12.9.2.1.1 Overview

This example demonstrates how to use the EFFECTS-STD on-chip flash file system. This file system is different than the EFFECTS-FAT file system which is a FAT32 file system that works with SD Flash cards.

12.9.2.1.1.1 Application Functionality

- Creation of a text file from within the application named "TestFile.txt"
- Display of the text file
- Read/Write test of the text file
- Display of file space and amount used
- Format of the file system
- Read/Write of a diagnostics text file. This will be the data displayed on the System Diagnostics web page if that feature is enabled.

12.9.2.1.1.2 Network Functionality

- NTP is used to set the system time and time zone so that file time stamps are correct.
- HTTP web page interface and file display. If you use FTP to put in your own index.html page, it will be used in place of the example default page.
- FTP Server for file access. You can use client programs such as FileZilla to upload/download files

Note

Running the example in an environment where the device has Internet access will enable the Network Time Protocol (NTP) to automatically set the time. Otherwise, you can set the time manually through the serial port menu.

12.9.2.1.1.3 File System Size

The example provides suggested file system sizes based on the total amount of available flash memory in each product. For example, the MODM7AE70 has a total of 2MB and 128k is allocated in the example. The MOD5441x products has 32MB total flash, and 1MB is used for the file system.

The flash chip configuration header files are located in the "flashChip" folder of the EFFS-STD example:

Platform	Header File	Total Flash	Example EFFS Size
MODM7AE70, SBE70LC	SAME70↔ Q21.h	2MB	128K
MOD54415, MOD54417	MX29GL256↔ F.h	32MB	1MB
NANO54415, SB800EX	MX25L6406↔ E.h	8MB	1MB

Please refer to the header files if you wish to change the size of the file system.

Note

The EFFS-STD manual, "EFFS-STD-R1p91.pdf", is located in the "\nburn\docs" folder.

12.9.2.1.2 Building The Example

12.9.2.1.2.1 NBEclipse With Example Import Feature or Command Line

If you build a new project and use the NBEclipse Example Import feature, all settings will be take care of automatically. If you are building using the example makefile on the command line, the makefile contains all necessary settings.

12.9.2.1.2.2 Adding EFFS-STD Functionality to an Existing NBEclipse Project

There are two steps to add EFFS-STD to your project.

1. Add the StdFFile.a library to the GCC Linker
2. Reserve space in the Flash memory for the file system

The first step is the same for all platforms:

- Right click on your project.
- Navigate to C/C++ Build > Settings > GNC C/C++ Linker > Libraries.
- In the Libraries section, add "StdFFile". Note there is no .a extension.

The second step is described in the following sections.

MODM7AE70, SBE70LC Platforms

The "Locate Application at Fixed Address in Flash" feature must be enabled. This will disable the background Flash erase feature, would will erase the flash file system sectors.

- Right-click on the project and select "Properties"
- Navigate to C/C++ Build > Settings > NB Flashpack > General
- Check the checkbox for "Locate Application at Fixed Address in Flash"

NANO54415, SB800EX Platforms

The CompCode Flag settings are used to reserve space in the memory map for the file system.

- Right click on your project.
- Navigate to C/C++ Build > Settings > NBCompCode > General
- Modify the memory address range field from:
COMPCODEFLAGS = 0x04000 0x800000 // Original
to
COMPCODEFLAGS = 0x04000 0x700000 // Reserve space for file system

The first address parameter is the start of application space, and the second is the start of file system space

MOD54415, MOD54417 Platforms

The CompCode Flag settings are used to reserve space in the memory map for the file system.

- Right click on your project.
- Navigate to C/C++ Build > Settings > NBCompCode > General
- Modify the memory address range field from:
COMPCODEFLAGS = 0xC0040000 0xC2000000 // Original
to
COMPCODEFLAGS = 0xC0040000 0xC1F00000 // Reserve space for file system

The first address parameter is the start of application space, and the second is the start of file system space

12.9.2.1.2.3 Adding EFS-STD Functionality to a Makefile

There are two steps:

1. Add the EFS-STD Library
2. Add the compcode flags (SB800EX, NANO54415, MOD5441x) or enable application fixed address (MODM7AE70, SBE70LC)

Add the EFS-STD Library

```
XTRALIB = $(NNDK_ROOT)/platform/$(PLATFORM)/original/lib/libStdFFFile.a
DBXTRALIB = $(NNDK_ROOT)/platform/$(PLATFORM)/original/lib/libStdFFFile.a
```

MODM7AE70, SBE70LC Platforms

Enable "Locate Application at Fixed Address in Flash" with: EXTRAPACKARGS += -cflag C:3

NANO54415, SB800EX Platforms

Modify the memory address range field from:

COMPCODEFLAGS = 0x04000 0x800000 // Original

to

COMPCODEFLAGS = 0x04000 0x700000 // Reserve space for file system

MOD54415, MOD54417 Platforms

Modify the memory address range field from:

COMPCODEFLAGS = 0xC0040000 0xC2000000 // Original

to

COMPCODEFLAGS = 0xC0040000 0xC1F00000 // Reserve space for file system

12.9.2.1.3 Running the Example

The application provides access through an interactive serial port menu, web page, and FTP.

12.9.2.1.3.1 Serial Interface

- Run MTTTY or some other serial terminal program and connect to the device
- Status messages will be displayed as well as an interactive menu. If this is the first time the application has been run, the file system will be formatted.
- You can experiment with the various menu options.

12.9.2.1.3.2 Web Server (HTTP)

- The default web page will provide various interactive options and information display.
- Entering the IP address followed by "/DIR" (eg. <http://10.1.1.57/DIR>) will display the file directory
- The web server will display the default page based on what is in the file system in the following order: – The `index.html` (will supersede `index.html` file compiled into the application) – The first available `*.html` file
- Use FTP (see below) to upload your own `index.html` file and it will replace the application default.

12.9.2.1.3.3 FTP Operation

Connection to the device with a FTP client such as FileZilla will enable you to upload, download and view files.

12.10 Ethernet

Ethernet examples:

- [Manual Configuration](#)

12.10.1 Manual Configuration

12.10.1.1 Ethernet Manual Configuration Test Application

12.10.1.1.1 Overview

This application demonstrates how to manually configure Ethernet speed and duplex settings instead of using the default autonegotiation mode. The program provides an interactive command-line interface for testing different Ethernet configurations and includes network diagnostic tools.

12.10.1.1.2 Features

12.10.1.1.2.1 Ethernet Configuration Options

- **Autonegotiation mode** - Automatically negotiates optimal speed and duplex
- **100Mbps Full Duplex** - High-speed bidirectional communication
- **100Mbps Half Duplex** - High-speed unidirectional communication
- **10Mbps Full Duplex** - Standard speed bidirectional communication
- **10Mbps Half Duplex** - Standard speed unidirectional communication

12.10.1.1.2.2 Network Diagnostic Tools

- **ARP Cache Display** - Shows the Address Resolution Protocol cache
- **Network Counters** - Displays various network statistics
- **Ethernet Register Dump** - Shows low-level Ethernet hardware registers
- **Ping Functionality** - Tests network connectivity to specified IP addresses
- **Link Status Monitor** - Displays current Ethernet link state and configuration
- **System Timer** - Shows OS tick counter and uptime

12.10.1.1.3 Usage

12.10.1.1.3.1 Interactive Menu Commands

After starting the application, use these single-character commands:

```
1 - Set Autonegotiate mode
2 - Set 100Mbps Full Duplex
3 - Set 100Mbps Half Duplex
4 - Set 10Mbps Full Duplex
5 - Set 10Mbps Half Duplex
A - Show ARP Cache
C - Show Counters
E - Show Ethernet Registers
P - Ping (Example: "P 192.168.1.1")
W - Show OS Seconds Counter
? - Display Menu
```

12.10.1.1.3.2 Ping Command Examples

- **P** - Ping the default gateway
- **P 192.168.1.1** - Ping a specific IP address
- **P 10.0.0.1** - Ping another specific address

12.10.1.1.4 Technical Details

12.10.1.1.4.1 Platform Support

- Supports NetBurner embedded systems
- Special handling for MCF5441X multi-port Ethernet devices
- Primary port configuration for multi-port devices

12.10.1.1.4.2 Network Protocols

- **IPv4** - Full support with [DHCP](#), static, and auto-IP configuration
- **IPv6** - Complete IPv6 support including:
 - Link-local addresses
 - Router-assigned addresses
 - [DHCP](#) v6 addresses
 - Static addresses
 - DNS configuration

Warning

Manual Ethernet configuration is not available in debug mode. The application will display a warning message when attempting to change settings while debugging.

12.10.1.1.5 File Structure

- `main.cpp` - Main application with command interface and Ethernet configuration
- `ip_util.cpp` - IP address utility functions for display and web interface
- `ip_util.h` - Header file for IP utilities

12.10.1.1.6 Network Information Display

12.10.1.1.6.1 IPv4 Details

- IP address and subnet mask
- Primary and secondary DNS servers
- Default gateway
- Auto-IP configuration status

12.10.1.1.6.2 IPv6 Details

- Link-local addresses
- Router-assigned prefixes
- [DHCP](#) v6 leases with timing information
- Static address assignments
- DNS server information

12.10.1.1.6.3 Interface Status

- Link state (UP/DOWN)
- Current speed (10/100 Mbps)
- Duplex mode (Full/Half)
- MAC address information

12.10.1.1.7 Web Interface

The application includes a web server component that displays network configuration through a browser interface. The web page shows:

- Complete interface information
- Clickable links for IPv4 and IPv6 addresses
- [DHCP](#) lease timing details
- DNS server assignments

12.10.1.1.8 System Requirements

- NetBurner RTOS environment
- Ethernet hardware interface
- Serial port for command interface
- Optional web browser for web interface access

12.10.1.1.9 Notes

- The application waits up to 5 seconds for network initialization on startup
- System diagnostics are enabled by default (should be disabled for production)
- All network interfaces are automatically detected and configured
- Link status is continuously monitored and displayed

12.11 Exception Try/Catch

12.11.1 Exception Try/Catch Application

12.11.1.1 Overview

This example program demonstrates how to use C++ exceptions in an embedded NetBurner RTOS environment. The application throws a runtime exception and catches it, displaying the error message to stdout for debugging purposes.

12.11.1.2 Application Details

Application Name:** ExceptionTryCatch

The main application performs the following operations:

1. Initializes the system and enables diagnostics
2. Waits for network connectivity (5 second timeout)
3. Throws a `runtime_error` exception with the message "Error"
4. Catches the exception and displays the error message
5. Enters an infinite loop with 1-second delays

12.11.1.3 Key Features

- Demonstrates proper C++ exception handling with try/catch blocks
- Uses standard C++ exception types (`runtime_error`, `exception`)
- Integrates with NetBurner RTOS system initialization
- Outputs results to stdout for debugging via MTTY on the debug serial port

12.11.1.4 Output

When run, the application will display:

```
Throwing an exception...  
An exception occurred. Exception: Error
```

12.11.1.5 Compilation Requirements

Note

C++ exceptions are disabled by default in NetBurner applications to keep application size small. To use exceptions, you must enable them during compilation.

12.11.1.5.1 Required Compiler Flags

The following flags must be added to the GNU C++ compiler options:

- `-fexceptions` - Enables C++ exception handling
- `-frtti` - Enables Run-Time Type Information (required for exceptions)

12.11.1.5.2 How to Enable Exceptions

1. Right-click on your project in the IDE
2. Select "Properties"
3. Navigate to "GNU C++ Compiler"
4. Select "Miscellaneous"
5. In the "Other Flags" field, add: `-fexceptions -frtti`

Warning

If these flags are not enabled, the compiler will notify you in the output console window.

12.11.1.6 Dependencies

- NetBurner RTOS headers ([init.h](#), [nbrtos.h](#))
- Standard C++ iostream library
- Standard C++ exception handling

12.11.1.7 Usage

This example is primarily educational and demonstrates the proper setup and usage of C++ exceptions in a NetBurner embedded environment. The output can be monitored using MTTTY connected to the debug serial port.

12.11.1.8 Development Notes

- System diagnostics are enabled in this example but should be removed for production code
- The application waits for network connectivity before proceeding
- Exception handling adds overhead, so consider the trade-offs for production applications

12.12 External IRQ

12.12.1 External IRQ Example

12.12.1.1 Overview

This example application demonstrates how to configure and use external interrupt (IRQ) functionality on NetBurner embedded platforms. The program sets up a GPIO pin to trigger an external interrupt, allowing you to test interrupt handling in real-time through a simple serial terminal interface.

12.12.1.2 Supported Platforms

- **MOD5441x** (MOD54415, MOD54417, SB800EX)
- **NANO54415**
- **MODM7AE70**
- **SBE70LC**
- **SOMRT1061**

12.12.1.3 Hardware Setup

12.12.1.3.1 Required Connection

A **jumper wire** must be placed between the GPIO output pin and the desired interrupt input pin for each platform:

12.12.1.3.1.1 MOD5441x Platforms

- **GPIO Output:** J2.33
- **Available IRQ Inputs:**
 - IRQ 1: J2.45
 - IRQ 2: J2.43
 - IRQ 3: J2.26 (default in code)
 - IRQ 6: J2.47

12.12.1.3.1.2 NANO54415

- **GPIO Output:** Pin 12 (Dev board P2.12)
- **Available IRQ Inputs:**
 - IRQ 2: Pin 50 (Dev board P1.50)
 - IRQ 3: Pin 49 (Dev board P1.49) (default in code)

12.12.1.3.1.3 MODM7AE70

- **GPIO Output:** P2.15 (connected to first LED on MOD-DEV-70 board)
- **IRQ Input:** P2.8 (configurable to any GPIO pin)

12.12.1.3.1.4 SBE70LC

- **GPIO Output:** J1[4]
- **IRQ Input:** J1[3]

12.12.1.3.1.5 SOMRT1061

- **GPIO Output:** Pin 86 (connected to first LED on DEV-S61 board)
- **IRQ Input:** Pin 14

12.12.1.4 How It Works

12.12.1.4.1 Interrupt Mechanism

1. The application configures a GPIO pin as an output (initially high)
2. Another pin is configured as an external interrupt input
3. When you trigger an interrupt via the menu, the GPIO output goes low
4. This transition triggers the external interrupt
5. The Interrupt Service Routine (ISR) executes:
 - Sets the GPIO pin back to high
 - Increments the interrupt counter
6. The main loop displays the current interrupt count

12.12.1.4.2 Interrupt Service Routine (ISR)

The ISR behavior varies by platform:

MCF5441x Platforms (MOD5441x, NANO54415):**

```
void ExternalISR()
{
    // Set GPIO pin high and increment counter
    IrqCounter++;
}
```

ARM Platforms (MODM7AE70, SBE70LC, SOMRT1061):**

```
void ExternalISR(int port, int portPin)
{
    // Records which port/pin triggered the interrupt
    gRxIrqPort = port;
    gRxIrqPortPin = portPin;
    IrqCounter++;
}
```

12.12.1.5 User Interface

The application provides a simple command-line menu accessible via serial terminal (MTTTY or similar):

12.12.1.5.1 Available Commands

- **T** - Trigger interrupt by setting GPIO low
- **?** - Display the main menu
- **Enter** - Display current interrupt count

12.12.1.5.2 Menu Output

```
----- Main Menu -----
T - Trigger interrupt by writing GPIO low
? - Display Menu
<enter> - Display interrupt count

Number of interrupts: [count]
Last Irq: port=[x], pin=[y] // ARM platforms only
```

12.12.1.6 Configuration Options

12.12.1.6.1 IRQ Priority Levels

- **MCF5441x:** Fixed priority levels (1=lowest, 7=highest)
- **ARM Platforms:** Configurable priority (1=highest, 7=lowest, default=4)

12.12.1.6.2 IRQ Trigger Modes (ARM Platforms)

```
#define IRQ_POLARITY_HIGH_LEVEL    2
#define IRQ_POLARITY_RISING_EDGE  1
#define IRQ_POLARITY_EITHER_EDGE  0
#define IRQ_POLARITY_FALLING_EDGE -1 // Used in example
#define IRQ_POLARITY_LOW_LEVEL    -2
```

12.12.1.7 Building and Running

1. **Hardware Setup:** Connect jumper wire between GPIO output and IRQ input pins
2. **Compile:** Build the project using NetBurner development tools
3. **Deploy:** Flash the application to your NetBurner module
4. **Connect:** Use MTTTY or serial terminal (115200 baud) to interact
5. **Test:** Use menu commands to trigger interrupts and monitor counts

12.12.1.8 Key Functions Used

- `SetPinIrq()` - Configures pin for external interrupt detection
- `SetPortIrqPriority()` - Sets interrupt priority level (ARM platforms)
- GPIO pin configuration functions (platform-specific)

12.12.1.9 Important Notes

- The Edge Port Flag Register is automatically cleared by the `SetPinIrq()` implementation
- On ARM platforms, all GPIO pins on the same port share the same IRQ priority level
- The example uses falling edge detection by default on ARM platforms
- System diagnostics are enabled for debugging (remove for production)

12.12.1.10 Troubleshooting

- **No interrupts triggered:** Verify jumper wire connections
- **Unexpected behavior:** Check that only one IRQ input is uncommented in the code
- **Serial issues:** Ensure correct baud rate (115200) and terminal settings

This example serves as a foundation for implementing external interrupt handling in your own NetBurner applications.

12.13 Extra FD Circular Buffer

12.13.1 NetBurner Extra File Descriptor Circular Buffer

12.13.1.1 Overview

This application demonstrates how to implement a custom I/O device using NetBurner's Extra File Descriptor system. The example creates a circular buffer that can be accessed using standard file operations like `read()` and `write()`, making it behave like any other file descriptor in the system. The Extra File Descriptors are defined in the `'iointernal.h'` header file.

Note

If you have an I/O device and wish to use interrupts, after adding the appropriate interrupt code that is specific to your device, you would use `USER_ENTER_CRITICAL` and `USER_EXIT_CRITICAL` to protect the internal data structures.

12.13.1.2 Key Features

- **Custom I/O Device:** Implements a circular buffer as a file descriptor
- **Thread-Safe Operations:** Uses critical sections for multi-task safety
- **Overwrite Mode:** Configurable buffer overflow handling
- **Terminator-Based Flushing:** Intelligent buffer management for line-based data
- **Standard File Operations:** Compatible with `read()`, `write()`, and `fdprintf()`

12.13.1.3 Architecture

12.13.1.3.1 Core Components

1. **CircularBufferObject:** Main data structure containing:

- Read/write indices for buffer management
- Data buffer pointer and size
- Critical section for thread safety
- Overwrite configuration settings

2. **I/O Functions:**

- `CircularBufferRead()`: Reads data from the circular buffer
- `CircularBufferWrite()`: Writes data to the circular buffer
- `CircularBufferClose()`: Cleanup when closing the file descriptor

3. **Utility Functions:**

- `CircularBufferInit()`: Initializes the buffer and returns a file descriptor
- `CircularBufferFlushBytes()`: Flushes data up to a terminator byte
- `CircularBufferRawPrint()`: Debug function to display raw buffer contents

12.13.1.3.2 Buffer Management

The circular buffer uses two indices:

- **Read Index:** Points to the next byte to be read
 - **Write Index:** Points to the next position for writing
- Buffer States**:
- Empty: `readIndex == writeIndex`
 - Full: `(writeIndex + 1) % bufferSize == readIndex`

12.13.1.4 Configuration Options

12.13.1.4.1 Overwrite Mode

- **Enabled:** When buffer is full, old data is automatically overwritten
- **Disabled:** Write operations fail when buffer is full

12.13.1.4.2 Terminator Byte

When overwrite mode is enabled, you can specify a terminator byte (e.g., `'` for line-based data). When the buffer becomes full, data is flushed up to this terminator instead of byte-by-byte, making it ideal for line-oriented protocols.

12.13.1.5 Usage Example

```
// Initialize buffer
const uint32_t bufferSize = 256;
char dataBuffer[bufferSize];
CircularBufferObject circBuffer;

// Create file descriptor with overwrite enabled and newline terminator
int fd = CircularBufferInit(&circBuffer, dataBuffer, bufferSize, true, '\n');

// Write data
fdprintf(fd, "Hello World\n");

// Read data
char readBuffer[100];
int bytesRead = read(fd, readBuffer, sizeof(readBuffer));
```

12.13.1.6 Test Functions

The application includes two test functions:

12.13.1.6.1 TestBufferFull()

Demonstrates buffer overflow behavior by writing more data than the buffer can hold, then reading it back to show how the system handles full buffer conditions.

12.13.1.6.2 TestBufferLineMode()

Tests the terminator-based flushing feature by sending line-terminated messages and displaying the raw buffer contents to show how data is managed.

12.13.1.7 File Descriptor Status Management

The implementation uses NetBurner's file descriptor status functions:

- `SetDataAvail (fd) / ClrDataAvail (fd)`: Indicates data ready for reading
- `SetWriteAvail (fd) / ClrWriteAvail (fd)`: Indicates space available for writing
- `SetHaveError (fd) / ClrHaveError (fd)`: Error status management

These functions are interrupt-safe and integrate with NetBurner's I/O system.

12.13.1.8 Thread Safety

All buffer operations are protected by critical sections using NetBurner's `OS_CRIT` class, ensuring safe operation in multi-task environments.

12.13.1.9 Debug Features

- Compile-time debug flag (`circBufDebug`) for detailed operation logging
- Raw buffer print function for visualizing buffer contents
- Detailed status reporting during read/write operations

12.13.1.10 Configuration

Key configuration parameters in `main.cpp`:

- `dataBufferSize`: Buffer size (set to 10 for testing, increase for production)
- `enableOverwrite`: Set to `true` for automatic overwrite mode
- `overwriteTermByte`: Terminator character for intelligent flushing

12.13.1.11 Integration Notes

This example demonstrates the NetBurner Extra File Descriptor system defined in `iointernal.h`. The same pattern can be used to create custom I/O devices for:

- Serial communication protocols
- Custom data logging
- Inter-task communication buffers
- Hardware device abstraction

12.13.1.12 Files

- `main.cpp`: Main application and test functions
- `ExtraFdCircBuffer.h`: Header file with structure definitions
- `ExtraFdCircBuffer.cpp`: Implementation of circular buffer operations

12.14 FileDescriptor

These series of examples provide code utilizing file descriptors.
JSON Examples:

- [Extra FD Circular Buffer](#)
- [FDBuffer](#)
- [fdprintf - printf to file descriptor](#)
- [FdToNBString](#)

12.15 Extra FD Circular Buffer

12.15.1 NetBurner Extra File Descriptor Circular Buffer

12.15.1.1 Overview

This application demonstrates how to implement a custom I/O device using NetBurner's Extra File Descriptor system. The example creates a circular buffer that can be accessed using standard file operations like `read()` and `write()`, making it behave like any other file descriptor in the system. The Extra File Descriptors are defined in the '`iointernal.h`' header file.

Note

If you have an I/O device and wish to use interrupts, after adding the appropriate interrupt code that is specific to your device, you would use `USER_ENTER_CRITICAL` and `USER_EXIT_CRITICAL` to protect the internal data structures.

12.15.1.2 Key Features

- **Custom I/O Device:** Implements a circular buffer as a file descriptor
- **Thread-Safe Operations:** Uses critical sections for multi-task safety
- **Overwrite Mode:** Configurable buffer overflow handling
- **Terminator-Based Flushing:** Intelligent buffer management for line-based data
- **Standard File Operations:** Compatible with `read()`, `write()`, and `fdprintf()`

12.15.1.3 Architecture

12.15.1.3.1 Core Components

1. **CircularBufferObject**: Main data structure containing:

- Read/write indices for buffer management
- Data buffer pointer and size
- Critical section for thread safety
- Overwrite configuration settings

2. **I/O Functions**:

- `CircularBufferRead()`: Reads data from the circular buffer
- `CircularBufferWrite()`: Writes data to the circular buffer
- `CircularBufferClose()`: Cleanup when closing the file descriptor

3. **Utility Functions**:

- `CircularBufferInit()`: Initializes the buffer and returns a file descriptor
- `CircularBufferFlushBytes()`: Flushes data up to a terminator byte
- `CircularBufferRawPrint()`: Debug function to display raw buffer contents

12.15.1.3.2 Buffer Management

The circular buffer uses two indices:

- **Read Index**: Points to the next byte to be read
- **Write Index**: Points to the next position for writing

Buffer States**:

- **Empty**: `readIndex == writeIndex`
- **Full**: `(writeIndex + 1) % bufferSize == readIndex`

12.15.1.4 Configuration Options

12.15.1.4.1 Overwrite Mode

- **Enabled**: When buffer is full, old data is automatically overwritten
- **Disabled**: Write operations fail when buffer is full

12.15.1.4.2 Terminator Byte

When overwrite mode is enabled, you can specify a terminator byte (e.g., `'\n'` for line-based data). When the buffer becomes full, data is flushed up to this terminator instead of byte-by-byte, making it ideal for line-oriented protocols.

12.15.1.5 Usage Example

```
// Initialize buffer
const uint32_t bufferSize = 256;
char dataBuffer[bufferSize];
CircularBufferObject circBuffer;

// Create file descriptor with overwrite enabled and newline terminator
int fd = CircularBufferInit(&circBuffer, dataBuffer, bufferSize, true, '\n');

// Write data
fdprintf(fd, "Hello World\n");

// Read data
char readBuffer[100];
int bytesRead = read(fd, readBuffer, sizeof(readBuffer));
```

12.15.1.6 Test Functions

The application includes two test functions:

12.15.1.6.1 TestBufferFull()

Demonstrates buffer overflow behavior by writing more data than the buffer can hold, then reading it back to show how the system handles full buffer conditions.

12.15.1.6.2 TestBufferLineMode()

Tests the terminator-based flushing feature by sending line-terminated messages and displaying the raw buffer contents to show how data is managed.

12.15.1.7 File Descriptor Status Management

The implementation uses NetBurner's file descriptor status functions:

- `SetDataAvail (fd) / ClrDataAvail (fd)`: Indicates data ready for reading
- `SetWriteAvail (fd) / ClrWriteAvail (fd)`: Indicates space available for writing
- `SetHaveError (fd) / ClrHaveError (fd)`: Error status management

These functions are interrupt-safe and integrate with NetBurner's I/O system.

12.15.1.8 Thread Safety

All buffer operations are protected by critical sections using NetBurner's `OS_CRIT` class, ensuring safe operation in multi-task environments.

12.15.1.9 Debug Features

- Compile-time debug flag (`circBufDebug`) for detailed operation logging
- Raw buffer print function for visualizing buffer contents
- Detailed status reporting during read/write operations

12.15.1.10 Configuration

Key configuration parameters in `main.cpp`:

- `dataBufferSize`: Buffer size (set to 10 for testing, increase for production)
- `enableOverwrite`: Set to `true` for automatic overwrite mode
- `overwriteTermByte`: Terminator character for intelligent flushing

12.15.1.11 Integration Notes

This example demonstrates the NetBurner Extra File Descriptor system defined in `iointernal.h`. The same pattern can be used to create custom I/O devices for:

- Serial communication protocols
- Custom data logging
- Inter-task communication buffers
- Hardware device abstraction

12.15.1.12 Files

- `main.cpp`: Main application and test functions
- `ExtraFdCircBuffer.h`: Header file with structure definitions
- `ExtraFdCircBuffer.cpp`: Implementation of circular buffer operations

12.15.2 FDBuffer

12.15.2.1 NetBurner FDBuffer Example

12.15.2.1.1 Overview

This example demonstrates how to use the NetBurner FDBuffer class to capture and manipulate data from file descriptors (fd) that would normally be used for direct I/O operations. The FDBuffer class allows you to intercept data streams and store them in a local buffer for debugging, logging, or further processing.

12.15.2.1.2 Purpose

Many NetBurner libraries utilize file descriptors for data input/output transmission, including:

- TCP connections
- WebSockets
- HTTP communications
- FTP transfers
- Email operations
- Serial communications
- Configuration system

This example shows how to capture I/O data from an fd to a buffer instead of using the fd as originally intended by a library. This technique is particularly useful for:

- Debugging network communications
- Logging system data
- Capturing configuration information
- Storing data for later transmission or processing

12.15.2.1.3 How It Works

The application demonstrates capturing the system's configuration JSON object using an FDBuffer:

1. **Initialization**: The application initializes the network stack and starts the HTTP server
2. **Buffer Setup**: Creates an FDBuffer object and associated data buffer
3. **Data Capture**: Uses the config system to write JSON data to the FDBuffer instead of directly to a socket or serial port
4. **Data Retrieval**: Reads the captured data from the FDBuffer into a local buffer
5. **Data Output**: Displays the captured JSON configuration data via serial output

12.15.2.1.4 Key Components

12.15.2.1.4.1 FDBuffer Class

- Defined in `fd_adapter.h`
- Associates a buffer with a file descriptor
- Supports standard file descriptor functions: `read()`, `write()`, `close()`
- By default, can dynamically allocate up to 20 pool buffers
- Each pool buffer stores `ETHER_BUFFER_SIZE` (1548) bytes
- Total default storage capacity: 30,960 bytes
- Storage limit can be increased by modifying `MAX_FDBUFFER_FIFO_BUFFERS` macro

12.15.2.1.4.2 Application Features

- **Pretty Print Option:** Toggle between formatted JSON (with indentation) and compact single-line output
- **Buffer Management:** Proper allocation, usage, and cleanup of file descriptors
- **Error Handling:** Checks buffer size constraints and provides appropriate error messages
- **Dual Output Methods:** Demonstrates both direct fd writing and printf-style output

12.15.2.1.5 Code Structure

12.15.2.1.5.1 Main Functions

1. **UserMain():** Primary application entry point
 - Initializes network stack and HTTP server
 - Sets up FDBuffer and data buffer
 - Captures config JSON data
 - Outputs captured data to serial port
2. **Buffer Operations:**
 - `myFdbufferObject.GetActiveFD()`: Allocates file descriptor
 - `root.RenderToFd()`: Writes config JSON to FDBuffer
 - `myFdbufferObject.SpaceUsed()`: Returns stored data size
 - `read()`: Reads data from FDBuffer to local buffer
 - `close()`: Releases file descriptor back to pool

12.15.2.1.5.2 Configuration Variables

- `dataBufferSize`: Size of local data buffer (4096 bytes)
- `prettyPrint`: Boolean flag for JSON formatting style

12.15.2.1.6 Usage Examples

12.15.2.1.6.1 Basic Usage

The application automatically captures and displays the system configuration JSON when started.

12.15.2.1.6.2 Alternative Streaming Method

The code includes an alternative approach that streams directly to stdout without using a local buffer:

```
int stdout_fd = CurrentStdioFD(1);
myFdbufferObject.StreamTo(stdout_fd);
```

12.15.2.1.7 Build Requirements

- NetBurner NNDK (Network Development Kit)
- Compatible NetBurner hardware platform
- Standard NetBurner libraries and headers

12.15.2.1.8 Key Headers Required

- `predef.h`: Platform definitions
- `config_obj.h`: Configuration system
- `fdprintf.h`: File descriptor printf functions
- `iointernal.h`: I/O internal functions

12.15.2.1.9 Error Handling

The application includes error checking for:

- Buffer size limitations
- File descriptor allocation
- Data read/write operations
- Network initialization timeouts

12.15.2.1.10 Best Practices Demonstrated

1. **Memory Management:** Proper clearing of buffers and FDBuffer objects
2. **Resource Cleanup:** Closing file descriptors after use to return them to the pool
3. **Error Checking:** Validating buffer sizes and operation return values
4. **Documentation:** Clear comments explaining each operation

12.15.2.1.11 Potential Applications

This technique can be extended for:

- Network packet inspection
- Configuration backup to persistent storage (flash, SD card)
- Remote data transmission (email, syslog, FTP)
- Real-time debugging and monitoring
- Data logging and analysis

12.15.3 fdprintf - printf to file descriptor

12.15.3.1 fdprintf Example Application

12.15.3.1.1 Overview

This application demonstrates the use of file descriptors for serial port communication using the NetBurner RTOS platform. It showcases how to close, open, and print to file descriptors, specifically focusing on serial port operations with formatted output.

Note

While this example uses a serial file descriptor, `fdprintf()` can be used with any file descriptor, such as dynamic web page output. A serial port is used here for simplicity, but the web examples make extensive use of `fdprintf()`.

12.15.3.1.2 Functionality

The application performs the following operations:

1. **Serial Port Initialization:** Configures and opens serial ports for data communication and debug output
2. **File Descriptor Management:** Demonstrates proper handling of file descriptors for serial communication
3. **Formatted Output:** Uses `fdprintf()` to send formatted data to serial ports
4. **Network Integration:** Initializes HTTP server and waits for network connectivity
5. **Continuous Operation:** Runs in a loop, outputting timestamp data every second

12.15.3.1.3 Configuration

12.15.3.1.3.1 Serial Port Settings

- **Data Serial Port:** Port 0
- **Debug Serial Port:** Port 0 (shared with data port in this example)
- **Baud Rate:** 115200 bps
- **Data Bits:** 8
- **Stop Bits:** 1
- **Parity:** None

12.15.3.1.3.2 Network Settings

- **Network Timeout:** 5 seconds wait for active network connection
- **HTTP Server:** Enabled

12.15.3.1.4 Key Features

12.15.3.1.4.1 File Descriptor Operations

- Opens serial ports using `OpenSerial()`
- Closes existing serial connections with `SerialClose()`
- Manages separate file descriptors for data and debug output
- Replaces standard I/O streams (stdin, stdout, stderr) with custom file descriptors

12.15.3.1.4.2 Output Functionality

- Sends formatted timestamp data every second
- Uses `fdprintf()` for formatted output to specific file descriptors
- Outputs current system time in seconds with carriage return and line feed

12.15.3.1.4.3 System Integration

- Initializes NetBurner system components
- Enables system diagnostics (recommended to disable for production)
- Integrates with HTTP server functionality
- Waits for network connectivity before main operation

12.15.3.1.5 Code Structure

12.15.3.1.5.1 Main Components

1. **Global Variables:**

- `fddebug`: File descriptor for debug output
- `fdserial`: File descriptor for serial data I/O

2. **InitializeSerialPorts():**

- Configures serial port settings
- Opens communication channels
- Redirects standard I/O streams

3. **UserMain():**

- Main application entry point
- Initializes system components
- Runs continuous operation loop

12.15.3.1.6 Usage

The application automatically:

1. Initializes the NetBurner system
2. Starts HTTP server
3. Waits for network connectivity
4. Configures serial ports
5. Begins continuous timestamp output

12.15.3.1.7 Output Format

The application outputs timestamps in the following format:

`Secs: [timestamp]`

Where `[timestamp]` is the current system time in seconds since startup.

12.15.3.1.8 Development Notes

- System diagnostics are enabled by default (consider disabling for production)
- Both data and debug ports use the same serial port (Port 0) in this configuration
- The application demonstrates proper file descriptor management for embedded systems
- Standard I/O streams are redirected to custom file descriptors for better control

12.15.4 FdToNBString

12.15.4.1 File Descriptor to NBString Example

12.15.4.1.1 Overview

This NetBurner application demonstrates how to capture file descriptor I/O data to an [NBString](#) object instead of using the file descriptor as originally intended by a library. This technique is useful for logging, debugging, or printing data to serial output.

The example utilizes the `NBStringBuilder` class which associates a buffer with a file descriptor that can then be used with API's that utilize file descriptors. The `NBStringBuilder` class is defined in [nbstring.h](#). The `NBStringBuilder` class supports the use of file descriptor functions defined in [iosys.h](#) such as: [read\(\)](#), [write\(\)](#), and [close\(\)](#).

12.15.4.1.2 Purpose

Many NetBurner libraries utilize file descriptors (fd) for data input/output transmission, including:

- TCP
- WebSockets
- HTTP
- FTP
- Serial communication
- Configuration system

This example shows how to redirect file descriptor data to a local [NSString](#) variable for further processing.

12.15.4.1.3 Key Components

12.15.4.1.3.1 NSStringBuilder Class

- Associates a buffer with a file descriptor
- Defined in [nbsstring.h](#)
- Supports standard file descriptor functions from [iosys.h](#) (read, write, close)
- Enables capturing fd data to [NSString](#) objects

12.15.4.1.3.2 System Diagnostics Integration

- Uses [EnableSystemDiagnostics\(\)](#) to maintain a diagnostics data set as JSON
- Demonstrates writing diagnostics data to an [NSString](#) instead of TCP socket or serial
- Default diagnostics data set is approximately 2.6KB but can grow based on:
 - Number of tasks
 - Socket usage
 - User-defined variables in diagnostic set

12.15.4.1.4 Application Flow

1. Initialization

- Network stack initialization
- Enable system diagnostics
- Start HTTP server on port 80
- Wait for active network connection

2. User Interaction

- Prompts user to press any key to print system diagnostics
- Waits for user input via [getchar\(\)](#)

3. Data Capture Process

- Reserves 8KB buffer space (overkill for default diagnostics but safe)
- Allocates file descriptor from fd pool
- Writes system diagnostics data to NSStringBuilder object
- Moves data from NSStringBuilder to [NSString](#) variable
- Closes the allocated file descriptor

4. Output

- Prints captured diagnostics data to serial output
- Data is now available for other uses (storage, transmission, etc.)

12.15.4.1.5 Memory Considerations

- **Buffer Size:** 8192 bytes (8KB) reserved for diagnostics data
- **Default Size:** ~2.6KB for basic diagnostics
- **Growth Factors:** Additional tasks, sockets, and user variables increase size

12.15.4.1.6 Potential Use Cases

Once captured in an `NBString`, the diagnostics data can be:

- Stored to persistent storage (flash memory, SD card)
- Transmitted remotely (email, syslog, FTP)
- Logged for debugging purposes
- Processed or filtered before use

12.15.4.1.7 Key APIs Used

- `EnableSystemDiagnostics()` - Enables system diagnostic data collection
- `NBStringBuilder::reservespace()` - Allocates buffer space
- `NBStringBuilder::GetActiveFD()` - Gets file descriptor for the builder
- `pDiagServer()` - Writes diagnostics data to specified file descriptor
- `NBStringBuilder::moveTo()` - Transfers data to `NBString` object
- `close()` - Closes file descriptor

12.15.4.1.8 Network Features

- HTTP server enabled on default port 80
- Waits up to 5 seconds for active network connection
- Network stack fully initialized before main loop

12.15.4.1.9 Usage

1. Build and deploy the application to NetBurner hardware
2. Connect to the device via serial terminal
3. Press any key when prompted to capture and display system diagnostics
4. The diagnostics JSON data will be printed to the serial output

12.16 FTP

FTP can be run with or without a file system. The examples in this section do not use a file system. These are useful for applications such as:

- Simple download of data as a file by a FTP client, such as measurement readings.
- Updating an application image.
- Viewing web or interface data in a file format.

Since there is no file system, these examples use callback functions to handle the various FTP transactions, such as reading, writing and listing a file. If you do not wish to use the callback scheme, please refer to the file system examples described below.

Examples of FTP using a file system take advantage of the Embedded Flash File System (EFFS). These examples are located in the "\nburn\examples\EFFS" folder. There are two types of file systems:

- EFFS FAT (File Allocation Table), which is a FAT32 file system used by external flash cards.
- EFFS STD (Standard File System), which is a custom file system that runs in the onboard flash memory of the NetBurner device.

FTP in this section:

- [FTP Client](#)
- [Display HTML Files \(FTPD\)](#)
- [Simple FTP Server \(FTPD\)](#)

12.16.1 FTP Client

12.16.1.1 FTP Client Example

A NetBurner embedded FTP client application that demonstrates basic FTP operations including connecting to a server, directory navigation, file listing, file creation, and file retrieval.

Note

For FTP and SFTP using the EFFS file systems, refer to examples in the \nburn\examples\EFFS folder

12.16.1.1.1 Overview

This application connects to an FTP server and performs the following operations:

1. Establishes an FTP connection to a specified server
2. Changes to a remote directory called "test1"
3. Retrieves and displays the directory listing
4. Creates a sample file named "WriteTest.txt" with test content
5. Reads back the created file and displays its contents

12.16.1.1.2 Features

- FTP server connection with authentication
- Directory navigation and listing
- File upload (creation) capability
- File download and content display
- Error handling and command result validation
- Timeout handling for network operations

12.16.1.1.3 Hardware Requirements

- NetBurner embedded device with network connectivity
- Active network connection

12.16.1.1.4 Software Requirements

- NetBurner RTOS (nbrtos)
- NetBurner system libraries including:
 - `init.h`
 - `system.h`
 - `ftp.h`
 - `string.h`
 - `iosys.h`
 - `netinterface.h`

12.16.1.1.5 Setup Requirements

Before running this application, ensure the following prerequisites are met:

1. **FTP Server Access:** You need access to a functioning FTP server
2. **Authentication:** Valid username and password for the FTP server
3. **Directory Structure:** A directory named "test1" must exist on the FTP server
4. **Permissions:** Write permissions must be granted to the "test1" directory
5. **Configuration:** Update the configuration defines in `main.cpp` (see Configuration section)

12.16.1.1.6 Configuration

Before compiling and running the application, modify the following defines in `main.cpp` to match your FTP server setup:

```
#define SERVER_IP_ADDR "10.1.1.193" // Replace with your FTP server IP
#define FTP_PORT 21 // Standard FTP port (usually 21)
#define USER "username" // Replace with your FTP username
#define PASS "password" // Replace with your FTP password
#define WRITE_TEST_FILE "WriteTest.txt" // Test file name
```

12.16.1.1.7 Application Flow

1. **Initialization:** The application initializes the NetBurner system and waits for an active network connection
2. **FTP Connection:** Establishes a connection to the FTP server using the configured credentials
3. **Directory Change:** Changes to the "test1" directory on the remote server
4. **Directory Listing:** Retrieves and displays the contents of the "test1" directory
5. **File Creation:** Creates a new file "WriteTest.txt" with sample content:
 - "This is a test file"
 - "This is line 2 of the test file"
 - "Last Line"
6. **File Retrieval:** Downloads and displays the contents of the newly created file
7. **Cleanup:** Closes the FTP session and enters an infinite loop

12.16.1.1.8 Key Functions

12.16.1.1.8.1 ShowFileContents(int fdr)

Reads data from a file descriptor and displays it to stdout (debug serial port). Handles the data stream in 255-byte chunks with a 20-second timeout.

12.16.1.1.8.2 UserMain(void *pd)

The main application function that orchestrates all FTP operations. Includes comprehensive error handling and status reporting.

12.16.1.1.9 Error Handling

The application includes robust error handling for:

- FTP session initialization failures
- Directory change failures
- File listing failures
- File creation failures
- File retrieval failures
- Command result validation (expects FTP code 226 for successful operations)

12.16.1.1.10 Output

The application provides detailed console output including:

- Connection status messages
- Directory listing contents
- File creation confirmation
- File content display
- Error messages with specific error codes

12.16.1.1.11 Timeout Configuration

All FTP operations use a 5-second timeout (5 * TICKS_PER_SECOND) to prevent hanging on network issues.

12.16.1.1.12 Production Notes

- The application enables system diagnostics which should be removed for production code
- Consider implementing more sophisticated error recovery mechanisms for production use
- Hardcoded credentials should be replaced with secure configuration methods in production

12.16.1.1.13 Compilation

Compile this application using the NetBurner development tools with the standard NetBurner RTOS libraries linked.

12.16.1.1.14 Troubleshooting

Common issues and solutions:

1. **Connection Failures:** Verify IP address, port, username, and password
2. **Directory Not Found:** Ensure "test1" directory exists on the FTP server
3. **Permission Denied:** Verify write permissions to the target directory
4. **Network Issues:** Check network connectivity and firewall settings
5. **Timeout Errors:** Consider increasing timeout values for slow network connections

12.16.2 Display HTML Files (FTPD)

12.16.2.1 FTPD HTML Expose Application

12.16.2.1.1 Overview

This application creates a read-only FTP server that exposes HTML files from an embedded system as a browsable file system. It's designed for embedded systems using a network stack and provides both web server functionality and FTP access to HTML content.

Note

For FTP and SFTP using the EDFS file systems, refer to examples in the `\nburn\examples\EDFS` folder

12.16.2.1.2 Features

- **FTP Server:** Runs on port 21 with read-only access to HTML files
- **Web Server:** HTTP server running on port 80
- **Network Integration:** Uses [DHCP](#) for automatic IP configuration
- **File System Emulation:** Creates a virtual directory tree from compiled HTML files
- **Cross-Platform Access:** Files accessible via web browsers and FTP clients

12.16.2.1.3 Application Components

12.16.2.1.3.1 Core Functionality

The application initializes a network stack and starts two main services:

1. **HTTP Web Server** - Provides web-based access and status information
2. **FTP Server** - Exposes HTML files as a read-only file system

12.16.2.1.3.2 File System Structure

The application builds a virtual file system from HTML files that are compiled into the application:

- Creates a tree structure with directories and files
- Supports nested directory hierarchies
- File metadata includes size and basic permissions
- Directory navigation follows standard FTP conventions

12.16.2.1.3.3 Key Functions

Network Services

- `UserMain()` - Main application entry point, initializes services
- `webDisplayFtpLink()` - Displays FTP access URL on web pages

Directory Tree Management

- `BuildDirTree()` - Constructs file system tree from HTML data
- `GetDir()` - Locates directories in the virtual file system
- `getDirString()` - Formats file/directory information for FTP clients

FTP Server Callbacks

- `FTPD_ListFile()` - Lists files in a directory
- `FTPD_ListSubDirectories()` - Lists subdirectories
- `FTPD_DirectoryExists()` - Checks if directory exists
- `FTPD_SendFileToClient()` - Transfers files to FTP clients
- `FTPD_FileExists()` - Verifies file existence
- `FTPDSessionStart()` - Handles FTP client connections
- `FTPDSessionEnd()` - Manages session cleanup

12.16.2.1.4 Usage

12.16.2.1.4.1 Accessing Files

1. **Via Web Browser:** Navigate to the device's IP address
2. **Via FTP Client:** Connect to `ftp://[device-ip]` using any FTP client
3. **FTP URL:** The application displays the FTP URL format as `ftp://[ip-address]`

12.16.2.1.4.2 Browser Compatibility

Starting in 2020, some web browsers have begun blocking direct FTP access. In such cases:

- Use dedicated FTP client applications such as FileZilla
- Access files through the web server interface instead

12.16.2.1.4.3 Authentication

The current implementation accepts all FTP connections without password verification. For production use, implement proper authentication in `FTPDSessionStart()`.

12.16.2.1.5 Technical Details

12.16.2.1.5.1 Dependencies

- Network stack initialization (`init.h`)
- FTP server library (`ftpd.h`)
- HTML file system (`htmlfiles.h`)
- Network interface management (`netinterface.h`)

12.16.2.1.5.2 Memory Management

- Dynamic allocation for directory and file structures
- Linked list implementation for file system hierarchy
- Cleanup handled automatically on application termination

12.16.2.1.5.3 Configuration

- FTP Server Port: 21
- HTTP Server Port: 80 (default)
- Task Priority: `MAIN_PRIO - 1` (higher than main task)
- Network Timeout: 5 seconds for `DHCP`

12.16.2.1.5.4 Read-Only Operations

The following FTP operations are supported:

- Directory listing (LIST, NLST)
- File download (RETR)
- Directory navigation (CWD, PWD)
- File existence checking

12.16.2.1.5.5 Unsupported Operations

These FTP operations always return failure:

- File upload (STOR)
- File deletion (DELE)
- Directory creation (MKD)
- Directory deletion (RMD)
- File renaming (RNFR/RNTO)

12.16.2.1.6 System Requirements

- Embedded system with network capability
- Network stack supporting TCP/IP
- Sufficient RAM for file system tree structure
- HTML files compiled into application data

12.16.2.1.7 Development Notes

This is an advanced application requiring knowledge of:

- FTP protocol implementation
- Embedded network programming
- File system navigation concepts
- HTML file management in embedded systems

The complexity primarily involves managing virtual directory structures and handling FTP client requests for file navigation and display.

12.16.2.1.8 Status Output

The application provides diagnostic information via serial port:

- Network initialization status
- FTP server startup confirmation
- IP address assignment
- Session start/end notifications
- System diagnostics (should be disabled for production)

12.16.2.1.9 Example Output

```
Started FTP Server
You can use a web browser to view files at URL "ftp://192.168.1.100"
Building the tree...
Ending session 1
```

12.16.3 Simple FTP Server (FTPD)

12.16.3.1 Simple FTP Server (FTPD) - Trivial Implementation

12.16.3.1.1 Overview

This is a lightweight FTP server implementation designed for NetBurner embedded systems. Unlike traditional FTP servers that rely on file systems, this trivial implementation uses callback functions to handle FTP operations, making it ideal for downloading system information and uploading configuration data without requiring persistent storage.

Note

For FTP and SFTP using the EDFS file systems, refer to examples in the `\nburn\examples\EDFS` folder

12.16.3.1.2 Features

- **No File System Required:** Operates entirely through callback functions
- **Lightweight Design:** Minimal memory footprint suitable for embedded systems
- **Debug Output:** Status messages and data are sent to the debug serial port
- **Fixed File Operations:**
 - Download: Single file named `ReadFile.txt` with customizable content
 - Upload: Accepts ASCII text files named `WriteFile.txt` (displayed via serial port)

12.16.3.1.3 Technical Details

12.16.3.1.3.1 Application Configuration

- **Application Name:** FTPD Trivial
- **FTP Port:** 21 (standard FTP port)
- **Priority:** MAIN_PRIO - 1

12.16.3.1.3.2 Supported Operations

File Download

- **Available File:** `ReadFile.txt`
- **Content:** Fixed string "This will be the contents of `ReadFile.txt`"
- **Customization:** Content can be modified to include dynamic system information

File Upload

- **Accepted File:** `WriteFile.txt` (ASCII text only)
- **Storage:** File content is not stored in memory
- **Output:** Uploaded content is displayed on the debug serial port

Directory Operations

- **Listing:** Returns single file entry for `ReadFile.txt`
- **Permissions:** Simulated as `rw-rw-rw-`
- **Timestamps:** Fixed to "Jan 01 00:00"

12.16.3.1.3.3 Authentication

- **Method:** Basic callback-based authentication
- **Default Behavior:** Accepts any username/password combination
- **Customization:** Authentication logic can be modified in `FTPDSessionStart()`

12.16.3.1.4 Code Structure

12.16.3.1.4.1 Main Functions

- `UserMain()`: Application entry point, initializes and starts FTP server
- `FTPD_ListFile()`: Handles FTP LIST command
- `FTPD_SendFileToClient()`: Manages file download requests
- `FTPD_GetFileFromClient()`: Processes file upload requests
- `FTPD_FileExists()`: Checks file availability
- `FTPD_AbleToCreateFile()`: Validates file creation permissions

12.16.3.1.4.2 Helper Functions

- `getdirstring()`: Formats directory listing entries
- `ShowFileContents()`: Displays uploaded file content to serial port

12.16.3.1.4.3 Unsupported Operations

The following FTP operations return `FTPD_FAIL`:

- Directory creation/deletion
- File deletion
- File renaming
- File size queries
- Subdirectory operations

12.16.3.1.5 Usage

12.16.3.1.5.1 Compilation and Deployment

1. Include required NetBurner headers: `ftpd.h`, `init.h`, `iosys.h`, `nbrtos.h`
2. Compile with NetBurner development tools
3. Deploy to NetBurner device

12.16.3.1.5.2 Testing

1. Connect FTP client to device IP address on port 21
2. Use any username/password (authentication passes by default)
3. Test operations:
 - `ls` or `dir`: Lists `ReadFile.txt`
 - `get ReadFile.txt`: Downloads the fixed content file
 - `put WriteFile.txt`: Uploads file (content displayed on serial port)

12.16.3.1.5.3 Serial Port Monitoring

Use a terminal application like `mtty` to monitor:

- Server startup messages
- File upload content
- Debug information

12.16.3.1.6 Customization

12.16.3.1.6.1 Dynamic Content

Modify the `READ_FILEDATA` constant or the `FTPD_SendFileToClient()` function to provide dynamic system information such as:

- Device status
- Sensor readings
- Configuration parameters
- Log data

12.16.3.1.6.2 Authentication

Enhance the `FTPD_SessionStart()` function to implement:

- User validation against stored credentials
- IP address filtering
- Session logging

12.16.3.1.6.3 File Operations

Extend functionality by:

- Supporting multiple downloadable files
- Implementing file storage for uploads
- Adding configuration file handling

12.16.3.1.7 Dependencies

- NetBurner RTOS
- NetBurner FTP daemon library
- Standard C library functions

12.16.3.1.8 Notes

- This implementation is designed for development and lightweight applications
- Remove `EnableSystemDiagnostics()` call for production deployment
- For file system-based FTP implementations, refer to examples in the `\nburn\examples\FFS` folder
- All FTP operations are logged to the debug serial port for monitoring and troubleshooting

12.16.3.1.9 Related Examples

For more advanced FTP server implementations using the EFS file system, see the NetBurner examples in the EFS folder which provide full file system integration and persistent storage capabilities.

12.17 GDB Debugger

12.17.1 NetBurner GDB Demo Application

12.17.1.1 Overview

This is a simple demonstration application designed to run on NetBurner network-enabled hardware platforms. The application serves as a basic example for testing and using the GDB debugger with NetBurner devices. Debugging can be done two ways:

1. NBEclipse: Debugging is done through the graphical IDE
2. Command Line: Debugging is done from a command prompt using commandline GDP on port 2159

12.17.1.2 Application Description

The GDB Demo application provides a minimal but functional embedded program that:

- Initializes the NetBurner network stack
- Starts a web server on the default port (80)
- Waits for network connectivity through [DHCP](#)
- Continuously displays runtime statistics to the serial console

12.17.1.3 Features

- **Network Initialization:** Automatically configures network settings
- **Web Server:** Built-in HTTP server for web-based interaction
- **System Diagnostics:** Enabled for debugging purposes
- **Runtime Monitoring:** Displays uptime information every 250ms
- **Serial Output:** Provides real-time feedback via default serial port

12.17.1.4 Hardware Requirements

- NetBurner network-enabled hardware platform
- Serial connection for console output
- Network connection (Ethernet)

12.17.1.5 Software Requirements

- NetBurner development environment
- GDB debugger support
- Compatible toolchain for NetBurner platforms

12.17.1.6 Application Behavior

Upon startup, the application will:

1. Initialize the network stack
2. Enable system diagnostics
3. Start the HTTP web server
4. Wait up to 5 seconds for network connectivity
5. Display "GDB Demo Application started" message
6. Enter main loop displaying runtime statistics

12.17.1.7 Console Output

The application continuously outputs the following information:

```
GDB Demo Application started  
Seconds since boot: X, Ticks since boot: Y
```

Where:

- X represents the number of seconds since the application started
- Y represents the system tick count since boot

12.17.1.8 Usage

This application is primarily intended for:

- **Debugging Practice:** Learning to use GDB with NetBurner hardware
- **System Testing:** Verifying basic network and system functionality
- **Development Reference:** Example of minimal NetBurner application structure

12.17.1.9 Development Notes

- System diagnostics are enabled for debugging purposes
- Consider disabling diagnostics for production deployment
- The application runs in an infinite loop for continuous monitoring
- Output update rate is 4 times per second (250ms intervals)

12.17.1.10 File Structure

- `main.cpp` - Main application source code
- `ReadMe.txt` - Original documentation (Doxygen format)

12.17.1.11 Building and Deployment

Follow standard NetBurner development procedures to build and deploy this application to your target hardware platform.

12.18 General Purpose I/O

These general GPIO examples apply to multiple NetBurner platforms. Please refer to the "Example Applications > Platform Specific" section of this document for additional GPIO examples specific to your particular NetBurner platform.

Examples in this section:

- [GPIO Blink](#)
- [General Purpose I/O Service](#)

12.18.1 GPIO Blink

12.18.1.1 GPIO Blink Application

12.18.1.1.1 Overview

The GPIO Blink application is a simple demonstration program that showcases the basic operation of GPIO pins using the NetBurner [PinIO](#) class. This example blinks one of the indicator LEDs on the device carrier board in a continuous loop.

Note

This example uses a bsp include file for LED pin mapping: `#include <bsp_devboard.h> // A project header that contains the mapping of LED pins`

12.18.1.1.2 Application Details

Application Name:** Blink

Purpose:** Basic GPIO pin control demonstration

Target Platform:** NetBurner embedded systems

12.18.1.1.3 Functionality

The application performs the following operations:

1. **System Initialization:** Initializes the NetBurner system and enables system diagnostics
2. **Status Output:** Displays the application name and NNDK revision information via serial output
3. **GPIO Configuration:** Configures LED1 pin as a GPIO output
4. **LED Blinking Loop:** Continuously blinks the LED with the following pattern:
 - Turn LED ON (drive pin LOW) for 1 second
 - Turn LED OFF (drive pin HIGH) for 1 second
 - Repeat indefinitely

12.18.1.1.4 Code Structure

12.18.1.1.4.1 Key Components

- **System Headers:** Includes necessary NetBurner system libraries for initialization, timing, and GPIO control
- **Pin Configuration:** Uses the [PinIO](#) class to configure and control GPIO pins
- **BSP Integration:** Leverages board support package definitions for LED pin mapping
- **RTOS Integration:** Utilizes NetBurner RTOS timing functions for precise delays

12.18.1.1.4.2 Main Functions

- `UserMain()`: The primary application entry point that handles all initialization and the main execution loop
- LED control through direct pin assignment operations
- System diagnostic output for debugging and monitoring

12.18.1.1.5 Hardware Requirements

- NetBurner development board with carrier board
- At least one indicator LED (LED1) available on the carrier board
- Serial connection for viewing diagnostic output

12.18.1.1.6 Technical Notes

This example is intentionally simplified to demonstrate basic GPIO concepts. For more advanced GPIO usage, refer to:

- **GpioServer Example:** More comprehensive GPIO functionality demonstration
- **PinIO Documentation:** Complete API reference for the [PinIO](#) class
- **Platform Specific Examples:** GPIO examples tailored to specific NetBurner platforms

12.18.1.1.7 Build Requirements

- NetBurner NNDK (NetBurner Network Development Kit)
- Compatible NetBurner hardware platform
- Standard NetBurner development environment

12.18.1.1.8 Usage

1. Compile and load the application onto your NetBurner device
2. Connect a serial terminal to view system output
3. Observe LED1 blinking at 1-second intervals
4. Monitor serial output for application name and NNDK version information

12.18.1.1.9 Expected Output

Serial console will display:

```
Application: Blink  
NNDK Revision: [Current NNDK Version]
```

The LED1 indicator will blink continuously with a 1-second on/off cycle.

12.18.1.1.10 Learning Objectives

This example teaches:

- Basic GPIO pin configuration using [PinIO](#) class
- Simple output control for driving LEDs
- NetBurner system initialization procedures
- RTOS timing and delay functions
- Board support package integration

12.18.1.1.11 Further Development

While this example covers basic GPIO output, the [PinIO](#) class supports many additional features including input reading, interrupt handling, and advanced pin functions. Consult the NetBurner documentation for expanded GPIO capabilities and platform-specific implementations. Please refer to the "Example Applications > Platform Specific" section of the NetBurner Documentation for additional GPIO examples specific to your particular NetBurner platform.

12.18.2 General Purpose I/O Service

Example to control GPIO pins through command interface.

12.18.2.1 GPIO Service Application

A NetBurner embedded system application for controlling General Purpose I/O (GPIO) pins through a command-line interface.

Note

For platform-specific GPIO examples and advanced usage, refer to the "Example Applications > Platform Specific" section in the NetBurner Documentation.

12.18.2.1.1 Overview

This application provides a serial command interface for configuring and controlling GPIO signals on NetBurner modules. It sets up a command parser on the default serial port, allowing users to interact with GPIO pins in real-time through simple text commands.

12.18.2.1.2 Features

- Real-time GPIO pin control via serial terminal
- Support for multiple NetBurner platforms with automatic connector detection
- Command history with navigation (Ctrl+D, Ctrl+U, Ctrl+R)
- Interactive help system
- Web server integration for network diagnostics

12.18.2.1.3 Supported Operations

The application supports the following GPIO operations:

- **HI** - Set a pin to logic high (3.3V/5V)
- **LO** - Set a pin to logic low (0V)
- **TGL** - Toggle a pin's current state
- **OUT** - Configure a pin as an output
- **IN** - Configure a pin as an input (high impedance)
- **RD** - Read the current state of a pin
- **HELP** - Display available commands

12.18.2.1.4 Platform Support

The application automatically adapts to different NetBurner platforms:

- **MODM7AE70**: Uses P2 connector with [PinIO::PIN_FN_OUT](#) function
- **NANO54415 & SB800EX**: Uses main connector (Pins) with PIN_GPIO function
- **Other platforms**: Uses J2 connector with PIN_GPIO function

12.18.2.1.5 Hardware Requirements

- NetBurner development module
- Serial terminal connection (debug/status port)
- GPIO-capable pins on the target connector

Warning

Not all pins on every NetBurner device can function as GPIO. Consult your specific device datasheet for GPIO capabilities and pin assignments.

12.18.2.1.6 Software Requirements

- NetBurner NNDK (NetBurner Network Development Kit)
- MTTTY or compatible serial terminal software
- Network connection for web server functionality (optional)

12.18.2.1.7 Building and Installation

1. Build the project using the NetBurner development environment
2. Load the compiled application onto your NetBurner device
3. Connect to the device's debug/status port using a serial terminal
4. The application will automatically start and display the command prompt

12.18.2.1.8 Usage Examples

12.18.2.1.8.1 Basic Pin Control

```
> HI 5          # Set pin 5 high
P2[5]_out: 1

> LO 3          # Set pin 3 low
P2[3]_out: 0

> TGL 7         # Toggle pin 7
P2[7]_out: 1
```

12.18.2.1.8.2 Pin Configuration

```
> OUT 2         # Configure pin 2 as output
> IN 4          # Configure pin 4 as input
> RD 4          # Read pin 4 state
P2[4]_in: 0
```

12.18.2.1.8.3 Interactive Features

- Press **Enter** on empty line to repeat last command
- Use **Ctrl+D** to navigate to previous command in history
- Use **Ctrl+U** to navigate to next command in history
- Use **Ctrl+R** to recall and execute previous command
- Use **Backspace** to edit current command

12.18.2.1.9 Command Reference

Command	Syntax	Description
HI	HI <pin>	Set specified pin to logic high
LO	LO <pin>	Set specified pin to logic low
TGL	TGL <pin>	Toggle pin state (repeatable with Enter)
OUT	OUT <pin>	Configure pin as output driver
IN	IN <pin>	Configure pin as high-impedance input
RD	RD <pin>	Read pin state (repeatable with Enter)
HELP	HELP	Display command help

12.18.2.1.10 Application Architecture

12.18.2.1.10.1 Key Components

- **Command Parser:** Processes serial input and maps to function calls
- **GPIO Abstraction:** Platform-independent pin control interface
- **Command History:** Circular buffer storing recent commands
- **Network Stack:** Built-in web server for diagnostics

12.18.2.1.10.2 Memory Usage

- Command buffers: 5 buffers x 80 characters each
- Read buffer: 256 bytes
- Minimal stack usage for embedded operation

12.18.2.1.11 Network Features

The application includes built-in network functionality:

- HTTP web server on port 80
- **DHCP** client for automatic IP assignment
- System diagnostics accessible via web interface
- 5-second network initialization timeout

12.18.2.1.12 Error Handling

- Graceful handling of invalid commands with "Unrecognized command" message
- Parameter validation for pin numbers
- Safe defaults for missing parameters
- Non-blocking serial input processing

12.18.2.1.13 Development Notes

- All commands are case-insensitive (automatically converted to uppercase)
- Pin numbers are parsed as decimal integers
- The application runs in an infinite loop with 1-second intervals
- Platform-specific macros handle connector and function differences

12.19 I2C

The `examples/I2C` folder contains platform-agnostic [I2C](#) examples for the new [Wire Interface](#).

- Wire
 - [PAC193X](#)
 - [PicKit Serial I2C Demo Board](#)
 - [RV3032 I2C Real-Time Clock](#)
 - [Wire Address Scan](#)
 - SparkFun Qwiic - examples for SparkFun Qwiic devices (compatible with the SOMRT1061 Development board's Qwiic connector)
 - * [SparkFun Qwiic Alphanumeric Display](#)
 - * [SparkFun Qwiic Joystick](#)
 - * [SparkFun Qwiic KeyPad](#)
 - * [SparkFun Qwiic OLED Display](#)

See the `examples/PlatformSpecific` folder for older [I2C](#) examples for each platform:

- [I2C](#) for MCF5441x-based Modules such as MOD5441x, NANO54415, SB800EX
- [I2C](#) for SAME70-based Modules such as MODM7AE70 and SBE70LC

12.19.1 PAC193X

`<examples/I2C/Wire/PAC193X>`

12.19.1.1 PAC193x Energy Monitor

12.19.1.1.1 Overview

This application demonstrates the use of the PAC193x Energy Monitor IC used onboard the SOMRT1061 development platform. The PAC193x is a multi-channel power/energy monitor that provides accurate measurements of voltage, current, power, and energy consumption.

12.19.1.1.2 Hardware Requirements

- SOMRT1061 development board with onboard PAC193x IC
- [I2C](#) interface for communication
- 24.9 milliohm sense resistor (default configuration)

12.19.1.1.3 Features

The application provides comprehensive power monitoring capabilities including:

- **Voltage Monitoring:** Bus voltage measurements in millivolts
- **Current Sensing:** Current measurements in milliamps through sense resistor
- **Power Calculation:** Instantaneous and average power in milliwatts
- **Energy Tracking:** Energy consumption in milliwatt-hours
- **Device Identification:** Product ID, Manufacturer ID, and Revision ID readout

12.19.1.1.4 Application Flow

1. Initialization

- Initialize I2C communication
- Configure PAC193x with default settings
- Read and display device identification information

2. Continuous Monitoring

- Updates all measurement values every 4 seconds
- Displays comprehensive power statistics
- Shows system tick counter for timing reference

12.19.1.1.5 Configuration Details

12.19.1.1.5.1 Default Settings

- **I2C Address:** 0x10
- **Sense Resistor:** 24.9 milliohms
- **Sample Rate:** Configurable (8, 64, 256, or 1024 samples/second)
- **Channels:** Channel 1 active, channels 2-4 disabled
- **Update Interval:** 4 seconds

12.19.1.1.5.2 Register Configuration

- NEG_PWR register: 0x00 (no negative power detection)
- CTRL register: 0x02 (control settings)
- CHANNEL_DIS register: 0x70 (disable channels 2-4)
- SLOW register: 0x20 (timing configuration)

12.19.1.1.6 Measurement Output

The application displays the following measurements:

```
Product      ID: 0x##
Manufacturer ID: 0x##
Revision     ID: 0x##
```

```
Read start:
Voltage      (mV) = ###.#####
Vsense       (mV) = #.#####
Current      (mA) = #.#####
Raw Power    (HEX) = 0x#####
Power        (mW) = #.#####
Avg Power    (mW) = #.#####
Raw Power Acc = 0x#####
Acc Count    = 0x#####
Energy       (mWh) = #.#####
```

12.19.1.1.7 Key Functions

12.19.1.1.7.1 Setup Functions

- `SetupPwrMon()`: Initializes I2C and PAC193x, reads device IDs
- `pwrMon.begin()`: Configures PAC193x registers and performs initial refresh

12.19.1.1.7.2 Measurement Functions

- `UpdateVoltage()`: Reads bus voltage
- `UpdateVsense()`: Reads sense voltage across resistor
- `UpdateCurrent()`: Calculates current from sense voltage
- `UpdatePower()`: Calculates instantaneous power
- `UpdateAvgPower()`: Calculates average power from accumulator
- `UpdateEnergy()`: Calculates energy consumption

12.19.1.1.8 Error Handling

The library includes comprehensive error handling with the following error codes:

- 0: No error
- -1: I2C read communication error
- -2: I2C write communication error
- -3: Invalid sense resistor value (zero or negative)
- -4: Invalid sample rate selection

12.19.1.1.9 Technical Specifications

12.19.1.1.9.1 Voltage Measurement

- **Range:** 0-32V
- **Resolution:** 16-bit
- **LSB:** $32000\text{mV} / 65536 = \sim 0.488\text{mV}$

12.19.1.1.9.2 Current Measurement

- **Method:** Sense voltage across precision resistor
- **Sense Voltage Range:** +/-100mV
- **Resolution:** 16-bit
- **Calculation:** $\text{Current} = (\text{Vsense} / \text{Rsense}) * 1,000,000 \text{ uA}$

12.19.1.1.9.3 Power Measurement

- **Method:** Digital multiplication of voltage and current
- **Resolution:** 28-bit
- **Accumulation:** 48-bit accumulator for average calculations

12.19.1.1.10 Library Architecture

The PAC193x library provides both raw register access and calculated real-world values:

- **Raw Functions:** Direct register values for maximum precision
- **Calculated Functions:** Converted values in standard engineering units
- **Flexible Configuration:** Multiple constructor options for different setups
- **I2C Abstraction:** Compatible with standard Wire library interface

12.19.1.1.11 Usage Example

```
#include "PAC193x.h"

PAC193x pwrMon(Wire, 100000); // Use Wire interface with 100k I2C speed

void setup() {
    Wire.begin();
    pwrMon.begin();

    // Read device information
    pwrMon.UpdateProductID();
    pwrMon.UpdateManufacturerID();
    pwrMon.UpdateRevisionID();
}

void loop() {
    // Update all measurements
    pwrMon.UpdateVoltage();
    pwrMon.UpdateCurrent();
    pwrMon.UpdatePower();
    pwrMon.UpdateEnergy();

    // Access measured values
    float voltage = pwrMon.Voltage; // mV
    float current = pwrMon.Current; // mA
    double power = pwrMon.Power; // mW
    float energy = pwrMon.Energy; // mWh

    delay(1000);
}
```

12.19.1.1.12 File Structure

- `main.cpp`: Main application demonstrating PAC193x usage
- `PAC193x.h`: Library header with class definition and register addresses
- `PAC193x.cpp`: Library implementation with all measurement functions

12.19.2 PicKit Serial I2C Demo Board

12.19.2.1 PicKit I2C Serial Demo Board Application

12.19.2.1.1 Overview

This application demonstrates interfacing with the Microchip PicKit Serial I2C Demo Board using NetBurner's I2C Wire library. The demo showcases communication with 5 different Microchip I2C peripherals connected to the board.

12.19.2.1.2 Supported I2C Peripherals

The application interfaces with the following I2C devices:

1. **24LC0B EEPROM** - 2Kbit (256 x 8) non-volatile memory
2. **MCP9800 Temperature Sensor** - 12-bit resolution temperature measurement
3. **MCP23008 GPIO Port Expander** - 8-bit I/O expander connected to LEDs
4. **TC1321 DAC** - 10-bit Digital-to-Analog Converter (0-2.5V range)
5. **MCP3221 ADC** - 12-bit Analog-to-Digital Converter

12.19.2.1.3 I2C Addressing

The application uses 7-bit I2C addressing as follows:

Device	Datasheet Address	Application Address
24LC0B EEPROM	0xA0	0x50
MCP9800 Temperature	0x92	0x49
MCP23008 GPIO	0x40	0x20

Device	Datasheet Address	Application Address
TC1321 DAC	0x90	0x48
MCP3221 ADC	0x9A	0x4D

Note: The application right-shifts datasheet addresses by 1 to remove the R/W bit for NetBurner [I2C](#) compatibility.*

12.19.2.1.4 Features

12.19.2.1.4.1 Interactive Menu System

The application provides a serial console menu with the following options:

- **1** - Erase EEPROM (fills with 0xFF)
- **2** - Read EEPROM (displays all 256 bytes)
- **3** - Write incrementing sequence to EEPROM (0x00-0xFF)
- **A** - Read ADC value and display as voltage
- **D** - DAC Test (prompts for 10-bit value input)
- **T** - Read Temperature Sensor (Celsius and Fahrenheit)

12.19.2.1.4.2 Automatic LED Sequencing

The application continuously cycles through the 8 LEDs connected to the MCP23008 GPIO expander, creating a moving light pattern that updates every 0.5 seconds.

12.19.2.1.5 Technical Details

12.19.2.1.5.1 EEPROM Operations

- Supports page writes up to 8 bytes at a time
- Implements write completion polling
- Handles write cycles automatically for larger data blocks
- 256-byte total capacity

12.19.2.1.5.2 Temperature Sensor

- Configured for 12-bit resolution
- Reads temperature in 2's complement format
- Converts to both Celsius and Fahrenheit
- Displays raw register data for debugging

12.19.2.1.5.3 DAC Operations

- 10-bit resolution (0-1023 range)
- 2.5V reference voltage
- Interactive value input with range validation
- Voltage output calculation and display

12.19.2.1.5.4 ADC Operations

- 12-bit resolution
- 3.3V reference voltage
- Automatic voltage conversion and display
- Raw register data output

12.19.2.1.6 Hardware Requirements

- NetBurner development board with I2C capability
- Microchip PicKit Serial I2C Demo Board
- Serial console connection for menu interaction

12.19.2.1.7 Important Notes

12.19.2.1.7.1 RTC Conflict Warning

The application includes a warning about I2C address conflicts with real-time clock (RTC) devices commonly found on NetBurner development boards. The RTC at address 0x51/0xA2 conflicts with the PicKit EEPROM address range and will not be functional when the PicKit board is connected.

12.19.2.1.7.2 Error Handling

The application includes comprehensive I2C error handling with descriptive status codes:

- `wire_success` - Operation completed successfully
- `wire_data_too_big` - Data size exceeds buffer limits
- `wire_NACK_on_address` - Device did not acknowledge address
- `wire_NACK_on_data` - Device did not acknowledge data
- `wire_other_error` - General I2C error
- `wire_timeout` - Communication timeout

12.19.2.1.8 Building and Running

1. Include the necessary NetBurner libraries and headers
2. Compile with the NetBurner toolchain
3. Flash to NetBurner device
4. Connect serial console at appropriate baud rate
5. Connect PicKit Serial I2C Demo Board to I2C bus
6. Use menu options to interact with peripherals

12.19.2.1.9 Code Structure

The application is organized into functional modules:

- **Main Menu System** - User interface and command processing
- **Device-Specific Functions** - Individual handlers for each I2C peripheral
- **Utility Functions** - ASCII conversion, buffer display, error handling
- **LED Animation** - Background LED sequencing task

12.19.3 RV3032 I2C Real-Time Clock

<examples/I2C/Wire/RTC-RV3032>

Supported Platforms: SOMRT1061 with DEV-SOMRT1061

External Real-Time Clock example using the Micro Crystal RV3032 RTC connected to the I2C bus, such as with the SOMRT1061-200IR on the DEV-SOMRT1061 development board containing the RV3032.

The example uses the Wire interface class for I2C operation, and requires an Internet connection to obtain the current time with NTP.

- Creates a NtpClientServlet instance to run as a background task in the configuration server.
- Reads the time stored in the RTC
- Compares the two values and sets the RTC to the NTP time if there is a difference of 2 minutes or more.

12.19.3.1 There are multiple I2C peripherals on each NetBurner platform as indicated in the tables below. The first

line is the default.

12.19.3.1.1 SOMRT1061

SCL	SOMRT PIN	SDA	SOMRT PIN	Pin Func	Module
Pin_EMC_12	9	Pin_EMC_11	10	PIN_FN↔ _2	4
Pin_SD_B0_00	27	Pin_SD_B0_01	26	PIN_FN↔ _2	3
Pin_AD_B1_00	40	Pin_AD_B1_01	41	PIN_FN↔ _3	1
Pin_AD_B1_07	47	Pin_AD_B1_06	46	PIN_FN↔ _1	3
Pin_EMC_22	83	Pin_EMC_21	84	PIN_FN↔ _2	3

12.19.3.1.2 MODM7AE70

Module	SCL	SDA
I2C0	P2.42	P2.39
I2C1	P2.12	P2.22
I2C2	P2.23	P2.26

12.19.3.1.3 SBE70LC

Module	SCL	SDA
I2C0	17	16

12.19.3.1.4 MOD5441X

Module	SCL	SDA
I2C0	J2.42	J2.39
I2C1	J2.41	J2.44

12.19.3.1.5 NANO54415

Module	SCL	SDA
I2C0	27	29

Module	SCL	SDA
I2C1	22	20
I2C4	26	24
I2C5	34	32

12.19.3.1.6 SB800EX

Module	SCL	SDA
I2C4	5	4

12.19.4 Wire Address Scan

12.19.4.1 I2C Wire Scan Application

12.19.4.1.1 Overview

This application scans the [I2C](#) bus for connected devices and displays their addresses using the Wire interface. It's a diagnostic utility that helps verify hardware connections and understand [I2C](#) device addressing.

12.19.4.1.2 Features

- Scans all valid [I2C](#) addresses (0x01 to 0x7F)
- Reports found device addresses in hexadecimal format
- Includes web server functionality for remote monitoring
- Provides network diagnostics capabilities

12.19.4.1.3 How It Works

The application performs the following operations:

1. **System Initialization:** Initializes the network stack and starts a web server on port 80
2. **Network Setup:** Waits for an active network connection
3. **I2C Scanning:** Sends ping requests to each address in the [I2C](#) address space
4. **Device Detection:** When a device responds with an acknowledgment (ACK), its address is recorded
5. **Results Display:** Outputs all found device addresses to the console

12.19.4.1.4 I2C Address Scanning Process

The scan covers addresses 1 through 127 (0x01 to 0x7F) in the 7-bit [I2C](#) address space. For each address, the application:

- Sends a request using `Wire.ping(address)`
- Waits for an acknowledgment from the device
- Records the address if a response is received

12.19.4.1.5 Output Format

For example, the application outputs scan results in the following format:

```
Doing scan
I2C Devices[01,48,68]
Scan done found 3 devices
```

Where:

- Device addresses are displayed in hexadecimal format
- Multiple addresses are comma-separated
- Total count of found devices is reported

12.19.4.1.6 Use Cases

This utility is particularly helpful for:

- **Hardware Verification:** Confirming that I2C devices are properly connected and powered
- **Address Discovery:** Identifying the actual addresses of connected I2C devices
- **Troubleshooting:** Diagnosing I2C communication issues
- **Documentation:** Understanding device addressing schemes in datasheets

12.19.4.1.7 Technical Notes

12.19.4.1.7.1 Address Format

Some I2C device datasheets include the read/write bit as part of the address specification, while others do not. This scanner reports the base 7-bit address without the read/write bit.

12.19.4.1.7.2 Address Range

- **Valid Range:** 0x01 to 0x7F (1 to 127 decimal)
- **Reserved Addresses:** 0x00 and addresses above 0x7F are not scanned as they are reserved or invalid

12.19.4.1.7.3 Network Features

The application includes network capabilities:

- Web server running on port 80
- System diagnostics enabled
- Network connection monitoring

12.19.4.1.8 Dependencies

- NNDK (NetBurner Network Development Kit)
- Wire library for I2C communication
- System libraries for network and timing functions

12.19.4.1.9 Building and Running

1. Compile the application using the NNDK toolchain
2. Flash to your NetBurner device
3. Connect to the device console to view scan results
4. Optionally access the web interface for remote monitoring

12.19.4.1.10 Example Output

```
Web Application: I2C Wire Scan
NNDK Revision: [version]
Doing scan
I2C Devices[20,3C,50]
Scan done found 3 devices
```

This example shows three I2C devices found at addresses 0x20, 0x3C, and 0x50.

12.19.4.1.11 Troubleshooting

If no devices are found:

- Check I2C wiring (SDA, SCL, power, ground)
- Verify pull-up resistors on SDA and SCL lines
- Confirm device power supply voltages
- Check for proper ground connections
- Verify device addresses in datasheets

12.19.4.1.12 File Structure

- `main.cpp`: Main application source code
- Web interface files (auto-generated by NNDK)

12.19.5 SparkFun Qwiic Alphanumeric Display

12.19.5.1 SparkFun Qwiic Alphanumeric Display Application

12.19.5.1.1 Overview

This application demonstrates the basic usage of the SparkFun Qwiic Alphanumeric Display (COM-16919) with NetBurner's RTOS platform. The program initializes the display, shows a welcome message, and then continuously displays a running seconds counter.

12.19.5.1.2 Hardware Requirements

- NetBurner development board with I2C/Wire support
- SparkFun Qwiic Alphanumeric Display (COM-16919)
- Qwiic connector cable or appropriate I2C wiring

12.19.5.1.3 Features

- Network stack initialization with web server capabilities
- I2C communication with the alphanumeric display
- Real-time display of system uptime in seconds
- Connection status verification
- System diagnostics enabled for development

12.19.5.1.4 Application Flow

1. Initialization Phase

- Initialize the network stack
- Enable system diagnostics
- Wait for active network connection (up to 25 seconds)
- Start HTTP web server on default port 80
- Initialize I2C Wire interface

2. Display Setup

- Initialize the HT16K33 display controller
- Verify display connection

- Display "NBRN" as initial message
- Wait 1 second before starting main loop

3. Main Loop

- Continuously monitor system seconds counter
- Update display with current seconds (modulo 10000)
- Print status dots to console every second

12.19.5.1.5 Code Structure

12.19.5.1.5.1 Key Components

- `SerialIf` `Serial` - Serial interface for debug output
- `TwoWire` `Wire` - `I2C` communication interface
- `HT16K33` `disp` - SparkFun alphanumeric display controller
- `UserMain()` - Main application entry point

12.19.5.1.5.2 Display Operations

The application uses the SparkFun Alphanumeric Display library which provides:

- `begin()` - Initialize display
- `isConnected()` - Check connection status
- `print()` - Display text/numbers

12.19.5.1.5.3 Network Features

The application includes network capabilities:

- Web server on port 80
- Network diagnostics
- Connection monitoring

12.19.5.1.6 Usage

1. Connect the SparkFun Qwiic Alphanumeric Display to your NetBurner board via `I2C`
2. Compile and upload the application
3. Monitor the serial console for:
 - Application name and NNDK revision
 - Display initialization status
 - Connection verification
 - Running status dots
4. Observe the display showing:
 - Initial "NBRN" message
 - Continuous seconds counter (0-9999, then repeats)

12.19.5.1.7 Console Output

The application provides debug information via serial console:

```
Web Application: SparkFunAlphaDisplay
NNDK Revision: [version]
Begin True/false
Connected True/false
. . . . (status dots every second)
```

12.19.5.1.8 Development Notes

- System diagnostics are enabled for development purposes
- Consider removing `EnableSystemDiagnostics()` for production builds
- The web server functionality is initialized but not actively used in this example
- Display shows seconds modulo 10000 to fit the 4-digit alphanumeric format

12.19.5.1.9 Dependencies

- NetBurner RTOS libraries
- SparkFun Alphanumeric Display library
- Standard I2C/Wire libraries

12.19.5.1.10 Troubleshooting

1. **Display not initializing:** Check [I2C](#) connections and power
2. **Connection false:** Verify proper [I2C](#) address and wiring
3. **Network timeout:** Ensure network configuration is correct
4. **No display updates:** Check that the main loop is running and seconds counter is incrementing

12.19.5.1.11 Related Documentation

This example is part of the NetBurner [I2C](#) examples collection: `examples/I2C/Wire/SparkFunQwiic/↔AlphaDisplay`

For more information about the SparkFun Qwiic Alphanumeric Display, refer to the SparkFun documentation for COM-16919.

12.19.6 SparkFun Qwiic Joystick

12.19.6.1 SparkFun Qwiic Alphanumeric Display Application

12.19.6.1.1 Overview

This application demonstrates integration between NetBurner's RTOS platform and the SparkFun Qwiic Alphanumeric Display (COM-16919). The program initializes both network services and the [I2C](#) display, then continuously shows a running seconds counter on the 4-digit alphanumeric display.

12.19.6.1.2 Hardware Requirements

- NetBurner development board with I2C/Wire support
- SparkFun Qwiic Alphanumeric Display (COM-16919)
- Qwiic connector cable or appropriate [I2C](#) wiring connections

12.19.6.1.3 Key Features

- **Network Integration:** Full network stack initialization with HTTP web server
- **I2C Communication:** Direct interface with HT16K33 display controller
- **Real-time Display:** Live system uptime counter (0-9999 seconds, cycling)
- **Connection Verification:** Automatic display connectivity checking
- **Development Diagnostics:** Built-in system diagnostics for debugging

12.19.6.1.4 Application Architecture

12.19.6.1.4.1 Initialization Sequence

1. Network Setup

- Initialize NetBurner network stack
- Enable system diagnostics for development
- Wait up to 25 seconds for active network connection
- Start HTTP web server on default port 80

2. Display Configuration

- Initialize I2C Wire interface
- Configure HT16K33 display controller
- Verify display connection status
- Display initial "NBRN" message
- Pause 1 second before entering main loop

3. Main Operation Loop

- Monitor system seconds counter continuously
- Update display with current seconds (modulo 10000)
- Output status indicators to console

12.19.6.1.4.2 Core Components

```
SerialIf Serial;    // Debug output interface
TwoWire Wire;       // I2C communication bus
HT16K33 disp;       // SparkFun display controller
```

12.19.6.1.4.3 Display Operations

The application utilizes the SparkFun Alphanumeric Display library:

- `begin()` - Initialize display hardware
- `isConnected()` - Verify I2C connection
- `print()` - Display text and numeric values

12.19.6.1.5 Usage Instructions

12.19.6.1.5.1 Hardware Setup

1. Connect the SparkFun Qwiic Alphanumeric Display to your NetBurner board
2. Use either Qwiic connector cable or manual I2C wiring (SDA/SCL)
3. Ensure proper power connections

12.19.6.1.5.2 Software Deployment

1. Compile the application using NetBurner development tools
2. Upload the compiled firmware to your NetBurner board
3. Monitor serial console for initialization status

12.19.6.1.5.3 Expected Behavior

Serial Console Output:**

```
Web Application: SparkFunAlphaDisplay
NNDK Revision: [version number]
Begin True/false
Connected True/false
. . . . (status dots every second)
```

Display Sequence:**

1. Initial display shows "NBRN" for 1 second
2. Continuous seconds counter (0000-9999, then cycles back to 0000)
3. Updates every second with current system uptime

12.19.6.1.6 Code Structure

12.19.6.1.6.1 Main Function Flow

```
void UserMain(void *pd)
{
    // Network and system initialization
    init();
    EnableSystemDiagnostics();
    WaitForActiveNetwork(TICKS_PER_SECOND * 25);
    StartHttp();

    // Display setup and verification
    Wire.begin();
    disp.begin();
    disp.isConnected(0);

    // Main operational loop
    while (1) {
        // Monitor and display seconds counter
    }
}
```

12.19.6.1.6.2 Timing Considerations

- Network timeout: 25 seconds maximum wait
- Display update rate: Every 1 second
- Initial display pause: 1 second
- Console status output: Every second

12.19.6.1.7 Development Notes

12.19.6.1.7.1 Production Considerations

- Remove `EnableSystemDiagnostics()` for production builds
- Consider adding error handling for network failures

12.19.6.1.7.2 Display Limitations

- 4-digit alphanumeric format limits counter to 0-9999
- Counter automatically cycles back to 0000 after reaching 9999
- Display uses modulo arithmetic: `Secs % 10000`

12.19.6.1.7.3 Network Features

- HTTP web server runs on port 80
- Network diagnostics enabled for development
- Connection monitoring with timeout protection

12.19.6.1.8 Troubleshooting

12.19.6.1.8.1 Common Issues

Display Not Initializing**

- Check I2C connections (SDA/SCL pins)
- Verify power supply to display
- Confirm I2C address conflicts
Connection Status False**
- Verify proper I2C wiring
- Check display power connections
- Ensure correct I2C address (default for HT16K33)
Network Timeout**
- Verify network configuration
- Check Ethernet cable connections
- Confirm network settings in NetBurner configuration
No Display Updates**
- Verify main loop execution
- Check that seconds counter is incrementing
- Confirm I2C communication is functioning

12.19.6.1.8.2 Debug Information

The application provides comprehensive debug output:

- Application name and NNDK revision
- Display initialization status
- Connection verification results
- Real-time status indicators (dots every second)

12.19.6.1.9 Dependencies

12.19.6.1.9.1 Required Libraries

- NetBurner RTOS core libraries
- SparkFun Alphanumeric Display library
- Standard I2C/Wire communication libraries
- Network stack libraries

12.19.6.1.10 Technical Specifications

12.19.6.1.10.1 Display Controller

- **Chip:** HT16K33 LED controller
- **Interface:** I2C communication
- **Display Type:** 4-digit alphanumeric
- **Update Rate:** 1 Hz (every second)

12.19.6.1.10.2 Network Services

- **Web Server:** HTTP on port 80
- **Connection Timeout:** 25 seconds
- **Diagnostics:** Development mode enabled

12.19.6.1.10.3 System Integration

- **Platform:** NetBurner RTOS
- **Communication:** [I2C](#) (Wire library)
- **Timing:** RTOS tick-based scheduling

12.19.6.1.11 Related Documentation

- SparkFun Qwiic Alphanumeric Display (COM-16919) datasheet
- NetBurner RTOS documentation
- HT16K33 LED controller specifications
- [I2C](#) communication protocols

12.19.7 SparkFun Qwiic KeyPad

12.19.7.1 SparkFun Qwiic KeyPad Application

12.19.7.1.1 Overview

This application demonstrates the basic use of the SparkFun Qwiic KeyPad (COM-15290) with an embedded system using the NetBurner NNDK (Network Development Kit). The application provides a simple interface to read button presses from the keypad and display them via serial output.

12.19.7.1.2 Hardware Requirements

- NetBurner development board with [I2C](#) support
- SparkFun Qwiic KeyPad (COM-15290)
- Qwiic connection cable

12.19.7.1.3 Features

- **[I2C](#) Communication:** Uses the Wire library for [I2C](#) communication with the keypad
- **Web Server:** Includes a basic HTTP web server on port 80
- **Network Stack:** Full network stack initialization with diagnostics
- **Real-time Button Detection:** Continuously monitors for button presses
- **Serial Output:** Displays button presses and status information via serial interface
- **Connection Monitoring:** Verifies keypad connectivity on startup

12.19.7.1.4 Application Behavior

12.19.7.1.4.1 Initialization Sequence

1. Initializes the network stack
2. Enables system diagnostics
3. Starts HTTP web server on port 80
4. Waits for active network connection (5 second timeout)
5. Initializes [I2C](#) communication
6. Attempts to connect to the SparkFun Qwiic KeyPad
7. Verifies connection status

12.19.7.1.4.2 Runtime Operation

- Continuously polls the keypad for button presses
- Updates the internal FIFO buffer to handle multiple key presses
- Displays pressed keys in square brackets via serial output: `[key]`
- Prints a dot (.) every second as a heartbeat indicator
- Runs indefinitely until system reset

12.19.7.1.4.3 Serial Output Examples

```
Web Application: SparkFunKeypad
NNDK Revision: [version]
Begin True
Connected True
[1][2][3]....[A][B]....
```

12.19.7.1.5 Code Structure

12.19.7.1.5.1 Key Components

- **Serial Interface:** `SerialIf Serial` - handles serial communication
- **Keypad Object:** `KEYPAD key` - SparkFun keypad library instance
- **I2C Interface:** `TwoWire Wire` - [I2C](#) communication handler

12.19.7.1.5.2 Main Functions

- `init()` - Network stack initialization
- `key.begin()` - Keypad initialization
- `key.isConnected()` - Connection verification
- `key.getButton()` - Button press detection
- `key.updateFIFO()` - Internal buffer management

12.19.7.1.6 Dependencies

- NetBurner NNDK framework
- SparkFun Qwiic Keypad Arduino Library
- Wire library for [I2C](#) communication
- Standard NetBurner system libraries ([init.h](#), [nbtos.h](#), [system.h](#))

12.19.7.1.7 Setup Instructions

1. Connect the SparkFun Qwiic KeyPad to your NetBurner board via I2C/Qwiic interface
2. Compile and flash the application to your NetBurner device
3. Open a serial terminal to monitor output
4. The application will automatically initialize and begin monitoring for key presses
5. Press keys on the keypad to see output in the serial terminal

12.19.7.1.8 Troubleshooting

12.19.7.1.8.1 Common Issues

- ****"Begin false"**: Keypad initialization failed** - check I2C connections
- ****"Connected false"**: Keypad not detected** - verify power and I2C address
- **No key output**: Check FIFO buffer and polling frequency

12.19.7.1.8.2 Debug Tips

- Verify I2C connections and pull-up resistors
- Check keypad I2C address (default is typically 0x4B)
- Ensure proper power supply to the keypad
- Monitor serial output for initialization status messages

12.19.7.1.9 Technical Notes

- The application uses a 1ms delay in the main loop for efficient polling
- Button presses are buffered internally by the keypad library
- The web server functionality provides additional networking capabilities
- System diagnostics can be disabled for production builds by removing `EnableSystemDiagnostics()`

12.19.8 SparkFun Qwiic OLED Display

12.19.8.1 SparkFun Qwiic OLED Display - Analog Clock

This application demonstrates the use of a SparkFun Qwiic OLED Display to create an analog clock interface using the NetBurner platform.

12.19.8.1.1 Overview

The SparkFun OLED Display example creates a real-time analog clock that displays the current time with traditional clock hands (hour, minute, and second hands). The clock face includes numeric markers at the 12, 3, 6, and 9 o'clock positions.

12.19.8.1.2 Supported Hardware

This application supports multiple SparkFun Qwiic OLED display models:

- QwiicMicroOLED (default)
- QwiicTransparentOLED
- QwiicNarrowOLED
- Qwiic1in3OLED

Note: Only one display type should be uncommented in the source code at a time.

12.19.8.1.3 Features

- **Real-time analog clock display** with hour, minute, and second hands
- **Configurable clock speed** - adjustable from real-time to accelerated modes
- **Automatic time progression** - seconds, minutes, and hours increment automatically
- **Circular clock face** with numeric position markers
- **Network connectivity** with built-in web server support
- **System diagnostics** integration

12.19.8.1.4 Configuration

12.19.8.1.4.1 Initial Time Settings

The clock can be initialized with custom time values:

```
int hours = 11;      // Initial hour (0-11, 12-hour format)
int minutes = 50;    // Initial minutes (0-59)
int seconds = 30;    // Initial seconds (0-59)
```

12.19.8.1.4.2 Clock Speed

The clock update rate can be adjusted using the `CLOCK_SPEED` constant:

- Set to 1 for rapid/demo mode
- Set to 1000 for approximately real-time operation (1 second intervals)

12.19.8.1.5 Hardware Requirements

- NetBurner development board with I2C/Wire support
- SparkFun Qwiic OLED Display (one of the supported models)
- Qwiic connection cable or appropriate I2C wiring

12.19.8.1.6 Key Components

12.19.8.1.6.1 Libraries Used

- [SparkFun_Qwiic_OLED.h](#) - SparkFun OLED display library
- [Wire.h](#) - I2C communication library
- [Arduino.h](#) - Arduino compatibility layer for NetBurner
- NetBurner RTOS libraries ([nbrtos.h](#), [system.h](#), [init.h](#))

12.19.8.1.6.2 Main Functions

- `initClockVariables()` - Calculates display positioning and sizing constants
- `updateTime()` - Handles time progression (seconds -> minutes -> hours)
- `drawArms()` - Renders the clock hands using trigonometric calculations
- `drawFace()` - Draws the clock border and numeric markers
- `UserMain()` - Main application loop with network initialization

12.19.8.1.7 Display Details

The clock automatically adapts to the display dimensions:

- **Clock radius** is calculated based on available screen space
- **Hand lengths** are proportional to the clock radius:
 - Second hand: 100% of available radius
 - Minute hand: 70% of available radius
 - Hour hand: 50% of available radius
- **Numeric markers** are positioned at cardinal directions (12, 3, 6, 9)

12.19.8.1.8 Network Features

The application includes full network stack initialization:

- Web server on port 80
- 5-second network connection timeout
- System diagnostics enabled
- Compatible with NetBurner's web-based configuration

12.19.8.1.9 Usage

1. Connect the SparkFun Qwiic OLED display to the NetBurner board via [I2C](#)
2. Uncomment the appropriate display type in the source code
3. Adjust initial time and clock speed settings if desired
4. Compile and upload the application
5. The analog clock will begin displaying immediately upon successful initialization

12.19.8.1.10 Error Handling

The application includes basic error handling:

- Display initialization failure detection
- Infinite loop protection if hardware initialization fails
- Serial output for debugging and status reporting

12.19.8.1.11 File Structure

- `main.cpp` - Main application source code
- `ReadMe.txt` - Basic project documentation
- This README provides comprehensive application documentation

12.19.8.1.12 Technical Notes

- The clock uses trigonometric functions to calculate hand positions
- Display updates occur in a non-blocking loop structure
- The application maintains compatibility with NetBurner's RTOS environment
- [I2C](#) communication is handled through the standard Wire library interface

12.20 IP Address Object (IPADDR)

12.20.1 FunWithIPADDR - NetBurner IP Address Object Example

12.20.1.1 Overview

`FunWithIPADDR` is a C++ demonstration application that showcases the NetBurner IPADDR object functionality. This application demonstrates how to work with IP addresses in a dual-stack (IPv4/IPv6) environment using NetBurner's unified IPADDR object system.

Note

In rare cases in which you want to create an object of a specific type, you can create objects as `IPADDR4` or `IPADDR6`. However, we highly recommend using `IPADDR` for all your IP addresses. Note that if you are using a `printf()` style format specifier, you must use `"%HI"` instead of `"%I"` and pass in a `IPADDR4` object, otherwise the value displayed will be 0.0.0.0.

12.20.1.2 What is IPADDR?

The `IPADDR` object is a C++ class that provides a unified interface for handling both IPv4 and IPv6 addresses without requiring separate code paths. This design enables applications to support both IP versions seamlessly, with the object automatically determining the appropriate operations based on the assigned address type.

12.20.1.2.1 Key Features

- **Dual-stack support:** Single object handles both IPv4 and IPv6 addresses
- **Automatic type detection:** Member functions automatically call appropriate IPv4 or IPv6 implementations
- **Unified printf formatting:** Custom `I` format specifier for easy display
- **Null address handling:** Built-in methods to check for and set null addresses
- **Type-specific objects:** Optional `IPADDR4` and `IPADDR6` for specialized use cases

12.20.1.3 Application Functionality

The application demonstrates the following IPADDR capabilities:

12.20.1.3.1 1. Basic Address Assignment and Display

- Creates `IPADDR` objects for both IPv4 and IPv6 addresses
- Uses `SetFromAscii()` to assign addresses from string literals
- Displays addresses using the custom `I` printf format specifier

12.20.1.3.2 2. Member Function Usage

- Demonstrates the `print()` member function for address output
- Shows how the same function works for both IPv4 and IPv6 addresses

12.20.1.3.3 3. Null Address Handling

- Tests for null addresses using `IsNull()` member function
- Demonstrates setting addresses to null with `SetNull()`

12.20.1.3.4 4. Type-Specific Objects (IPADDR4)

- Shows creation of `IPADDR4` objects for IPv4-specific needs
- Demonstrates the required `HI` format specifier for `IPADDR4` objects

12.20.1.3.5 5. IPv6 Static Methods (Conditional)

- When IPv6 is enabled, shows usage of `AsciiToIp6()` static method

12.20.1.4 Code Examples

12.20.1.4.1 Creating and Using IPADDR Objects

```
// IPv4 address
IPADDR ipaddrIPv4;
ipaddrIPv4.SetFromAscii("192.168.1.4");
iprintf("IPv4 Address: %I\r\n", ipaddrIPv4);

// IPv6 address
IPADDR ipaddrIPv6;
ipaddrIPv6.SetFromAscii("2001:0db8:85a3:0000:0000:8a2e:0370:0006");
iprintf("IPv6 Address: %I\r\n", ipaddrIPv6);
```

12.20.1.4.2 Null Address Checking

```
if (ipaddrIPv4.IsNull())
    iprintf("Address is null\r\n");
else
    iprintf("Address: %I\r\n", ipaddrIPv4);
```

12.20.1.4.3 Using Type-Specific Objects

```
IPADDR4 ipaddrIPADDR4;
ipaddrIPADDR4.SetFromAscii("192.168.4.4");
// Note: Must use %HI for IPADDR4 objects
iprintf("IPADDR4: %HI\r\n", ipaddrIPADDR4);
```

12.20.1.5 Important Notes

12.20.1.5.1 Printf Format Specifiers

- Use `I` for general IPADDR objects
- Use `HI` for `IPADDR4` objects (failure to do so will display 0.0.0.0)
- The `I` format works with all printf variants (printf, iprintf, sprintf, etc.)

12.20.1.5.2 Best Practices

- Use the generic IPADDR class for most applications
- Only use `IPADDR4` or `IPADDR6` when type-specific functionality is required
- Always check for null addresses when necessary
- The unified approach reduces code complexity and maintenance

12.20.1.6 Application Flow

1. **Initialization:** Sets up the NetBurner system and network stack
2. **Network Wait:** Waits up to 5 seconds for active network connectivity
3. **HTTP Start:** Initializes the HTTP server
4. **IPADDR Demonstrations:** Runs through various IPADDR usage examples
5. **Main Loop:** Enters an infinite loop with 1-second delays

12.20.1.7 Build Requirements

- NetBurner development environment
- NetBurner RTOS ([nbrtos.h](#))
- HTTP support ([http.h](#))
- Standard C++ compiler with NetBurner extensions

12.20.1.8 System Features Used

- System diagnostics (enabled for debugging)
- Network stack initialization
- HTTP server functionality
- Custom printf formatting

12.21 IPv6

IPv6 Examples:

- [DHCPv6](#)

12.21.1 DHCPv6

12.21.1.1 DHCPv6 Client Application

12.21.1.1.1 Overview

This application demonstrates the implementation of a DHCPv6 (Dynamic Host Configuration Protocol for IPv6) client using the NetBurner NNDK (NetBurner Network Development Kit). The application showcases various callback functions for requesting, adding, and processing DHCPv6 options in an IPv6 network environment.

12.21.1.1.2 Application Name

IPv6-DHCPv6**

12.21.1.1.3 Purpose

The DHCPv6 client application serves as a comprehensive example of how to:

- Implement DHCPv6 client functionality in embedded systems
- Configure and handle DHCPv6 callback functions
- Request specific DHCPv6 options from servers
- Process DHCPv6 reply messages
- Manage device naming for [DHCP](#) offers
- Display IPv6 address information

12.21.1.1.4 Key Features

12.21.1.1.4.1 DHCPv6 Client Callbacks

The application implements three main callback functions in the `NB::V6::DHCPv6` namespace:

1. ****Option Request Callback (AddORO)****
 - Returns additional options to receive from the DHCPv6 server
 - Configured to request: Preference, DNS Servers, and Domain List options
 - The returned list must be NULL-terminated
2. ****Request Option Callback (AddSendOption)****
 - Allows setting additional options for outgoing DHCPv6 messages
 - Can determine message type using `msg.GetType()`
 - Currently contains example code for FQDN options (commented out)

3. ****Reply Callback (GetReply)****

- Called when Reply messages are received and verified
- Processes DHCPv6 options that the system requires
- Provides debug output for received messages

12.21.1.1.4.2 Device Naming

- Uses the `DEVICE_NAME` parameter from configuration settings for DHCP offers
- Automatically generates device names if not configured (when `autoDeviceName` is enabled)
- Format: `NB_[PlatformName]_[Last3MacBytes]`
- Automatically saves configuration and reboots when creating new device names

12.21.1.1.4.3 Network Interface Management

- Waits for active network connection (10-second timeout)
- Retrieves and displays the first available network interface
- Shows device name used for DHCP offers

12.21.1.1.5 Required Dependencies

12.21.1.1.5.1 NetBurner Libraries

- `init.h` - Initialization functions
- `nbrtos.h` - NetBurner Real-Time OS
- `netinterface.h` - Network interface management
- `system.h` - System functions
- `config_server.h` - Configuration management
- `ipv6/dhcpv6_internal.h` - DHCPv6 internal definitions
- `ip_util.h` - IP utility functions

12.21.1.1.5.2 Compilation Requirements

- IPv6 support must be enabled (`IPV6` preprocessor definition)
- Application will display an error message if IPv6 is disabled

12.21.1.1.6 Usage

12.21.1.1.6.1 Startup Sequence

1. Initialize system and enable diagnostics
2. Wait for active network connection
3. Start HTTP server
4. Check and configure device name if necessary
5. Set up DHCPv6 callbacks
6. Display application information
7. Enter interactive mode

12.21.1.1.6.2 Interactive Mode

- Press Enter to display current IPv6 address information
- Application continuously monitors and displays network status
- Note: Routable addresses may take several seconds to receive

12.21.1.1.6.3 Debug Information

- Enable `DHCP_DEBUG` in `dhcpv6_internal.h` and recompile the library to see full DHCPv6 messages
- System diagnostics are enabled by default (should be disabled for production)

12.21.1.1.7 DHCPv6 Options Requested

The application is configured to request the following DHCPv6 options:

- `OPT_PREFERENCE` - Server preference values
- `OPT_DNS_SERVERS` - DNS server addresses
- `OPT_DOMAIN_LIST` - Domain search list

12.21.1.1.8 Configuration

12.21.1.1.8.1 Device Name Configuration

- Set `DEVICE_NAME` in the network interface configuration
- If not set and `autoDeviceName` is enabled, the application will:
 1. Generate a name based on platform and MAC address
 2. Save the configuration
 3. Automatically reboot the device

12.21.1.1.8.2 Network Interface

- Application uses the first available network interface
- Displays the interface's MAC address and device name
- Waits up to 10 seconds for network activation

12.21.1.1.9 Build Configuration

12.21.1.1.9.1 Preprocessor Definitions

- `IPV6` must be defined for the application to function
- Without IPv6 support, the application will display an error message

12.21.1.1.9.2 Production Considerations

- Remove `EnableSystemDiagnostics()` call for production builds
- Disable `DHCP_DEBUG` in production environments
- Consider disabling automatic device name generation (`autoDeviceName = false`)

12.21.1.1.10 Example Output

```
Application: IPv6-DHCPv6
NNDK Revision: [Version]
DHCP Offer Name: [DeviceName]
Type the enter key to display address information
Note that routable addresses may take a few seconds to receive

Reply Callback
[DHCPv6 debug information if enabled]
[IPv6 address information]
```

12.21.1.1.11 Error Handling

- Graceful handling of missing device names
- Automatic configuration and reboot when needed
- Network timeout protection (10-second wait)
- IPv6 availability checking

12.21.1.1.12 Files

- `main.cpp` - Main application implementation
- `ReadMe.txt` - Original documentation
- `ip_util.h` - IP utility functions (referenced)

12.21.1.1.13 Technical Notes

- The application uses the NetBurner RTOS task system
- Memory management follows NetBurner guidelines
- All DHCPv6 option arrays must be statically allocated (not stack-based)
- The library automatically prevents duplicate options from being sent

12.22 JSON

These series of examples provide code for a JSON server running on a NetBurner device, "demo.netburner.com", and JSON client examples for GET, POST, loading applications and HTML. If you type "demo.netburner.com" in your web browser and can see a JSON server, you can run the client examples against it.

- [JSON Lexer](#)

JSON Examples:

- [NetBurner Demo Server for JSON Client Examples](#)
- [Get JSON Object From Server](#)
- [Get Non-JSON Object From Server](#)
- [Get JSON Object From Server](#)
- [JSON Linter Test](#)
- [Post JSON to Server and Display Result](#)
- [Post JSON to Server](#)
- [Get Application Image and Load it From a Web Server](#)
- [Simple JSON HTML Example](#)
- [Simple JSON Post Receiver Example](#)

12.22.1 JSON Lexer

The examples here show how to use the [JSON Lexer](#) classes, [ParsedJsonDataSet](#) and [JsonRef](#), to build and interact with JSON data sets.

JSON Lexer Examples:

- [JSON Array](#)
- [JSON Array with Objects](#)
- [Parse Test](#)

12.22.1.1 JSON Array

12.22.1.1.1 JSON Array Application

12.22.1.1.1.1 Overview

This application demonstrates how to use the JSON Lexer library classes [ParsedJsonDataSet](#) and [JsonRef](#) to create and access JSON array information. The program builds a JSON array with numeric data and then reads back the stored values.

12.22.1.1.1.2 Application Details

Application Name:** JsonArray

The application showcases the fundamental operations of:

- Building JSON data structures programmatically
- Storing arrays within JSON datasets
- Accessing and iterating through JSON array elements
- Converting JSON data to native data types

12.22.1.1.1.3 Code Structure

Main Components

BuildJsonArray ()

This function demonstrates the JSON array creation process:

- Initializes the JSON data set building process
- Creates an array named "FirstArray"
- Populates the array with integers 0-9
- Finalizes the array and dataset construction

ReadJsonArray ()

This function shows how to access and iterate through JSON array data:

- Retrieves a reference to the "FirstArray"
- Iterates through array elements using index-based access
- Converts JSON values to native double format
- Prints each array element with its index

UserMain()

The main application entry point that:

- Initializes the system
- Enables system diagnostics
- Waits for network connectivity (5 seconds)
- Orchestrates the build and read operations
- Displays the complete JSON object structure
- Enters an infinite loop with 1-second delays

12.22.1.1.1.4 Expected Output

When executed, the application will:

1. Display "Application started" message
2. Show "Build JSON Object:" followed by the complete JSON structure
3. Display each array element in the format: "[index]: [value]"
4. Show "End of Array Reached" when iteration completes

Example output format:

```
Application started

Build JSON Object:
[JSON structure display]

  0: 0
  1: 1
  2: 2
  ...
  9: 9
End of Array Reached
```

12.22.1.1.1.5 Dependencies

The application requires:

- `init.h` - System initialization functions
- `json_lexer.h` - JSON parsing and manipulation library
- Network stack (uses `DHCP` and network timing functions)
- RTOS functionality (uses `OSTimeDly` and `TICKS_PER_SECOND`)

12.22.1.1.1.6 Key Features

- **Dynamic JSON Construction:** Build JSON arrays programmatically at runtime
- **Type Safety:** Automatic conversion between JSON and native data types
- **Iterator Support:** Easy traversal of array elements using index notation
- **Validation:** Built-in validity checking for array elements
- **Debug Output:** Complete JSON object structure printing capability

12.22.1.1.1.7 Usage Notes

- The application waits for network initialization before proceeding
- System diagnostics are enabled for development/debugging purposes
- The main loop runs indefinitely with 1-second intervals
- Array elements are accessed using standard bracket notation
- The `Valid()` method is used to detect the end of array iteration

12.22.1.1.1.8 Technical Implementation

The application leverages the JSON Lexer library's object-oriented approach to JSON manipulation, using:

- `ParsedJsonDataSet` for the main JSON container
- `JsonRef` for element references and access
- Method chaining for efficient array construction
- Implicit type conversion for data access

12.22.1.2 JSON Array with Objects

12.22.1.2.1 JSON Array with Objects Example

12.22.1.2.1.1 Overview

This application demonstrates how to use the JSON Lexer classes `ParsedJsonDataSet` and `JsonRef` to create and access arrays containing sub-objects. The program creates a JSON array with 10 objects, each containing two key-value pairs, and then reads back the data to display it.

12.22.1.2.1.2 Application Details

Application Name:** JsonArrayWithObjects

The program showcases the following JSON manipulation capabilities:

- Building JSON arrays programmatically
- Adding objects as array elements
- Accessing array elements by index
- Reading object properties within arrays

12.22.1.2.1.3 Code Structure

Main Functions

`BuildJsonArray()`

Creates a JSON array named "FirstArray" containing 10 objects. Each object has:

- Key "a" with value equal to the loop index (0-9)
- Key "b" with value equal to the loop index multiplied by 10 (0, 10, 20, ..., 90)

`ShowObject(JsonRef element)`

Displays the contents of a single object by printing its "a" and "b" values.

`ReadJsonArrayWithObj()`

Iterates through the "FirstArray" and displays each object's contents using the `ShowObject()` function.

UserMain(void *pd)

Main entry point that:

1. Initializes the system
2. Enables system diagnostics
3. Waits for network connectivity
4. Builds the JSON array
5. Prints the complete JSON object
6. Reads and displays the array contents
7. Enters an infinite loop

12.22.1.2.1.4 JSON Structure Created

The application creates a JSON structure similar to:

```
{
  "FirstArray": [
    { "a": 0, "b": 0 },
    { "a": 1, "b": 10 },
    { "a": 2, "b": 20 },
    ...
    { "a": 9, "b": 90 }
  ]
}
```

12.22.1.2.1.5 Dependencies

- [init.h](#) - System initialization
- [json_lexer.h](#) - JSON parsing and manipulation classes

12.22.1.2.1.6 Usage

The application automatically runs when started and will:

1. Display startup message
2. Build and print the JSON object
3. Read through the array and display each object's contents
4. Continue running indefinitely

12.22.1.2.1.7 Key Classes Used

- [ParsedJsonDataSet](#): Main class for building and storing JSON data
- [JsonRef](#): Reference class for accessing JSON elements and navigating the structure

12.22.1.3 Parse Test**12.22.1.3.1 JSON Parser Test Application****12.22.1.3.1.1 Overview**

This application demonstrates the usage of a JSON lexer and parser library, specifically showcasing how to use the [ParsedJsonDataSet](#) and [JsonRef](#) classes to create and access JSON information. The program is designed as a test application that parses a complex JSON structure and demonstrates various access methods and type checking capabilities.

12.22.1.3.1.2 Application Details

- **Application Name:** JsonParseTest
- **Main File:** main.cpp
- **Purpose:** Parse Test demonstration for JSON Lexer classes

12.22.1.3.1.3 Features

The application demonstrates the following JSON parsing capabilities:

1. JSON Data Structure

The program works with a complex nested JSON structure that includes:

- Boolean values (SUCCESS, b3, aftaav)
- Null values (ACTION, b4)
- Numeric values (integers and floats)
- String values
- Nested objects (JDATA, bar, b2, s4, wa)
- Arrays (av, a, z)
- Mixed-type arrays containing numbers, strings, objects, and nested arrays

2. Access Methods

The application demonstrates two primary methods for accessing JSON data:

Explicit Type Enforcement

```
int32_t j = pjd.object("wa").name("a")[5].name("z")[1];
const char *str = pjd.object("wa").name("a")[4];
NBString nbs = pjd.object("JDATA").object("bar").object("b2").object("s4").name("Sub4a");
```

Simplified Access (Without Type Enforcement)

```
j = pjd("wa")("a")[5]("z")[1];
str = pjd("wa")("a")[4];
nbs = (const char*)(pjd("JDATA")("bar")("b2")("s4")("Sub4a"));
```

3. Type Checking

The application includes comprehensive type checking capabilities:

- `IsNumber()` - Check if value is numeric
- `IsObject()` - Check if value is a JSON object
- `IsNull()` - Check if value is null
- `IsArray()` - Check if value is an array
- `IsBool()` - Check if value is boolean
- `IsString()` - Check if value is a string

4. Error Handling

The application demonstrates proper error handling for:

- Non-existent objects and properties
- Invalid parse attempts
- Validation using `JsonRef.Valid()` method

12.22.1.3.1.4 JSON Structure Example

The embedded JSON data structure includes:

```
{
  "SUCCESS": true,
  "ACTION": null,
  "JDATA": {
    "foo": 7,
    "bar": {
      "b1": "Sb1",
      "b2": {
        "s1": "Sub1",
        "s2": "Sub2",
        "s3": "Sub3",
        "s4": {
          "Sub4a": "Test4a",
          "Sub4b": "Test4b"
        },
        "s5": "Sub5",
        "s6": "Sub6"
      },
      "b3": true,
      "b4": null
    },
    "bogus": "Bstring",
    "nubog": 1234.57,
    "av": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    "aftaav": true
  },
  "wa": {
    "a": [1, 2, 3, 4, "bob",
      {"b": "c", "z": [1, 2, 3]},
      [12, 10, 9, "zoro"],
      93]
  }
}
```

12.22.1.3.1.5 Key Classes

ParsedJsonDataSet

- Main class for parsing and storing JSON data
- Supports initialization from character array data
- Provides `PrintObject()` method for debugging
- Supports both explicit and simplified access methods

JsonRef

- Reference class for accessing JSON elements
- Provides type checking methods
- Includes validation through `Valid()` method
- Supports automatic type conversion

12.22.1.3.1.6 Usage Pattern

1. **Initialize:** Create a `ParsedJsonDataSet` object with JSON data
2. **Access:** Use either explicit or simplified syntax to navigate the JSON structure
3. **Validate:** Check if the parse was successful using `Valid()`
4. **Type Check:** Verify data types before processing
5. **Extract:** Convert to appropriate C++ types (`int32_t`, `const char*`, `NBString`, etc.)

12.22.1.3.1.7 System Requirements

- Embedded system environment (references to [DHCP](#), network waiting)
- NetBurner platform (references to [NBString](#), system diagnostics)
- Real-time operating system (OSTimeDly, TICKS_PER_SECOND)

12.22.1.3.1.8 Build Dependencies

- [init.h](#) - System initialization
- [json_lexer.h](#) - JSON parsing library

12.22.1.3.1.9 Example Output

The application provides detailed console output showing:

- Parsed integer values from nested structures
- String extraction from various JSON locations
- Error handling for non-existent properties
- Type validation results
- Success/failure status of parse operations

12.22.2 NetBurner Demo Server for JSON Client Examples

12.22.2.1 NetBurner Demo Server

12.22.2.1.1 Overview

This is a comprehensive demonstration server application for the NetBurner platform that showcases various embedded web server capabilities including JSON handling, SSL/TLS support, tide calculations, and sunrise/sunset computations. The application serves as the backend for JSON client examples and provides a rich set of web services.

12.22.2.1.2 Features

12.22.2.1.2.1 Core Web Services

- **HTTP and HTTPS Server:** Dual protocol support with SSL/TLS encryption
- **JSON API Endpoints:** RESTful JSON services for client applications
- **Authentication System:** Multi-level access control with user authentication
- **Real-time Data Logging:** Stores and retrieves POST request data with timestamps

12.22.2.1.2.2 Specialized Functionality

- **Tide Calculations:** Real-time tide height computations
- **Astronomical Calculations:** Sunrise and sunset time calculations based on geographic coordinates
- **Time Zone Support:** Configurable time zone handling with NTP synchronization
- **Data Visualization:** Time-series data plotting for tide information

12.22.2.1.2.3 Security Features

- **SSL/TLS Encryption:** Secure communication channels
- **Access Control:** Password-protected endpoints with different security levels
- **Let's Encrypt Integration:** Automatic SSL certificate management via ACME protocol

12.22.2.1.3 API Endpoints

12.22.2.1.3.1 JSON Services

- `GetJson*` - Returns basic JSON response
- `PGetJson*` - Protected JSON endpoint requiring authentication
- `Post*` - Accepts JSON POST requests
- `PPost*` - Protected JSON POST endpoint
- `JPost*` - JSON POST with JSON response
- `PJPost*` - Protected JSON POST with JSON response

12.22.2.1.3.2 Data Retrieval

- `Show*` - HTML table view of stored POST data
- `JShow*` - JSON view of stored POST data
- `PlShow*` - JSON payload display for specific POST entries

12.22.2.1.3.3 Content Types

- `Text` - Plain text responses
- `Html` - HTML formatted responses
- `PText` - Protected text endpoint
- `PHtml` - Protected HTML endpoint

12.22.2.1.4 Configuration

12.22.2.1.4.1 Geographic Settings

- **Latitude:** Default 32.9834953 (North positive)
- **Longitude:** Default 117.2651886 (West negative)
- **Time Zone:** Default "PST8PDT7" (Pacific Standard/Daylight Time)

12.22.2.1.4.2 SSL Configuration

- **Certificate:** Requires valid SSL certificate and private key
- **Domain:** Configured for demo.netburner.com
- **ACME Support:** Automatic certificate renewal via Let's Encrypt

12.22.2.1.5 Data Storage

The application maintains a circular buffer of POST request data including:

- Timestamp (local and UTC)
- Client IP address
- Security status (HTTP/HTTPS)
- Request URL
- JSON payload data

Maximum storage: 256 entries with automatic rollover

12.22.2.1.6 Authentication Levels

1. **Level 0:** No authentication required
2. **Level 1:** HTTPS required (SSL/TLS encryption)
3. **Level 2:** Username/password authentication ("user"/"password")
4. **CONFIG_ACCESS_GROUP:** System-configured credentials

12.22.2.1.7 Real-time Calculations

12.22.2.1.7.1 Tide Calculations

- Continuous tide height monitoring
- 3-minute interval updates
- Historical data plotting
- Day/night awareness for display

12.22.2.1.7.2 Solar Calculations

- Sunrise/sunset times for specified coordinates
- Equation of time calculations
- Solar declination and hour angle computations
- Daylight/darkness detection

12.22.2.1.8 Build Requirements

12.22.2.1.8.1 Dependencies

- NetBurner development environment
- SSL certificate and private key files
- Network connectivity for NTP synchronization

12.22.2.1.8.2 Libraries Used

- [crypto/ssl.h](#) - SSL/TLS support
- [json_lexer.h](#) - JSON parsing
- [nbtime.h](#) - NTP time synchronization
- [httppost.h](#) - HTTP POST handling
- [acmeRFC8555Servlet.h](#) - Let's Encrypt ACME protocol

12.22.2.1.9 Network Configuration

The application automatically:

- Obtains IP address via [DHCP](#)
- Synchronizes time via NTP
- Starts both HTTP (port 80) and HTTPS (port 443) servers
- Enables system diagnostics

12.22.2.1.10 Usage

12.22.2.1.10.1 Deployment

1. Flash the application to a NetBurner device
2. Ensure SSL certificates are properly configured
3. Connect to network with internet access for NTP
4. Access via web browser at device IP address

12.22.2.1.10.2 Testing

- Use HTTP clients to test JSON endpoints
- Monitor POST data via Show* endpoints
- Verify SSL functionality with HTTPS connections

12.22.2.1.11 Development Notes

This application serves as a reference implementation demonstrating:

- Embedded web server best practices
- JSON API design patterns
- SSL/TLS integration
- Real-time data processing
- Astronomical calculation algorithms
- Time zone handling in embedded systems

Warning

This project requires a valid SSL certificate and private key to build successfully. The certificate should be configured for the target domain (default: demo.netburner.com).

12.22.3 Get JSON Object From Server

12.22.3.1 Get JSON From Server Application

A NetBurner embedded application that demonstrates retrieving JSON objects from various web servers and displaying the parsed results.

12.22.3.1.1 Overview

This application showcases HTTP client functionality on NetBurner embedded systems by fetching JSON data from multiple endpoints and parsing the responses. It demonstrates both HTTP and HTTPS requests, as well as authenticated requests using basic authentication.

12.22.3.1.2 Features

- **HTTP/HTTPS Support:** Fetches data from both secure and non-secure endpoints
- **JSON Parsing:** Automatically parses JSON responses and displays structured output
- **Authentication:** Supports basic HTTP authentication for protected endpoints
- **Error Handling:** Includes custom error reporting for HTTP failures
- **Multiple Data Sources:** Demonstrates various types of JSON APIs including:

- Simple configuration data
- Real-time earthquake data from USGS
- IP address information
- Custom NetBurner demo endpoints

12.22.3.1.3 Tested Endpoints

The application includes several predefined URLs for testing:

1. NetBurner Demo Endpoints:

- <http://demo.netburner.com/ut.json> - Simple JSON object
- <https://demo.netburner.com/ut.json> - Same endpoint over HTTPS
- <http://demo.netburner.com/GetJson> - Custom JSON endpoint
- <http://demo.netburner.com/PGetJson> - Password-protected endpoint
- <https://demo.netburner.com/PGetJson> - Secure password-protected endpoint

2. Public APIs:

- https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/4.5_↔day.geojson - USGS earthquake data
- <http://httpbin.org/ip> - IP address information
- <https://httpbin.org/ip> - Same over HTTPS

12.22.3.1.4 Code Structure

12.22.3.1.4.1 Main Components

- **Network Initialization:** Sets up the NetBurner network stack and waits for [DHCP](#)
- **HTTP Client:** Uses NetBurner's webclient library for HTTP requests
- **JSON Parser:** Leverages the built-in JSON lexer for parsing responses
- **Error Reporting:** Custom error handler for HTTP failures

12.22.3.1.4.2 Key Functions

- `GetAndDisplay()`: Core function that fetches and displays JSON from a given URL
- `MyReport()`: Custom error reporting function for HTTP failures
- `UserMain()`: Main application entry point

12.22.3.1.5 Usage

12.22.3.1.5.1 Building and Running

1. Compile the application using the NetBurner development environment
2. Flash the application to your NetBurner device
3. Monitor the serial output to see the JSON responses

12.22.3.1.5.2 Authentication

For password-protected endpoints, the application uses basic HTTP authentication:

```
GetAndDisplay(url5, "user", "password");
```

12.22.3.1.5.3 Customization

To test additional endpoints, modify the URL constants at the top of `main.cpp`:

```
const char *url_custom = "https://your-api-endpoint.com/data.json";
```

Then add the call in `UserMain()`:

```
GetAndDisplay(url_custom);
```

12.22.3.1.6 Dependencies

- NetBurner Runtime Library
- NetBurner HTTP Client Library ([webclient/http_funcs.h](#))
- NetBurner JSON Parser ([json_lexer.h](#))
- NetBurner Initialization Library ([init.h](#))

12.22.3.1.7 Output Format

The application displays:

1. The URL being requested
2. The parsed JSON object in a structured format
3. Any HTTP errors encountered

Example output:

```
Getting web site [https://httpbin.org/ip]
{
  "origin": "192.168.1.100"
}
```

12.22.3.1.8 Error Handling

The application includes comprehensive error handling:

- HTTP status code reporting
- Network connectivity checks
- JSON parsing error detection
- Timeout handling (60-second timeout per request)

12.22.3.1.9 Network Requirements

- Active network connection (Ethernet or WiFi)
- [DHCP](#) configuration (automatic IP assignment)
- Internet access for external APIs
- DNS resolution capability

12.22.3.1.10 Development Notes

- The application uses a 60-second timeout for all HTTP requests
- System diagnostics are enabled for debugging (should be disabled in production)
- The main loop includes a 1-second delay between iterations
- All URLs are tested sequentially at startup

12.22.4 Get Non-JSON Object From Server

12.22.4.1 Get Non-JSON From Server

A NetBurner embedded application that demonstrates retrieving non-JSON content from web servers using HTTP GET requests.

12.22.4.1.1 Overview

This application showcases how to use the NetBurner platform's HTTP client functionality to fetch various types of content from remote servers. The program demonstrates both authenticated and non-authenticated HTTP requests, making it useful for learning web client development on embedded systems.

12.22.4.1.2 Features

- HTTP GET requests to multiple endpoints
- Basic authentication support (username/password)
- Network initialization and [DHCP](#) configuration
- System diagnostics integration
- Response display and error handling

12.22.4.1.3 Application Flow

1. **Network Initialization:** Initializes the network stack and waits for [DHCP](#) address assignment
2. **HTTP Requests:** Makes GET requests to five different endpoints:
 - <http://demo.netburner.com/Html> - HTML content (no auth)
 - <http://demo.netburner.com/Text> - Text content (no auth)
 - <http://demo.netburner.com/PHtml> - Protected HTML content (requires auth)
 - <http://demo.netburner.com/PText> - Protected text content (requires auth)
 - <http://demo.netburner.com/SHtml.html> - Static HTML file (no auth)
3. **Response Processing:** Displays the retrieved content or error messages
4. **Continuous Operation:** Enters an infinite loop with 1-second delays

12.22.4.1.4 Authentication

The application demonstrates HTTP Basic Authentication for protected resources using:

- Username: `user`
- Password: `password`

12.22.4.1.5 Key Components

12.22.4.1.5.1 Functions

- `GetAndDisplay()`: Core function that performs HTTP GET requests and displays results
- `UserMain()`: Main application entry point that handles initialization and request sequencing

12.22.4.1.5.2 Dependencies

- [init.h](#) - Network stack initialization
- [json_lexer.h](#) - JSON parsing utilities (included but not used in this example)
- [webclient/http_funcs.h](#) - HTTP client functions

12.22.4.1.6 Configuration

- **Buffer Size:** 5000 bytes for response storage
- **Timeout:** 60 seconds for HTTP requests
- **Network Timeout:** 5 seconds for [DHCP](#) address acquisition

12.22.4.1.7 Build Requirements

- NetBurner development environment
- Network connectivity for testing
- Access to demo.netburner.com endpoints (or modify URLs for your own test servers)

12.22.4.1.8 Usage

1. Compile and flash the application to your NetBurner device
2. Ensure network connectivity is available
3. Monitor the serial output to see HTTP request results
4. The application will continuously run, making the requests once at startup

12.22.4.1.9 Output Format

For each successful request, the output shows:

```
Getting web site [URL]
Result = [number_of_bytes]
[response_content]
```

For failed requests:

```
Getting web site [URL]
Failed to contact server
```

12.22.4.1.10 Notes

- System diagnostics are enabled by default (consider disabling for production)
- HTTP diagnostics are available but commented out
- The application demonstrates handling both public and password-protected content
- Response content is displayed as raw text/HTML

12.22.4.1.11 Example Applications

This code pattern can be adapted for:

- Web scraping on embedded systems
- API data retrieval
- Remote configuration updates
- Status monitoring from web services
- Integration with web-based dashboards

12.22.5 JSON Linter Test

12.22.5.1 JSON Linter Test Application

12.22.5.1.1 Overview

This is a JSON linting test application built for NetBurner embedded systems. The application serves as a comprehensive test suite for validating JSON parsing capabilities, testing both valid JSON files (that should pass) and invalid JSON files (that should fail) to ensure the JSON parser correctly identifies various syntax errors and edge cases.

It will test all `bad*.json` and `pass*.json` files in `html` directory. To add additional tests add a bad or pass files to the `html` directory. `pass*.json` files are expected to pass and `bad*.json` are expected to fail.

Note

Some failures are considered warnings; they not legal JSON but we parse them for this test anyway.

12.22.5.1.2 Application Architecture

The application is built on the NetBurner RTOS platform and includes:

- **Web Server Integration:** Starts an HTTP server on port 80
- **Network Stack:** Full TCP/IP networking capabilities
- **JSON Parser Testing:** Comprehensive validation of JSON syntax
- **File System Integration:** Embedded file system containing test JSON files

12.22.5.1.3 Test Files

The application includes 47 JSON test files categorized into two groups:

12.22.5.1.3.1 Valid JSON Files (Should Pass)

- **pass1.json:** Comprehensive JSON test pattern with various data types, escape sequences, and formatting
- **pass2.json:** Deeply nested array structure (19 levels deep)
- **pass3.json:** Simple object validation test
- **pass4.json:** Complex configuration object with nested structures (network configuration example)

12.22.5.1.3.2 Invalid JSON Files (Should Fail)

Original Test Suite

- **bad1.json:** Invalid escape sequences (`\a`, `\x`, incomplete `\u` sequences)
- **bad2.json:** Unclosed arrays and objects, extra closing braces
- **bad3.json:** Missing opening brace
- **bad4.json:** Trailing commas in objects and arrays
- **bad5.json:** Invalid number formats (leading `+`, trailing decimals, hex notation, Infinity, NaN)
- **bad6.json:** Missing values after commas
- **bad7.json:** Missing commas between object properties
- **bad8.json:** Unquoted property names
- **bad9.json:** Complex nested errors (missing quotes, invalid numbers, trailing commas)
- **bad10.json:** Mismatched brackets and braces
- **bad11.json:** Comments in JSON (not allowed in strict JSON)
- **bad12.json:** Unescaped control characters in strings

Extended Test Suite

- **bad_2.json:** Unclosed array (missing closing bracket)
- **bad_3.json:** Unquoted object key (keys must be in quotes)
- **bad_4.json:** Trailing comma in array
- **bad_5.json:** Double trailing comma in array
- **bad_6.json:** Missing value at start of array (leading comma)
- **bad_7.json:** Comma after array close
- **bad_8.json:** Extra closing bracket
- **bad_9.json:** Trailing comma in object
- **bad_10.json:** Extra content after valid JSON (misplaced quoted value)
- **bad_11.json:** JavaScript expression instead of JSON value (1 + 2)
- **bad_12.json:** JavaScript function call instead of JSON value (alert())
- **bad_13.json:** Leading zeros in numbers (013)
- **bad_14.json:** Hexadecimal number notation (0x14)
- **bad_15.json:** Invalid backslash escape sequence (\x15)
- **bad_16.json:** Naked/unquoted value (not a string, number, boolean, or null)
- **bad_17.json:** Invalid octal escape sequence (\017)
- **bad_19.json:** Missing colon between key and value
- **bad_20.json:** Double colon (::)
- **bad_21.json:** Comma instead of colon in object
- **bad_22.json:** Colon instead of comma in array
- **bad_23.json:** Invalid boolean value ("truth" instead of "true")
- **bad_24.json:** Single quotes instead of double quotes for strings
- **bad_25.json:** Unescaped tab characters in string
- **bad_26.json:** Invalid escape sequences with tabs
- **bad_27.json:** Unescaped line break in string
- **bad_28.json:** Invalid line continuation with backslash
- **bad_29.json:** Incomplete scientific notation (0e)
- **bad_30.json:** Incomplete scientific notation with sign (0e+)
- **bad_31.json:** Invalid scientific notation with double sign (0e+-1)
- **bad_32.json:** Missing closing brace (comma instead of brace)
- **bad_33.json:** Mismatched brackets and braces (array opened, object closed)
- **bad_34.json:** Extra closing brace

12.22.5.1.4 Key Features Tested

12.22.5.1.4.1 String Handling

- Proper escape sequences (`\n`, `\r`, `\t`, `\"`, `\\`, `\/`, `\b`, `\f`)
- Unicode escape sequences (`\u` followed by 4 hex digits)
- Invalid escape sequences detection
- Control character validation
- Quote character validation (double quotes required, single quotes invalid)
- Unescaped control characters (tabs, line breaks)

12.22.5.1.4.2 Number Validation

- Integer formats
- Floating-point numbers
- Scientific notation (e/E notation)
- Invalid number formats (leading zeros, trailing decimals, hex notation)
- Incomplete scientific notation (missing exponent values)
- Invalid scientific notation syntax (double signs, incomplete expressions)

12.22.5.1.4.3 Punctuation and Syntax Errors

- Missing or extra commas
- Missing or extra colons
- Misplaced punctuation (comma instead of colon, colon instead of comma)
- Trailing commas in objects and arrays
- Missing closing brackets or braces
- Extra closing brackets or braces

12.22.5.1.4.4 Invalid Values and Expressions

- JavaScript expressions (arithmetic operations, function calls)
- Naked/unquoted values that aren't valid JSON primitives
- Extra content after valid JSON structures
- Invalid boolean values (non-standard true/false representations)
- Single-quoted strings (JSON requires double quotes)

12.22.5.1.4.5 Edge Cases

- Deeply nested structures
- Empty objects and arrays
- Whitespace handling
- Comment detection (should fail in strict JSON)

12.22.5.1.5 Code Structure

12.22.5.1.5.1 Main Components

```
void TestJson(int i, bool should_pass)
```

Core testing function that:

- Creates a `ParsedJsonDataSet` object from file data
- Runs `CheckLint()` to validate JSON syntax
- Reports success/failure with detailed error messages
- Handles both expected passes and failures

```
void UserMain(void *pd)
```

Application entry point that:

- Initializes network stack and web server
- Iterates through all embedded files
- Tests files starting with 'b' (bad) as failures
- Tests files starting with 'p' (pass) as successes

12.22.5.1.5.2 Dependencies

- **NetBurner RTOS:** Real-time operating system
- **HTML Files System:** Embedded file system for test data
- **JSON Lexer:** Custom JSON parsing library
- **Network Stack:** TCP/IP networking support

12.22.5.1.6 Expected Output

The application produces console output for each test:

- **Successful validation:** `OK for filename.json`
- **Expected failure:** `OK for filename.json [error_message]`
- **Unexpected pass:** `Testing filename.json should fail, it passes.`
- **Unexpected failure:** `Testing filename.json should pass, it fails. for error_↵
message`

12.22.5.1.7 Usage

1. Deploy the application to a NetBurner embedded device
2. The application automatically runs all JSON lint tests on startup
3. Monitor console output for test results
4. Access the embedded web server on port 80 for additional functionality

12.22.5.1.8 Development Notes

- The application uses embedded file records for test data
- All JSON files are compiled into the application binary
- The web server remains active after testing for additional functionality
- System diagnostics are enabled for debugging purposes

12.22.5.1.9 Test Coverage

This comprehensive test suite validates JSON parsing against a wide range of syntax errors and edge cases, ensuring robust JSON processing in embedded applications. The test suite covers:

- **Basic JSON Structure:** Objects, arrays, and proper nesting
- **Data Types:** Strings, numbers, booleans, null values
- **String Formatting:** Escape sequences, Unicode handling, control characters
- **Number Formats:** Integer, floating-point, scientific notation validation
- **Syntax Rules:** Comma placement, colon usage, bracket matching
- **Error Conditions:** All major categories of JSON syntax violations
- **Edge Cases:** Deeply nested structures, empty containers, whitespace handling

The test suite provides comprehensive coverage of JSON specification compliance and common parsing pitfalls, making it an excellent validation tool for JSON parser implementations.

12.22.6 Post JSON to Server and Display Result

12.22.6.1 JSON POST Application

12.22.6.1.1 Overview

This NetBurner application demonstrates how to POST JSON data to external servers and display the returned JSON results. The application showcases both secure (HTTPS) and insecure (HTTP) connections, with and without authentication.

12.22.6.1.2 Application Name

`PostAndGetJsonExternalServer`

12.22.6.1.3 Features

- **Multiple Connection Types:** Supports both HTTP and HTTPS connections
- **Authentication Support:** Demonstrates both authenticated and non-authenticated requests
- **JSON Handling:** Shows two methods for creating and sending JSON data:
 - Raw JSON strings
 - Programmatic JSON construction using [ParsedJsonDataSet](#)
- **Response Processing:** Parses and displays JSON responses from the server

12.22.6.1.4 Test Endpoints

The application connects to the following NetBurner demo endpoints:

- <http://demo.netburner.com/JPost> - HTTP JSON endpoint
- <https://demo.netburner.com/JPost> - HTTPS JSON endpoint
- <http://demo.netburner.com/PJPost> - HTTP JSON endpoint with authentication
- <https://demo.netburner.com/PJPost> - HTTPS JSON endpoint with authentication

12.22.6.1.5 Authentication

When authentication is required, the application uses:

- **Username:** `user`
- **Password:** `password`

12.22.6.1.6 Code Structure

12.22.6.1.6.1 Key Functions

PostAndDisplayTextJson()

Posts raw JSON strings to specified URLs and displays the response.

Parameters:**

- `purl` - Target URL
- `pJson` - JSON string to post
- `pUser` - Username (optional)
- `pPass` - Password (optional)

PostAndDisplayPJSJson()

Posts programmatically constructed JSON objects and displays the response.

Parameters:**

- `purl` - Target URL
- `JsonOut` - [ParsedJsonDataSet](#) object containing JSON data
- `pUser` - Username (optional)
- `pPass` - Password (optional)

12.22.6.1.6.2 Main Execution Flow

1. **Network Initialization:** Initializes the network stack and waits for [DHCP](#)
2. **String-based JSON Posts:** Sends predefined JSON strings to all four endpoints
3. **Programmatic JSON Posts:** Constructs JSON objects programmatically and posts them
4. **Continuous Loop:** Enters an infinite loop with 1-second delays

12.22.6.1.7 JSON Examples

12.22.6.1.7.1 Raw JSON Strings

```
{ "test": "Simple" }
{ "test": "Secure" }
{ "test": "InSecure w Password" }
{ "test": "Secure w Password" }
```

12.22.6.1.7.2 Programmatically Built JSON

```
{
  "BuildByPJS": true,
  "Simple": "example"
}
{
  "BuildByPJS": true,
  "Secure": "WPassword"
}
```

12.22.6.1.8 Dependencies

- `init.h` - Network stack initialization
- `json_lexer.h` - JSON parsing and construction
- `webclient/http_funcs.h` - HTTP client functions

12.22.6.1.9 Build and Run

1. Compile the application using your NetBurner development environment
2. Deploy to your NetBurner device
3. Ensure network connectivity is available
4. Monitor the serial output to see the JSON POST results

12.22.6.1.10 Output

The application will display:

- Connection attempts to each URL
- Parsed JSON responses from successful requests
- Error messages for failed connections

12.22.6.1.11 Notes

- The application includes system diagnostics (should be disabled for production)
- HTTP diagnostic output is available but commented out
- All requests have a 60-second timeout
- The application runs indefinitely, making requests once at startup

12.22.6.1.12 Security Considerations

This is a demonstration application that includes:

- Hardcoded credentials
- Mixed HTTP/HTTPS usage
- System diagnostics enabled

For production use, consider:

- Secure credential storage
- Proper error handling and retry logic

12.22.7 Post JSON to Server

12.22.7.1 NetBurner JSON POST Application

12.22.7.1.1 Overview

This application demonstrates how to POST JSON data to external web servers using the NetBurner embedded platform. The program showcases both secure (HTTPS) and insecure (HTTP) connections, with and without authentication, and displays the server responses as text.

12.22.7.1.2 Application Name

PostJsonGetTextExternalServer

12.22.7.1.3 Features

- HTTP and HTTPS JSON POST requests
- Basic authentication support (username/password)
- Multiple JSON creation methods:
 - Direct JSON string formatting
 - [ParsedJsonDataSet](#) object for structured JSON building
- Response handling and display
- Network initialization and [DHCP](#) support
- System diagnostics capability

12.22.7.1.4 Test Endpoints

The application connects to four different NetBurner demo server endpoints:

1. <http://demo.netburner.com/Post> - HTTP without authentication
2. <https://demo.netburner.com/Post> - HTTPS without authentication
3. <http://demo.netburner.com/PPost> - HTTP with authentication
4. <https://demo.netburner.com/PPost> - HTTPS with authentication

12.22.7.1.5 Authentication

For authenticated endpoints, the application uses:

- Username: `user`
- Password: `password`

12.22.7.1.6 JSON Data Examples

The application sends various JSON payloads:

12.22.7.1.6.1 Direct JSON Strings

```
{ "test": "Simple" }
{ "test": "Secure" }
{ "test": "InSecure w Password" }
{ "test": "Secure w Password" }
```

12.22.7.1.6.2 ParsedJsonDataSet Generated JSON

```
{ "BuildByPJS": true, "Simple": "example" }
{ "BuildByPJS": true, "Secure": "WPassword" }
```

12.22.7.1.7 Code Structure

12.22.7.1.7.1 Main Functions

- `UserMain()` - Main application entry point
- `PostJsonDisplayText()` - Posts JSON strings to servers
- `PostPJSJsonDisplayText()` - Posts [ParsedJsonDataSet](#) objects to servers

12.22.7.1.7.2 Key Components

1. Network Initialization

- Initializes the network stack
- Enables system diagnostics
- Waits for [DHCP](#) address assignment

2. JSON Posting

- Uses [DoJsonPost \(\)](#) function for HTTP/HTTPS requests
- Handles authentication headers when required
- Manages response buffering

3. Response Handling

- Captures server responses in a 4KB buffer
- Null-terminates response strings
- Displays results or error messages

12.22.7.1.8 Usage

1. Compile and flash the application to your NetBurner device
2. Ensure network connectivity ([DHCP](#) or static IP)
3. Monitor serial output for:
 - Network initialization status
 - POST request URLs
 - Server responses
 - Any error messages

12.22.7.1.9 Output Format

The application prints status messages and results to the serial console:

```
Application started
Posting to web site [http://demo.netburner.com/Post]
Result:[Server response here]
Posting to web site [https://demo.netburner.com/Post]
Result:[Server response here]
...
```

12.22.7.1.10 Error Handling

- Network connectivity issues result in "Failed to contact server" messages
- Authentication failures are handled by the underlying HTTP client
- Buffer overflow protection with 4KB response limit

12.22.7.1.11 Dependencies

- NetBurner Runtime Libraries
- [init.h](#) - Network initialization
- [json_lexer.h](#) - JSON parsing and building
- [webclient/http_funcs.h](#) - HTTP client functions

12.22.7.1.12 Configuration

- Response buffer size: 4096 bytes
- Network timeout: 5 seconds for [DHCP](#)
- HTTP timeout: 60 seconds per request
- Main loop delay: 1 second

12.22.7.1.13 Notes

- The application runs in an infinite loop after completing all POST requests
- System diagnostics are enabled for debugging (consider disabling for production)
- HTTP diagnostics can be enabled by uncommenting `SetHttpDiag(true)`
- All endpoints use NetBurner's demo server for testing purposes

12.22.8 Get Application Image and Load it From a Web Server

12.22.8.1 LoadCodeFromServer

A NetBurner embedded application that demonstrates retrieving and loading new firmware images from a web server.

12.22.8.1.1 Overview

This application showcases how to perform over-the-air (OTA) firmware updates on NetBurner embedded devices. It connects to a web server, downloads a new application image, and loads it onto the device for execution.

12.22.8.1.2 Features

- **Network Initialization:** Automatically initializes the network stack and waits for [DHCP](#) configuration
- **Web Client Functionality:** Downloads files from web servers using HTTP GET requests
- **Authentication Support:** Supports HTTP basic authentication for secured downloads
- **Firmware Update Process:** Implements the complete OTA update workflow
- **Interactive Interface:** Simple command-line interface for triggering updates

12.22.8.1.3 How It Works

1. **Network Setup:** The application initializes the network stack and waits for an active network connection via [DHCP](#)
2. **URL Configuration:** Constructs a platform-specific URL pointing to the firmware image on AWS S3
3. **User Interaction:** Waits for user input to trigger the update process
4. **Download and Install:** Downloads the new firmware image and attempts to install it
5. **Automatic Reboot:** On successful installation, the device reboots with the new firmware

12.22.8.1.4 Usage

12.22.8.1.4.1 Basic Operation

1. Deploy the application to your NetBurner device
2. Connect to the device's console/terminal
3. Wait for the network to initialize and receive a [DHCP](#) address
4. When prompted, press 'U' to start the update process
5. The application will download and install the new firmware

12.22.8.1.4.2 Console Output

The application provides detailed console output including:

- Network initialization status
- Update prompts and instructions
- Download progress and results
- Error messages if the update fails

12.22.8.1.5 Code Structure

12.22.8.1.5.1 Main Components

- `**UserMain()`: Primary application entry point that handles initialization and user interaction
- `**GetAndDisplay()`: Utility function for downloading and displaying web content (used for demonstration)
- **Network Stack**: Utilizes NetBurner's built-in networking capabilities
- **HTTP Client**: Uses NetBurner's webclient library for HTTP operations

12.22.8.1.5.2 Key Libraries Used

- `init.h`: Network stack initialization
- `json_lexer.h`: JSON parsing capabilities
- `webclient/http_funcs.h`: HTTP client functions
- `config_obj.h`: Configuration management

12.22.8.1.6 Configuration

The application is configured to download firmware from:

`https://netburner-apps.s3.amazonaws.com/{PlatformName}/SimpleHtml.bin`

Where `{PlatformName}` is automatically determined based on the target NetBurner platform.

12.22.8.1.7 Security Considerations

- The application includes support for HTTP basic authentication
- Downloads are performed over HTTPS for security
- The update process includes validation to prevent corrupted installations

12.22.8.1.8 Error Handling

The application includes comprehensive error handling:

- Network connectivity verification
- HTTP request timeout handling (60 seconds)
- Download validation
- Graceful failure recovery with user feedback

12.22.8.1.9 Development Notes

- System diagnostics are enabled for debugging (should be disabled in production)
- HTTP diagnostics can be enabled by uncommenting diagnostic flags
- The application uses a 5KB buffer for HTTP responses
- Network timeout is set to 5 seconds for initial connection

12.22.8.1.10 Platform Compatibility

This application is designed for NetBurner embedded devices and requires:

- NetBurner RTOS
- Active network connection (Ethernet or WiFi)
- Sufficient flash memory for firmware storage
- Internet connectivity to reach the update server

12.22.8.1.11 Building and Deployment

1. Ensure you have the NetBurner development environment set up
2. Compile the application using the NetBurner toolchain
3. Deploy to your NetBurner device using standard deployment methods
4. Connect to the device console to interact with the application

12.22.8.1.12 Safety Notes

- **Backup Your Firmware:** Always maintain a backup of your current firmware before performing updates
- **Stable Network:** Ensure a stable network connection during updates to prevent corruption
- **Power Supply:** Maintain stable power during the update process
- **Testing:** Test updates in a development environment before deploying to production devices

12.22.8.1.13 Troubleshooting

12.22.8.1.13.1 Common Issues

- **Network Connection Failed:** Verify [DHCP](#) configuration and network connectivity
- **Download Failed:** Check internet connectivity and server availability
- **Update Failed:** Ensure sufficient flash memory and stable power supply
- **Authentication Issues:** Verify credentials if using authenticated downloads

12.22.8.1.13.2 Debug Options

Enable diagnostic output by uncommenting:

```
SetHttpDiag(true);  
SetHttpUpDiag(true);
```

12.22.8.1.14 Example Output

```
Application started  
  
Hit U to update from  
[https://netburner-apps.s3.amazonaws.com/PLATFORM/SimpleHtml.bin]  
Starting update process...  
update returned /failed rv=0
```

12.22.9 Simple JSON HTML Example

12.22.9.1 NetBurner Simple JSON HTML Example

An example application demonstrating various methods for generating JSON responses from a NetBurner embedded web server.

12.22.9.1.1 Overview

This application showcases multiple approaches to serve JSON data from a NetBurner device, including static files, dynamic generation, and configuration-based responses. It provides a practical reference for developers working with NetBurner's web capabilities.

Shows several methods to have a NetBurner respond to web requests with JSON objects:

- Respond and generate the text directly.
- Respond and generate the response with a JSON lever object.
- Respond and generate the response using configuration variables.
- Respond and generate the response using a pre-built page stored in with web pages and using the dynamic web page capabilities.

12.22.9.1.2 Features

The application demonstrates several JSON response methods:

12.22.9.1.2.1 1. Static JSON Files

- **plain.json**: Pre-formatted static JSON content
- Simple text-based JSON stored as a file

12.22.9.1.2.2 2. Variable-Based Dynamic JSON

- **ut.json**: Template-based JSON with embedded variables
- Uses NetBurner's variable substitution system
- Displays runtime information like uptime, build date/time, and NNDK version

12.22.9.1.2.3 3. Callback-Based Dynamic JSON

- **callback.json**: Uses C++ callback functions to generate JSON
- Demonstrates the `<!--CPPCALL DoGeneerateJson -->` mechanism
- Generates JSON content programmatically at request time

12.22.9.1.2.4 4. Function-Based JSON Responses

- **GetJson**: Direct HTTP callback without associated HTML file
- **GetLexerJson**: Uses NetBurner's JSON lexer for structured generation
- Provides complete HTTP response handling in C++ code

12.22.9.1.2.5 5. Configuration System Integration

- **Config**: Exposes the entire configuration tree as JSON
- **Config/AppData/MyOwnValue**: Access specific configuration values
- Requires enabling the configuration mirror

12.22.9.1.3 File Structure

```
/
main.cpp          # Main application code
htmlvar.h         # Variable declarations for web pages
index.html        # Main navigation page
plain.json        # Static JSON example
ut.json           # Variable-based JSON template
callback.json     # Callback-based JSON page
```

12.22.9.1.4 Key Components

12.22.9.1.4.1 Main Application (main.cpp)

- Initializes network stack and HTTP server
- Defines global variables for web page access
- Implements callback functions for dynamic JSON generation
- Sets up configuration system integration

12.22.9.1.4.2 JSON Generation Methods

Method 1: Direct Text Output

```
int BuildAJsonResponseByText(int sock, HTTP_Request &pd)
{
    writestring(sock, "HTTP/1.0 200 OK\r\nPragma: no-cache\r\nContent-Type: application/json\r\n\r\n");
    writestring(sock, "{\"Test\":\"Json\"}");
    return 1;
}
```

Method 2: JSON Lexer

```
int BuildAJsonResponseByLexer(int sock, HTTP_Request &pd)
{
    ParsedJsonDataSet pjs;
    pjs.StartBuilding();
    pjs.Add("ANumber", 1234);
    pjs.Add("AString", "Testing123");
    pjs.AddMyMac("MyMacAddress");
    pjs.DoneBuilding();
    pjs.PrintObjectToFd(sock, true);
    return 1;
}
```

Method 3: Callback Function

```
void DoGenerateJson(int sock, PCSTR url)
{
    fdprintf(sock, "{\r\n\"Generated\":\"ByCallback\", \r\n");
    fdprintf(sock, "\"Uptime\":%ld\r\n", Secs);
}
```

12.22.9.1.5 Usage

1. **Build and Deploy:** Compile the application and deploy to your NetBurner device
2. **Access Web Interface:** Navigate to your device's IP address (default port 80)
3. **Test JSON Endpoints:** Use the provided links to test different JSON generation methods

12.22.9.1.5.1 Example URLs

Assuming your device IP is 10.1.2.3:

- <http://10.1.2.3/> - Main navigation page
- <http://10.1.2.3/plain.json> - Static JSON
- <http://10.1.2.3/ut.json> - Variable-based JSON
- <http://10.1.2.3/callback.json> - Callback-generated JSON
- <http://10.1.2.3/GetJson> - Direct function response
- <http://10.1.2.3/GetLexerJson> - JSON lexer response
- <http://10.1.2.3/Config> - Full configuration tree
- <http://10.1.2.3/Config/AppData/MyOwnValue> - Specific config value

12.22.9.1.6 Configuration

The application includes a configuration variable example:

```
config_int gMyOwnVal{appdata, 199, "MyOwnValue"};
```

This demonstrates how to expose application-specific configuration data through the web interface.

12.22.9.1.7 Build Requirements

- NetBurner NNDK (NetBurner Network Development Kit)
- C++ compiler with NetBurner libraries
- Standard NetBurner build environment

12.22.9.1.8 Key Features Demonstrated

- HTTP server initialization and setup
- Multiple JSON response generation techniques
- Variable substitution in web pages
- Callback-based dynamic content generation
- Configuration system integration
- Proper HTTP header handling
- Network stack initialization

12.22.9.1.9 Notes

- The application enables system diagnostics (should be removed for production)
- Configuration mirror must be enabled for config-based endpoints
- Default HTTP port is 80
- Application waits for active network connection before proceeding

12.22.9.1.10 Learning Objectives

This example is ideal for developers who want to understand:

- NetBurner web server capabilities
- JSON API development on embedded systems
- Dynamic content generation techniques
- Configuration system integration
- HTTP request handling in embedded C++

12.22.10 Simple JSON Post Receiver Example

12.22.10.1 Simple JSON Post Receiver Example

A NetBurner NNDK application that demonstrates how to receive JSON objects via HTTP POST requests from external RESTful APIs or JavaScript within web pages.

12.22.10.1.1 Overview

This application sets up a web server that can handle JSON POST requests through multiple endpoints. It provides both raw text processing and parsed JSON data handling capabilities, making it suitable for various integration scenarios.

12.22.10.1.2 Features

- **Multiple POST Endpoints:** Handles different types of POST requests
- **JSON Parsing:** Uses NetBurner's JSON lexer for structured data processing
- **Raw Text Processing:** Can handle plain text POST data
- **Configuration Integration:** Supports posting to the device's configuration system
- **Web Interface:** Includes a simple HTML interface with usage examples

12.22.10.1.3 POST Endpoints

The application provides three main POST endpoints:

12.22.10.1.3.1 1. /PostText

- Handles raw text POST data
- Processes content as plain text without JSON parsing
- Useful for simple data transmission

12.22.10.1.3.2 2. /PostLexer

- Processes JSON POST data using NetBurner's JSON lexer
- Validates JSON structure and provides lint checking
- Outputs parsed JSON object structure

12.22.10.1.3.3 3. /Config/AppData/MyOwnValue

- Allows posting directly to the configuration system
- Updates configuration values via HTTP POST
- Integrated with NetBurner's configuration server

12.22.10.1.4 Usage Examples

The application can be tested using the `wget` command-line utility:

12.22.10.1.4.1 Send JSON to Text Processor

```
wget -O- --post-data='{ "Data":1}' --header='Content-Type:application/json' 'http://[device-ip]/PostText'
```

12.22.10.1.4.2 Send JSON to Lexer Processor

```
wget -O- --post-data='{ "Data":2}' --header='Content-Type:application/json' 'http://[device-ip]/PostLexer'
```

12.22.10.1.4.3 Update Configuration Value

```
wget -O- --post-data='{ "MyOwnValue":8765}' --header='Content-Type:application/json'
'http://[device-ip]/Config/AppData/MyOwnValue'
```

12.22.10.1.5 Code Structure

12.22.10.1.5.1 Main Components

- ****main.cpp**:** Core application logic and POST handlers
- ****htmlvar.h**:** Header file for HTML variable functions
- ****index.html**:** Web interface with usage instructions
- ****ReadMe.txt**:** Original documentation

12.22.10.1.5.2 Key Functions

- `**GetHost ()`: Retrieves the host information from HTTP requests
- `**SendResponse ()`: Sends standardized HTTP responses
- `**MyJsonPostHandlerText ()`: Processes raw text POST data
- `**MyJsonPostHandler ()`: Processes parsed JSON data

12.22.10.1.5.3 Configuration

The application includes a configuration integer `gMyOwnVal` with:

- Default value: 199
- Configuration path: `appdata`
- Name: "MyOwnValue"

12.22.10.1.6 Technical Details

12.22.10.1.6.1 Dependencies

- NetBurner NNDK framework
- HTTP server functionality
- JSON lexer library
- Configuration server support

12.22.10.1.6.2 Network Setup

- Initializes network stack on startup
- Starts HTTP server on default port 80
- Waits for active network connection
- Enables configuration mirror for remote management

12.22.10.1.6.3 Error Handling

- JSON lint validation with error reporting
- Timeout handling for POST data reception
- Network connection status monitoring

12.22.10.1.7 Development Notes

- System diagnostics are enabled (should be removed for production)
- Uses NetBurner's real-time operating system (NBRTOS)
- Includes proper memory management and timeout handling
- Supports both synchronous and asynchronous POST processing

12.22.10.1.8 Getting Started

1. Compile the application using NetBurner NNDK
2. Deploy to your NetBurner device
3. Access the web interface at `http://[device-ip]/`
4. Test POST endpoints using the provided wget examples
5. Monitor console output for debugging information

12.22.10.1.9 Response Format

All POST requests return a simple HTML response indicating successful processing:

```
<html><body>Post has been Processed by [processor-name]</body></html>
```

This application serves as a foundation for building more complex JSON-based communication systems with NetBurner devices.

12.23 malloc

12.23.1 Malloc Example Application

12.23.1.1 Overview

This application demonstrates dynamic memory allocation and deallocation using `malloc()` and `free()` functions. It showcases how to monitor heap space usage and illustrates the concept of memory fragmentation in embedded systems.

12.23.1.2 Purpose

The primary goals of this example are to:

- Demonstrate proper usage of `malloc()` and `free()` functions
- Show how to track heap space usage using `spaceleft()` and `mallinfo()`
- Illustrate memory fragmentation effects when freeing memory in different orders
- Provide insights into heap memory management for embedded applications

12.23.1.3 Key Features

12.23.1.3.1 Memory Tracking Functions

The application uses two important functions to monitor heap status:

1. `**spaceleft()` - Returns the amount of "untouched" memory (never allocated)
2. `**mallinfo()` - Returns detailed heap information including both untouched and freed memory

12.23.1.3.2 mallinfo Structure

The `mallinfo()` function returns a structure containing:

- `arena` - Total space allocated from system
- `ordblks` - Number of non-inuse chunks
- `smlblks` - Unused (always zero)
- `hblks` - Number of mmapped regions
- `hblkhd` - Total space in mmapped regions

- `usmblks` - Unused (always zero)
- `fsmblks` - Unused (always zero)
- `uordblks` - Total allocated space
- `fordblks` - Total non-inuse space
- `keepcost` - Top-most, releasable space

12.23.1.4 Application Flow

The application runs in a continuous loop performing the following sequence:

12.23.1.4.1 Allocation Phase

1. Allocates 1,000,000 bytes (pA)
2. Allocates 3,000,000 bytes (pB)
3. Allocates 512,000 bytes (pC)

12.23.1.4.2 Deallocation Phase

The memory is freed in a different order than allocated:

1. Frees pB (3,000,000 bytes)
2. Frees pA (1,000,000 bytes)
3. Frees pC (512,000 bytes)

After each allocation and deallocation, the heap status is displayed showing:

- Used memory
- Space left (untouched memory)
- Total free memory

12.23.1.5 Memory Fragmentation Example

This application demonstrates an important concept in dynamic memory management. When memory is freed in a different order than it was allocated, it can lead to fragmentation:

12.23.1.5.1 Example Scenario (1MB heap):

1. Allocate 100KB three times (A, B, C) - 700KB remains
2. Free A and B - `spaceleft()` still reports 700KB (watermark)
3. `mallinfo()` reports 900KB (including freed memory)

The `spaceleft()` function acts as a "watermark" showing the largest contiguous block available, while `mallinfo()` shows total available memory including fragmented blocks.

12.23.1.6 Key Observations

- **Order Matters:** The order of memory deallocation affects heap fragmentation
- **Watermark Effect:** `spaceleft()` value cannot increase until the last allocated block is freed
- **Monitoring is Critical:** Using both functions provides complete heap visibility
- **Production Considerations:** System diagnostics should be disabled in production code

12.23.1.7 Code Structure

- `ShowHeapSpace()` - Displays current heap statistics
- `UserMain()` - Main application loop with allocation/deallocation cycle
- Error handling for failed malloc operations
- 10-second delay between cycles for observation

12.23.1.8 Usage Notes

- The application includes system diagnostics that should be removed for production
- Memory allocation failures are handled gracefully with error messages
- The continuous loop allows observation of memory behavior over time
- All allocated memory is properly freed to prevent memory leaks

12.23.1.9 Educational Value

This example is particularly valuable for understanding:

- Dynamic memory management in embedded systems
- Memory fragmentation effects
- Proper error handling for memory allocation
- Heap monitoring techniques
- The difference between total available memory and contiguous available memory

12.23.1.10 Platform Requirements

- RTOS environment (uses `OSTimeDly`)
- Hardware abstraction layer (HAL)
- Standard C library with malloc/free support
- Support for `mallinfo()` function

12.24 MQTT Examples

Easy [MQTT](#) Object Examples:

- [MQTT Config Object Example](#)
- [MQTT Custom Object Example](#)
- [MQTT Object I/O Example](#)
- [MQTT Simple Object Pub/Sub Example](#)
- [MQTT SubHandler GPIO Example](#)

Advanced Examples:

- [Advanced MQTT Periodic Example](#)
- [Advanced MQTT Pub/Sub Example](#)
- [Advanced MQTT Echo Example](#)

12.24.1 MQTT Config Object Example

12.24.1.1 MQTT Config Object Application

12.24.1.1.1 Overview

This NetBurner application demonstrates how to expose the device's configuration system through [MQTT](#) messaging. It allows remote configuration of NetBurner devices by sending JSON configuration messages to specific [MQTT](#) topics, providing a convenient way to manage device settings without requiring direct HTTP access to the configuration system.

12.24.1.1.2 Application Features

- **Remote Configuration:** Modify device configuration remotely via [MQTT](#) messages
- **Dynamic Topic Generation:** Automatically creates device-specific topics based on MAC address
- **JSON Configuration Support:** Accepts standard NetBurner configuration JSON format
- **Bidirectional Communication:** Supports both reading and writing configuration values
- **Rate Limiting:** Implements [MQTT](#) publish rate limiting for network stability
- **Interactive Testing:** Provides console-based testing interface

12.24.1.1.3 How It Works

12.24.1.1.3.1 Topic Structure

The application creates a device-specific [MQTT](#) topic using the format:

`device/nburn_<MAC_SUFFIX>/config`

Where `<MAC_SUFFIX>` is the last 3 octets of the device's MAC address. For example:

- If MAC address is `00:03:F4:1A:2B:3C`, the topic becomes `device/nburn_1A2B3C/config`

12.24.1.1.3.2 Configuration Messages

Send JSON configuration messages to the device topic to modify settings. The JSON format follows the standard NetBurner configuration structure.

Example Configuration Message:**

```
{
  "Sys": {
    "Application": "My Custom App Name"
  }
}
```

This is equivalent to sending the same configuration via the HTTP configuration interface.

12.24.1.1.4 Usage

12.24.1.1.4.1 Prerequisites

- NetBurner device with [MQTT](#) support
- [MQTT](#) broker (configure connection details via NetBurner config system)
- Network connectivity

12.24.1.1.4.2 Setup

1. **Configure [MQTT](#) Connection:** Use the NetBurner configuration system to set:
 - [MQTT](#) broker URI
 - Username/password (if required)
 - SSL certificates (if using secure connection)
2. **Deploy Application:** Flash the application to your NetBurner device
3. **Network Connection:** Ensure the device has active network connectivity

12.24.1.1.4.3 Testing

Remote Configuration

1. Connect to your [MQTT](#) broker using any [MQTT](#) client
2. Send a JSON message to the device's config topic
3. The device will apply the configuration changes immediately

Interactive Testing

1. Connect to the device via serial console
2. Press any key when prompted
3. The application will send a test configuration message
4. Observe the response and applied changes

12.24.1.1.4.4 Example Configuration Messages

Change Application Name:**

```
{
  "Sys": {
    "Application": "Production Device v1.0"
  }
}
```

Modify Network Settings:**

```
{
  "Sys": {
    "NetIf": {
      "Ethernet0": {
        "DHCP": false,
        "IPAddr": "192.168.1.100"
      }
    }
  }
}
```

12.24.1.1.5 Code Structure

12.24.1.1.5.1 Key Components

- **MqttClient**: Configured [MQTT](#) client with connection management
- **mqttConfigTree**: [MQTT](#) configuration exposier that handles topic subscription and message processing
- **Rate Limiting**: Configured with 20 messages per 5-second window, burst of 4 messages
- **Connection Monitoring**: Waits for established [MQTT](#) connection before proceeding

12.24.1.1.5.2 Main Application Flow

1. Initialize file system (if not available)
2. Initialize network stack
3. Enable system diagnostics
4. Wait for active network connection
5. Configure [MQTT](#) rate limiting
6. Establish [MQTT](#) connection
7. Enter interactive testing loop

12.24.1.1.6 Configuration Options

The application uses the NetBurner configuration system for MQTT connection settings:

- **Broker Configuration:** Set via `MqttClient` configuration object
- **SSL/TLS:** Certificates stored in file system
- **Rate Limiting:** 20 messages per 5 seconds, burst capacity of 4 messages

12.24.1.1.7 Error Handling

- **Connection Monitoring:** Continuously monitors MQTT connection status
- **Network Dependency:** Waits for active network before attempting MQTT connection
- **Rate Limiting:** Prevents overwhelming the MQTT broker with messages

12.24.1.1.8 Security Considerations

- **Topic Naming:** Uses MAC address suffix for device identification
- **SSL Support:** Supports secure MQTT connections with certificates
- **Local Message Filtering:** Configured to ignore messages from the same client

12.24.1.1.9 Dependencies

- NetBurner NNDK
- MQTT library support
- File system support (for certificate storage)
- Network stack

12.24.1.1.10 Troubleshooting

12.24.1.1.10.1 Common Issues

1. **Connection Failures:** Verify MQTT broker settings and network connectivity
2. **Topic Not Found:** Check MAC address format in topic name
3. **Configuration Not Applied:** Ensure JSON format matches NetBurner config structure
4. **Rate Limiting:** Reduce message frequency if experiencing delays

12.24.1.1.10.2 Debugging

- Enable system diagnostics for detailed logging
- Monitor serial console for connection status messages
- Verify MQTT broker logs for message delivery

12.24.1.1.11 Production Deployment

For production use:

1. Remove or disable system diagnostics
2. Configure appropriate SSL certificates
3. Set production MQTT broker settings
4. Implement proper error handling and logging
5. Consider removing interactive testing code

12.24.2 MQTT Custom Object Example

12.24.2.1 MQTT Custom Object Example

12.24.2.1.1 Overview

This application demonstrates an industrial control system that exposes thermostat and PID controller data over [MQTT](#). It provides a practical example of how to implement bidirectional [MQTT](#) communication for industrial automation, allowing both local and remote monitoring and control of temperature systems and PID control loops. The application creates custom [MQTT](#) objects that automatically serialize to JSON and can be controlled through [MQTT](#) topic subscriptions, making it ideal for IoT industrial applications.

12.24.2.1.2 Features

12.24.2.1.2.1 Thermostat Control

- **Temperature Range Management:** Configurable minimum, maximum, and set point temperatures
- **Location Tracking:** Configurable location string (default: "Warehouse 1")
- **Active State Control:** Boolean flag to enable/disable thermostat operation
- **Real-time Monitoring:** Continuous JSON publication of thermostat status

12.24.2.1.2.2 PID Controller Sets

- **Dual PID Controllers:** Separate horizontal and vertical PID control sets
- **Tunable Parameters:** Individual P (Proportional), I (Integral), and D (Differential) terms
- **Industrial Ready:** Designed for precision control applications

12.24.2.1.2.3 MQTT Integration

- **Automatic JSON Serialization:** All objects automatically convert to JSON format
- **Bidirectional Communication:** Read current values and write new configurations
- **Topic-based Organization:** Clean topic structure for easy integration
- **Configurable Client:** Uses [MQTT](#) ConfiguredClient for flexible broker connections

12.24.2.1.3 Architecture

12.24.2.1.3.1 Class Hierarchy

```
mqtt_obj (base class)
  TempRange
    MaxTemp (mqtt_int)
    SetTemp (mqtt_int)
    MinTemp (mqtt_int)
  Thermostat (extends TempRange)
    Active (mqtt_bool)
    Location (mqtt_string)
  PIDSet
    P (mqtt_float) - Proportional term
    I (mqtt_float) - Integral term
    D (mqtt_float) - Differential term
  ControlSet
    Hset (PIDSet) - Horizontal controller
    Vset (PIDSet) - Vertical controller
```

12.24.2.1.3.2 MQTT Topics

Topic	Purpose	Data Format
Thermostat	Thermostat status and control	JSON with temperature ranges, active state, and location
ctrl_set	PID controller parameters	JSON with horizontal and vertical PID settings

12.24.2.1.4 Default Configuration

12.24.2.1.4.1 Thermostat Defaults

- **Maximum Temperature:** 100 deg
- **Set Temperature:** 30 deg
- **Minimum Temperature:** 10 deg
- **Active State:** true
- **Location:** "Warehouse 1"

12.24.2.1.4.2 PID Controller Defaults

- **Proportional (P):** 1.0
- **Integral (I):** 0.0001
- **Differential (D):** 0.0

12.24.2.1.5 Usage

12.24.2.1.5.1 Running the Application

1. The application initializes the network stack and waits for network connectivity
2. Connects to the configured [MQTT](#) broker
3. Publishes current object states as JSON to their respective topics
4. Subscribes to the same topics for remote configuration updates
5. Continuously displays current status to console

12.24.2.1.5.2 Interactive Mode

- Press any key during operation to toggle between display and control modes
- The application will show real-time JSON representations of both the thermostat and control sets
- System diagnostics are enabled for monitoring and debugging

12.24.2.1.5.3 MQTT Communication Examples

Thermostat JSON Structure:**

```
{
  "MaxTemp": 100,
  "SetTemp": 30,
  "MinTemp": 10,
  "Active": true,
  "Thermostat Location": "Warehouse 1"
}
```

Control Set JSON Structure:**

```
{
  "Hset": {
    "P": 1.0,
    "I": 0.0001,
    "D": 0.0
  },
  "Vset": {
    "P": 1.0,
    "I": 0.0001,
    "D": 0.0
  }
}
```

12.24.2.1.6 Configuration

The [MQTT](#) client configuration (broker URI, username, password, certificates) is managed through the NetBurner configuration system. Ensure proper [MQTT](#) broker settings are configured before running the application.

12.24.2.1.6.1 Prerequisites

- NetBurner RTOS platform
- Network connectivity
- [MQTT](#) broker access
- Filesystem support (for certificate storage)

12.24.2.1.7 Technical Implementation

12.24.2.1.7.1 Object Behavior

- **Auto-Publication:** Objects automatically publish to [MQTT](#) when values change (`eObj_Flag_PubOnWrite`)
- **Subscription:** Objects subscribe to their topics for remote updates (`eObj_Flag_Subscribe`)
- **JSON Serialization:** Built-in JSON rendering for all object types
- **Type Safety:** Strongly typed [MQTT](#) objects (int, float, bool, string)

12.24.2.1.7.2 Network Handling

- Automatic re-connection logic
- Connection status monitoring
- Graceful handling of network interruptions
- Configurable timeout and retry mechanisms

12.24.2.1.8 Use Cases

This example is ideal for:

- Industrial temperature control systems
- Remote monitoring of manufacturing processes
- IoT-enabled HVAC systems
- Process control applications requiring PID tuning
- Educational demonstrations of [MQTT](#) in industrial contexts
- Prototyping industrial automation systems

12.24.2.1.9 Development Notes

- The application uses NetBurner's [MQTT](#) library with custom object extensions
- Includes comprehensive error handling and connection management
- Designed for continuous operation in industrial environments
- Supports both local debugging and remote operation
- Extensible architecture for adding additional control objects

12.24.3 MQTT Object I/O Example

12.24.3.1 MQTT Object I/O Application

12.24.3.1.1 Overview

This application demonstrates bidirectional [MQTT](#) communication using hardware I/O pins on embedded platforms. It creates a bridge between physical hardware inputs/outputs and [MQTT](#) messaging, allowing remote monitoring and control of GPIO pins through [MQTT](#) topics.

12.24.3.1.2 Features

- **Input Monitoring:** Monitors a physical input pin and publishes state changes to an [MQTT](#) topic
- **LED Control:** Subscribes to an [MQTT](#) topic to control a physical LED
- **Serial Console Interface:** Provides local control and monitoring through serial console
- **Multi-Platform Support:** Compatible with multiple NetBurner development boards

12.24.3.1.3 Hardware Configuration

The application uses different GPIO pins depending on the target platform:

Platform	Input Pin	LED Pin
SOMRT1061	Pin 22	LED1
NANO54415	Pin 50	LED1
MODM7AE70	P2[8]	LED1
MOD5441X	J2[45]	LED1
SBE70LC	P1[5]	LED1
SB800EX	No input pin	LED1

12.24.3.1.4 MQTT Topics

12.24.3.1.4.1 Published Topics

- **Device/Input1:** Publishes input pin state changes
 - Values: 0 (low/GND) or 1 (high/3.3V)
 - Published automatically when input pin state changes

12.24.3.1.4.2 Subscribed Topics

- **Device/LED1:** Controls LED state
 - Accepted values:
 - * `true` or any value `> 0`: Turns LED ON
 - * `false` or 0: Turns LED OFF

12.24.3.1.5 Operation

12.24.3.1.5.1 Startup Behavior

1. Initializes network stack and file system (if needed for certificates)
2. Waits for active network connection (5 second timeout)
3. Displays application name and NNDK revision
4. Sets up [MQTT](#) I/O objects for input monitoring and LED control
5. Outputs current input pin state (0 or 1) to serial console
6. Enters main loop for serial console interaction

12.24.3.1.5.2 Runtime Behavior

- **Input Pin Monitoring:** Automatically publishes to `Device/Input1` when the input pin state changes
- **LED Control via [MQTT](#):** LED state changes when messages are received on `Device/LED1` topic
- **Serial Console Control:** Pressing any key on the serial console toggles the LED state

12.24.3.1.6 Testing

12.24.3.1.6.1 Input Pin Testing

1. Connect the specified input pin to GND to read 0
2. Connect the specified input pin to 3.3V to read 1
3. Monitor the serial console or [MQTT](#) topic `Device/Input1` for state changes

12.24.3.1.6.2 LED Control Testing

1. **Via [MQTT](#):** Send messages to the `Device/LED1` topic:
 - Send `1`, `true`, or any positive number to turn LED on
 - Send `0` or `false` to turn LED off
2. **Via Serial Console:** Press any key to toggle the LED state

12.24.3.1.7 Configuration

The [MQTT](#) client configuration (broker URI, username, password, certificates, etc.) is handled through the NetBurner configuration system. Ensure your [MQTT](#) broker settings are properly configured before running the application.

12.24.3.1.8 Code Structure

12.24.3.1.8.1 Key Components

- [MQTT::ConfiguredClient](#) `MqttClient`: Main [MQTT](#) client instance
- [MQTT::mqtt_IO_Output](#) `myLED`: MQTT-controlled LED output object
- [MQTT::mqtt_IO_Input](#) `myInput`: MQTT-publishing input monitoring object

12.24.3.1.8.2 Optional Features

- Uncomment the `myLED.SetFlags(MQTT::eObj_Flag_PubOnWrite)` line to publish LED state changes to the [MQTT](#) topic when the LED is controlled locally

12.24.3.1.9 Dependencies

- NetBurner NNDK (NetBurner Network Development Kit)
- [MQTT](#) library support
- Network connectivity
- File system support (for [MQTT](#) certificates/keys)

12.24.3.1.10 Build Requirements

- Compatible NetBurner development board
- Properly configured [MQTT](#) broker
- Network connection (Ethernet or WiFi depending on platform)

12.24.3.1.11 Notes

- The SB800EX platform does not support input pin monitoring due to hardware limitations
- System diagnostics are enabled by default (consider removing for production deployments)
- The application waits up to 5 seconds for network connectivity before proceeding
- Certificate and key storage requires file system support on platforms that don't have built-in file system capabilities

12.24.4 MQTT Simple Object Pub/Sub Example

12.24.4.1 MQTT Simple Object Pub/Sub Application

12.24.4.1.1 Overview

This application demonstrates a simple MQTT publish/subscribe implementation using the NetBurner NNDK (NetBurner Network Development Kit). The application showcases automatic MQTT object synchronization, where a local variable can be both published to and updated from an MQTT topic.

12.24.4.1.2 Application Description

The MQTT Simple Object Pub/Sub Example is a console-based application that manages an integer variable (`myInt`) through MQTT messaging. The application performs the following key functions:

12.24.4.1.2.1 Core Functionality

- **Automatic Publishing:** The `myInt` variable is automatically published to the MQTT topic `testTopic/HelloWorld` whenever its value changes
- **Periodic Publishing:** Additionally publishes the current value every 3 minutes, regardless of whether it has changed
- **Automatic Subscription:** The same variable automatically subscribes to the MQTT topic, allowing remote updates
- **Interactive Control:** Users can increment the variable by typing any character in the serial console
- **Real-time Monitoring:** Continuously displays the current value of `myInt` on the console

12.24.4.1.2.2 Technical Implementation

- **MQTT Client:** Uses a configured MQTT client that gets its settings (broker URI, credentials, etc.) from the system configuration
- **MQTT Object:** Utilizes the `MQTT:mqtt_int` class for seamless integration between local variables and MQTT topics
- **Network Stack:** Initializes the full network stack and waits for active network connectivity
- **File system Support:** Includes conditional file system initialization for platforms that support it (used for storing MQTT certificates and keys)

12.24.4.1.3 Key Components

12.24.4.1.3.1 Variables and Objects

- `myInt`: An MQTT-enabled integer that automatically syncs with the MQTT topic
- `MqttClient`: Configured MQTT client instance
- `MQTT_TOPIC_NAME`: Defines the topic as "testTopic/HelloWorld"

12.24.4.1.3.2 Main Operations

1. **Network Initialization:** Establishes network connectivity and waits up to 5 seconds for an active connection
2. **MQTT Configuration:** Sets up automatic publishing every 3 minutes (180 seconds)
3. **Interactive Loop:** Continuously monitors for user input and displays current values

12.24.4.1.4 Usage

12.24.4.1.4.1 Running the Application

1. Ensure your NetBurner device is properly configured with network settings
2. Configure [MQTT](#) broker settings through the system configuration
3. Flash and run the application on your NetBurner device
4. Monitor the serial console for output

12.24.4.1.4.2 Interacting with the Application

- **Local Control:** Type any character in the serial console to increment `myInt`
- **Remote Control:** Send a numeric [MQTT](#) message to topic `testTopic/HelloWorld` to update `myInt`
- **Monitoring:** Watch the console output to see the current value of `myInt`

12.24.4.1.4.3 Expected Behavior

- Each keypress increments the variable and triggers an [MQTT](#) publish
- The current value is displayed on the console after each update
- Every 3 minutes, the current value is published automatically
- Remote [MQTT](#) messages to the topic will update the local variable

12.24.4.1.5 Configuration Requirements

Before running this application, ensure the following are configured:

- **Network Settings:** IP address, subnet mask, gateway, DNS
- **MQTT Broker:** Broker URI/IP address and port
- **Authentication:** Username and password (if required by broker)
- **Security:** TLS certificates and keys (if using secure connection)

12.24.4.1.6 Development Notes

- The application includes system diagnostics that should be removed for production code
- Filesystem initialization is conditional based on platform capabilities
- The code uses NetBurner's RTOS and network stack APIs
- [MQTT](#) objects provide automatic serialization and deserialization

12.24.4.1.7 Platform Requirements

- NetBurner NNDK-compatible hardware
- Network connectivity (Ethernet or Wi-Fi)
- Access to an [MQTT](#) broker
- Serial console for interaction and monitoring

This example serves as a foundation for more complex [MQTT](#) applications and demonstrates the ease of integrating [MQTT](#) messaging into NetBurner applications using the provided [MQTT](#) object classes.

12.24.5 MQTT SubHandler GPIO Example

This example demonstrates a custom subscription handler with the easy MqttObj system, for the case where a developer wants to easily use custom logic for subscribing to messages without implementing low-level things. Once connected, a dynamic topic name will be output to stdout. The user can send an MQTT message of 1 or 0 to that topic to turn on or off the LED pin.

Any numbered PinIO pin can be accessed in this way. For platforms that access pins like P1[int], a slash can separate the header and pin, like: pins/1/1

12.24.6 Advanced MQTT Periodic Example

12.24.6.1 MQTT Periodic Publish Example

12.24.6.1.1 Overview

The MQTT Periodic Publish Example is a comprehensive demonstration of implementing the NetBurner MQTT client at a deep level, designed for developers who want to customize every aspect of the MQTT interaction. This example showcases advanced MQTT publishing techniques using periodic timers, semaphores, and quality of service (QoS) configurations.

12.24.6.1.2 Application Features

12.24.6.1.2.1 Core Functionality

- **Dual Publishing Modes:** The application operates in two distinct modes:
 - **Echo Mode (PublishCounter):** Sends sequential binary data chunks every 10 seconds
 - **Sine Wave Mode:** Publishes calculated sine wave values at different frequencies

12.24.6.1.2.2 MQTT Topics

The application publishes to three primary MQTT topics:

- testTopic/example/echo - Binary counter data in echo mode
- testTopic/example/sine_x - Sine wave values (sin(x))
- testTopic/example/sine_3x - Scaled sine wave values (3*sin(x))

12.24.6.1.2.3 Advanced MQTT Features

- **Quality of Service (QoS):** Implements "At Least Once" delivery guarantee
- **Publish Acknowledgment Callbacks:** Custom handlers for publication confirmations
- **Connection State Management:** Automatic handling of connection establishment
- **Priority-based Message Queuing:** Configurable message priorities for publishing
- **Semaphore-based Flow Control:** Prevents message queue overflow

12.24.6.1.3 Technical Implementation

12.24.6.1.3.1 Key Components

MQTT Client Configuration

- Uses MQTT::ConfiguredClient with broker settings from the configuration system
- Implements connection notifications and automatic reconnection handling
- Supports secure connections with certificate/key storage via filesystem

Interval Timing System

- **IntervalTimer:** Manages periodic publishing operations
- **IntervalAction:** Defines callback actions for different publishing tasks
- **OS Semaphores:** Provides thread-safe publishing coordination

Publishing Mechanisms

1. Sine Wave Publishers:

- Calculate mathematical sine values with configurable step size
- Publish floating-point values as formatted strings
- Implement separate callbacks for acknowledgment handling

2. Counter Publisher:

- Generates sequential binary data (64 bytes per message)
- Uses union structure for efficient byte/word access
- Publishes at lower priority with fire-and-forget semantics

12.24.6.1.3.2 State Management

The application operates in three distinct run states:

- **State 0:** Connection establishment and timer initialization
- **State 1:** Active publishing with user interaction support
- **State 2:** Graceful shutdown and cleanup

12.24.6.1.4 Application Behavior

12.24.6.1.4.1 Startup Sequence

1. **Initialization:** Network stack and filesystem setup
2. **Connection Waiting:** Displays "NotConnected" until [MQTT](#) broker connection established
3. **Timer Start:** Shows "Starting IntervalSem: {timer}" when connected and ready
4. **Active Publishing:** Begins sending data according to configured intervals

12.24.6.1.4.2 Runtime Operation

- **Echo Mode:** Sends 64-byte binary counter data every 5 seconds to echo topic
- **Sine Mode:** Publishes calculated sine values with different frequencies
 - `sine_x` published every 4th cycle
 - `sine_3x` published every cycle
- **Interactive Control:** Press any key to toggle between modes or stop operation

12.24.6.1.4.3 Message Flow Control

- Uses semaphores to prevent overwhelming the [MQTT](#) broker
- Implements timeout mechanisms for reliable message delivery
- Provides graceful fallback when broker becomes unavailable

12.24.6.1.5 Configuration Requirements

12.24.6.1.5.1 Network Setup

- Requires active network connection before MQTT operations begin
- Waits up to 5 seconds for network establishment during startup

12.24.6.1.5.2 MQTT Broker Settings

- Broker URI, username, password, and other client configurations are managed through the NetBurner configuration system
- Supports secure connections with TLS/SSL certificates stored in filesystem

12.24.6.1.5.3 Development Environment

- Designed for NetBurner embedded systems
- Requires NNDK (NetBurner Network Development Kit)
- File system support recommended for certificate management

12.24.6.1.6 Known Issues

- Subscription and message reception functionality for the echo topic is not currently implemented (marked as TODO)
- The application currently focuses on publishing; subscriber callbacks need to be added for complete bidirectional communication

12.24.6.1.7 Usage Notes

This example is ideal for developers who need:

- Fine-grained control over MQTT message publishing
- Understanding of advanced QoS implementation
- Integration of periodic timers with MQTT operations
- Custom acknowledgment handling and flow control
- Educational reference for NetBurner MQTT client implementation

12.24.7 Advanced MQTT Pub/Sub Example

12.24.7.1 MQTT Advanced Pub/Sub Example

12.24.7.1.1 Overview

This NetBurner application demonstrates advanced MQTT publish/subscribe functionality with additional features including remote configuration management and GPIO pin control over MQTT. The application serves as a comprehensive example of integrating MQTT communication with embedded system control capabilities.

12.24.7.1.2 Features

12.24.7.1.2.1 Core MQTT Functionality

- **Automatic Publishing:** Publishes "Hello World" messages to `testTopic/HelloWorld` every 10 seconds
- **Message Subscription:** Subscribes to the same topic and echoes received messages to the serial console
- **Connection Management:** Handles MQTT connection establishment and recovery automatically
- **Rate Limiting:** Implements publish rate limiting (20 messages per 5 seconds, burst of 4)

12.24.7.1.2.2 Advanced Features

- **Remote Configuration:** Exposes NetBurner Config system via [MQTT](#) topic `device/nburnxxxxxx/config`
- **GPIO Control:** Allows remote control of device pins through [MQTT](#) topics `device/nburnxxxxxx/pins/+`
- **MAC-Based Topics:** Uses device MAC address for unique topic identification
- **Multi-Platform Support:** Supports various NetBurner platforms with different pin configurations

12.24.7.1.3 MQTT Topics

12.24.7.1.3.1 Published Topics

- `testTopic/HelloWorld` - Publishes "Hello World" message every 10 seconds

12.24.7.1.3.2 Subscribed Topics

- `testTopic/HelloWorld` - Echoes received messages to serial console
- `device/nburn_{MAC}/config` - Configuration management interface
- `device/nburn_{MAC}/pins/+` - GPIO pin control interface

Note

`{MAC}` is replaced with the last 3 octets of the device's MAC address (e.g., `device/nburn123456/config`)

12.24.7.1.4 GPIO Control

12.24.7.1.4.1 Pin Control Format

The application accepts pin control messages in the following formats:

For platforms with direct pin access (NANO54415, SB800EX, SOMRT1061, SBE70LC):**

- Topic: `device/nburn_{MAC}/pins/{pin_number}`
 - Examples: `device/nburn123456/pins/1`, `device/nburn123456/pins/15`
- For platforms with header/pin structure (MOD5441X):**
- Topic: `device/nburn_{MAC}/pins/{header}/{pin}`
 - Examples: `device/nburn123456/pins/1/5`, `device/nburn123456/pins/2/3`

12.24.7.1.4.2 Control Values

Pins can be controlled using various message formats:

- `1` or `true` - Set pin HIGH
- `0` or `false` - Set pin LOW
- Any numeric value - Set pin to that boolean value (0 = LOW, non-zero = HIGH)

12.24.7.1.5 Application Flow

1. Initialization

- Initialize file system (if not present)
- Initialize network stack
- Wait for active network connection
- Configure [MQTT](#) client with rate limiting

2. Connection Phase

- Establish [MQTT](#) connection using configured broker settings
- Wait for connection establishment before proceeding

3. Operation Phase

- Subscribe to echo topic (`testTopic/HelloWorld`)
- Enter main loop:
 - Publish "Hello World" message every 10 seconds
 - Process incoming [MQTT](#) messages
 - Handle pin control requests
 - Monitor for user input to change states

4. Message Handling

- Echo messages: Forward received messages to serial console
- Pin control: Parse topic and message to control GPIO pins
- Configuration: Handle remote configuration changes

12.24.7.1.6 Configuration

The application uses the NetBurner Config system for [MQTT](#) broker configuration:

- Broker URI
- Username/Password authentication
- SSL/TLS certificates (stored in file system)
- Connection parameters

Configuration can be modified remotely via the [MQTT](#) configuration topic.

12.24.7.1.7 Build Requirements

- NetBurner NNDK (NetBurner Network Development Kit)
- Supported NetBurner hardware platform
- [MQTT](#) broker for testing

12.24.7.1.8 Usage Instructions

1. Setup

- Configure [MQTT](#) broker settings using NetBurner Config system
- Flash the application to your NetBurner device
- Ensure network connectivity

2. Testing

- Monitor serial output for connection status and published messages
- Use [MQTT](#) client to subscribe to `testTopic/HelloWorld` to see published messages
- Publish messages to `testTopic/HelloWorld` to test echo functionality
- Control GPIO pins by publishing to appropriate pin control topics

3. Interactive Control

- Press any key during operation to change application states
- Use serial console to monitor application status and debug information

12.24.7.1.9 Platform-Specific Notes

12.24.7.1.9.1 Supported Platforms

- **NANO54415:** Direct pin access via `Pins[]` array
- **SB800EX:** Direct pin access via `Pins[]` array
- **SOMRT1061:** Direct pin access via `Pins[]` array
- **SBE70LC:** Direct pin access via `Pins[]` array
- **MOD5441X:** Header-based pin access (J1/J2 headers)

12.24.7.1.9.2 Pin Control Differences

- **Direct access platforms:** Use single pin number (e.g., `/pins/5`)
- **Header-based platforms:** Use header/pin format (e.g., `/pins/1/5` for J1 pin 5)

12.24.7.1.10 Error Handling

- **Connection Loss:** Automatically attempts reconnection
- **Subscription Failures:** Reports error codes and reason strings
- **Invalid Pin Messages:** Silently ignores malformed pin control messages
- **Rate Limiting:** Prevents message flooding with configurable limits

12.24.7.1.11 Debug Information

The application provides extensive debug output including:

- Connection status updates
- Subscription confirmation
- Message reception notifications
- Pin control operations
- Error conditions and recovery attempts

12.24.7.1.12 Security Considerations

- Uses NetBurner's built-in [MQTT](#) security features
- Supports SSL/TLS encryption when configured
- MAC-based topic naming provides device-specific addressing
- Configuration changes logged for audit trail

12.24.8 Advanced MQTT Echo Example

12.24.8.1 MQTT Echo Application

12.24.8.1.1 Overview

This NetBurner [MQTT](#) Echo application demonstrates advanced [MQTT](#) client functionality by subscribing to a specific topic and echoing received messages to the serial console. The application uses low-level [MQTT](#) protocol handling to provide detailed message processing capabilities.

12.24.8.1.2 Features

- **MQTT Topic Subscription:** Automatically subscribes to `testTopic/HelloWorld`
- **Message Echo:** Outputs all received messages to the serial console
- **Low-Level Protocol Handling:** Uses file descriptors and `readall/writeall` functions for direct message processing
- **Connection Management:** Handles MQTT connection establishment and monitoring
- **Property Handling:** Processes MQTT message properties (metadata) appropriately
- **Error Handling:** Manages subscription termination and connection failures

12.24.8.1.3 Architecture

12.24.8.1.3.1 Key Components

- **ConfiguredClient:** MQTT client configured through the NetBurner configuration system
- **TopicHandler:** Low-level message handler that processes incoming MQTT Publish messages
- **EchoSubscriber:** Callback function that handles message reception and console output

12.24.8.1.3.2 Message Processing Flow

1. Application establishes connection to MQTT broker
2. Subscribes to the configured topic (`testTopic/HelloWorld`)
3. Receives MQTT Publish messages through file descriptor interface
4. Processes message properties (skips them in this implementation)
5. Reads message payload and outputs to serial console
6. Handles connection status changes and subscription termination

12.24.8.1.4 Configuration

The MQTT client configuration (broker URI, username, password, certificates) is managed through the NetBurner configuration system rather than hardcoded values.

12.24.8.1.4.1 Default Settings

```
#define MQTT_TOPIC_NAME      "testTopic/HelloWorld"
#define MQTT_PUBLISH_MSG    "Hello World"
```

12.24.8.1.5 Technical Details

12.24.8.1.5.1 File Descriptor Interface

This application demonstrates the low-level MQTT interface using:

- `readall()` function for reading message data
- `writeall()` function for console output
- Direct file descriptor manipulation for message processing

12.24.8.1.5.2 Message Structure Handling

The application properly handles MQTT Publish message structure:

- **Properties Section:** MQTT v5.0 properties for metadata (skipped in this example)
- **Payload Section:** Actual message content (echoed to console)

12.24.8.1.5.3 Connection Management

- Waits for active network connection (5-second timeout)
- Monitors MQTT connection status continuously
- Automatically attempts subscription when connection is established
- Handles connection failures gracefully

12.24.8.1.6 Usage

1. **Network Setup:** Ensure the device is connected to a network
2. **MQTT Configuration:** Configure MQTT broker settings through NetBurner config
3. **Deploy Application:** Flash the application to the NetBurner device
4. **Monitor Output:** Watch the serial console for received messages
5. **Test Messaging:** Publish messages to `testTopic/HelloWorld` from any MQTT client

12.24.8.1.7 Console Output

The application provides detailed console output including:

- Connection status updates
- Subscription confirmation with handler ID
- Received message topic information
- Message payload content
- Subscription termination notifications

12.24.8.1.8 Error Handling

- **Subscription Termination:** Detects when subscriptions end and reports reason codes
- **Invalid Message Types:** Filters out non-Publish messages
- **Connection Failures:** Continuously monitors and reports connection status
- **File System Support:** Conditionally initializes filesystem for certificate storage

12.24.8.1.9 Dependencies

- NetBurner NNDK (NetBurner Network Development Kit)
- MQTT library support
- Network stack initialization
- Optional: Filesystem support for certificates

12.24.8.1.10 Development Notes

- System diagnostics are enabled for development (should be disabled for production)
- Uses 10-second polling interval for connection monitoring
- 4-second timeout for subscription operations
- Supports both platforms with and without filesystem capabilities

12.24.8.1.11 Example Use Cases

- IoT device message monitoring
- [MQTT](#) broker testing and validation
- Network communication debugging
- Real-time message processing demonstrations
- Integration testing for MQTT-based systems

12.25 Multicast

12.25.1 Multicast Example Application

12.25.1.1 Overview

This example program demonstrates how to add a NetBurner device to a multicast group for sending and receiving UDP packets. The application allows you to specify a multicast IP address and port number, then send data by typing it in MTTTY or view received data from other devices in the same multicast group.

Note

This example only supports IPv4 multicast communication.

12.25.1.2 Features

- Join IPv4 multicast groups
- Send UDP packets to multicast group
- Receive and display UDP packets from multicast group
- Interactive console interface for user input
- Concurrent packet listening and sending

12.25.1.3 Application Components

12.25.1.3.1 main.cpp

The main NetBurner application written in C++ that:

- Prompts user for multicast group port number
- Prompts user for multicast IP address (administratively-scoped local IPv4 range: 239.0.0.0 to 239.255.255.255)
- Creates a background task to listen for incoming multicast packets
- Allows interactive sending of text messages to the multicast group
- Handles graceful shutdown and unregistration from multicast group

12.25.1.3.2 MultiRx.py

A Python utility script for testing multicast reception:

- Listens on multicast group 239.1.2.3 port 35123
- Receives and displays UDP packets from the multicast group
- Useful for testing the NetBurner application from a PC

12.25.1.4 How It Works

1. **Initialization:** The application starts by enabling system diagnostics and prompting for configuration
2. **Configuration:** User enters the port number and multicast IP address
3. **Task Creation:** A background task (MultiReaderMain) is created to handle incoming packets
4. **Packet Reception:** The background task registers with the multicast group and continuously listens for packets
5. **Packet Sending:** The main loop allows user to type messages that are sent as UDP packets to the multicast group
6. **Shutdown:** Empty input triggers unregistration and shutdown

12.25.1.5 Usage Instructions

12.25.1.5.1 NetBurner Device Setup

1. Flash the application to your NetBurner device
2. Connect to the device via MTTY
3. When prompted, enter the multicast port number (e.g., 35123)
4. Enter a valid multicast IP address in the range 239.0.0.0 to 239.255.255.255
5. The device will join the multicast group and display confirmation

12.25.1.5.2 Sending Messages

1. Type any text message and press Enter to send it to the multicast group
2. The message will be broadcast to all devices in the same multicast group
3. Enter an empty string (just press Enter) to shutdown the listener

12.25.1.5.3 Receiving Messages

- The application automatically displays received packets with:
 - Packet size in bytes
 - Source IP address
 - Packet contents

12.25.1.5.4 Testing with Python Script

1. Run the MultiRx.py script on a PC connected to the same network
2. The script will listen for packets on multicast group 239.1.2.3 port 35123
3. Send messages from the NetBurner device to see them received by the Python script

12.25.1.6 Technical Details

12.25.1.6.1 Key Functions

- `RegisterMulticastFifo()`: Registers to receive multicast packets
- `UnregisterMulticastFifo()`: Unregisters from multicast group
- `UDPPacket`: Class for creating and parsing UDP packets
- `OSTaskCreatewName()`: Creates background task for packet reception

12.25.1.6.2 Multicast IP Address Ranges

The application uses administratively-scoped local IPv4 multicast addresses:

- Range: 239.0.0.0 to 239.255.255.255
- These addresses are reserved for organizational use and don't route across the internet

12.25.1.6.3 Threading Model

- Main thread handles user input and packet sending
- Background thread (MultiReaderMain) handles packet reception
- Uses OS_FIFO for inter-thread communication

12.25.1.7 Important Notes

- Only IPv4 multicast is supported
- The application requires a reset to restart after shutdown
- System diagnostics are enabled (consider removing for production)
- The hardcoded IP address 239.1.2.3 overrides user input in the current code

12.25.1.8 Troubleshooting

1. **No packets received:** Verify all devices are on the same network and using the same multicast address/port
2. **Application hangs:** Check network connectivity and firewall settings
3. **Build errors:** Ensure all NetBurner SDK libraries are properly linked

12.25.1.9 Dependencies

- NetBurner SDK
- Standard C++ libraries
- NetBurner multicast support library

12.26 Multihome

12.26.1 Multihome Application

A NetBurner TCP/IP Stack example demonstrating IPv4 multihoming capabilities.

12.26.1.1 Overview

This application demonstrates how to implement multiple IP addresses on a single NetBurner device using the Multihome functionality. The device will respond to requests from multiple network interfaces simultaneously:

- **DHCP Interface:** Obtains a dynamic IP address from a **DHCP** server
- **AutoIP Interface:** Uses automatic IP address assignment
- **Static Interface:** Uses a fixed IP address (10.1.1.240)

12.26.1.2 Features

- Multiple network interface support (up to 10 interfaces)
- Web-based IP address display and diagnostics
- Real-time IP address monitoring via serial debug output
- Responds to HTTP requests on any configured IP address

12.26.1.3 Prerequisites

12.26.1.3.1 Required System Configuration

Before building and running this application, you must enable multihome support:

The application example includes a `predef-overload.h` file that automatically enables:

- `MULTIHOME` functionality
- Support for up to 10 network interfaces (`NUM_MULTI_INTERFACES`)

The file utilizes the NetBurner **overload** feature by locating the file at: `\MultiHome\overload\nbrtos\include\predef-overload.h`

This way NetBurner system file `predef.h` does not need to be modified. See the **OverloadDirectory** example or the Overload section of this manual for more information.

Alternatively, you can modify the NetBurner `predef.h` system file as described below:

1. Navigate to `\nburn\nbrtos\include\predef.h`
2. Uncomment the `MULTIHOME` definition
3. Rebuild the system files

12.26.1.4 Application Behavior

12.26.1.4.1 Network Interface Setup

1. **Primary Interface:** Attempts `DHCP` address assignment
2. **Secondary Interface:** Uses AutoIP for automatic address assignment
3. **Tertiary Interface:** Configures static IP address (10.1.1.240) with subnet mask 255.255.255.0

12.26.1.4.2 Web Interface

The application starts an HTTP server that provides IP address information. Access the web interface using any of the configured IP addresses to view:

- Source IP address of the requesting client
- Destination IP address (which interface received the request)
- All configured IP addresses on the device

12.26.1.4.3 Debug Output

The application continuously outputs IP address information to the debug serial port every 5 seconds, showing:

- `DHCP` assigned IP address
- AutoIP assigned address
- Static multihome IP address

12.26.1.5 Code Structure

12.26.1.5.1 Key Files

- `main.cpp`: Main application logic and web server implementation
- `ReadMe.txt`: Original documentation (Doxygen format)
- `predef-overload.h`: Compiler definitions for multihome support

12.26.1.5.2 Key Functions

- `UserMain()`: Main application entry point
- `WebDisplayIpSettings()`: Web page handler for IP address display
- `AddInterface()`: Adds the static multihome interface

12.26.1.5.3 Important Variables

- `DhcpIb`: Interface block for DHCP interface
- `StaticIb`: Interface block for AutoIP interface
- `MultiHome1Ib`: Interface block for static multihome interface

12.26.1.6 Usage

1. Ensure multihome support is enabled in system configuration
2. Build and deploy the application to your NetBurner device
3. Connect the device to your network
4. Monitor the debug serial output for IP address assignments
5. Access the web interface using any of the assigned IP addresses

12.26.1.7 Network Configuration

The static interface is configured with:

- **IP Address**: 10.1.1.240
- **Subnet Mask**: 255.255.255.0
- **Gateway**: Not configured (`IPADDR4::NullIP()`)

Ensure your network topology can accommodate this address range if you plan to access the static interface.

12.26.1.8 Error Handling

The application includes a compile-time check to ensure multihome support is enabled. If `MULTIHOME` is not defined, compilation will fail with an informative error message directing you to enable the feature in the system configuration.

12.26.1.9 System Requirements

- NetBurner device with TCP/IP stack
- NBRTOS operating system
- Multihome capability enabled in system build

12.27 NB Approve Shutdown

12.27.1 NBApproveShutdown Application

12.27.1.1 Overview

The NBApproveShutdown application demonstrates how to implement safe system shutdown handling for NetBurner embedded systems. This application provides a callback mechanism that allows developers to put their system in a safe state before application updates, configuration updates, or custom reboot events occur.

12.27.1.2 Purpose

The primary purpose of this application is to showcase the `NBApproveShutdown()` callback function, which enables applications to:

- Close active TCP sockets before shutdown
- Ensure Flash and file system write operations are complete
- Put critical peripherals in a safe state
- Control whether a shutdown request should be approved or denied

12.27.1.3 System Requirements

- NetBurner Hardware
- NetBurner Network Development Kit (NNDK)
- Serial terminal for interaction

12.27.1.4 Application Features

12.27.1.4.1 Shutdown Callback Function

The application implements a custom `NBApproveShutdown()` function that overrides the default system implementation. This function handles different shutdown reasons:

- **Code Update** (`SHUTDOWN_CODEUPDATE`): Triggered when a firmware update is requested
- **Configuration Reboot** (`SHUTDOWN_CONFIGURE_REBOOT`): Triggered when configuration changes require a reboot
- **Custom Reboot** (`SHUTDOWN_CUSTOM_REBOOT`): Application-defined reboot scenarios

12.27.1.4.2 Custom Reboot Reason

Custom reboot reason. System reasons start at 1, so pick something much larger

```
#define SHUTDOWN_CUSTOM_REBOOT 100

if ( NBApproveShutdown (SHUTDOWN_CUSTOM_REBOOT) )
{
    OSTimeDly (TICKS_PER_SECOND * 5);
    ForceReboot ();
}
```

12.27.1.4.3 Interactive Serial Menu

The application provides a simple serial port menu interface with the following options:

1. **Enable Shutdown** - Allows shutdown requests to proceed
2. **Disable Shutdown** - Blocks shutdown requests for testing
3. **Custom Reboot Request** - Initiates a custom reboot sequence

12.27.1.4.4 Web Server Integration

The application includes a basic web server that starts on port 80, allowing for potential web-based interactions and monitoring.

12.27.1.5 How It Works

12.27.1.5.1 Shutdown Approval Process

1. When a shutdown event occurs, the system calls `NBApproveShutdown()` with a reason code
2. The application can examine the reason and current system state
3. Based on application logic, the function returns:
 - `true` to approve the shutdown
 - `false` to abort the shutdown

12.27.1.5.2 Global Control Flag

The application uses a global boolean flag `bAllowShutdown` to demonstrate shutdown control:

- When `true`: All shutdown requests are approved
- When `false`: All shutdown requests are denied (except unknown reasons)

12.27.1.6 Usage Instructions

12.27.1.6.1 Building and Running

1. Compile the application using the NetBurner development tools
2. Flash the application to your NetBurner device
3. Connect a serial terminal to the device
4. Monitor the serial output for menu options and status messages

12.27.1.6.2 Testing Shutdown Behavior

1. **Enable/Disable Testing:** Use menu options 1 and 2 to toggle shutdown approval
2. **Custom Reboot Testing:** Use menu option 3 to trigger a custom reboot sequence
3. **Update Testing:** Attempt a firmware update to test the code update shutdown path

12.27.1.6.3 Serial Commands

- Press 1 to enable shutdown approval
- Press 2 to disable shutdown approval
- Press 3 to request a custom reboot
- Press any other key to display the menu

12.27.1.7 Code Structure

12.27.1.7.1 Key Components

- `NBApproveShutdown()`: Main callback function that processes shutdown requests
- `displayMenu()`: Shows available menu options and current status
- `processCommand()`: Handles serial input and executes commands
- `UserMain()`: Application entry point and main loop

12.27.1.7.2 Important Constants

```
#define SHUTDOWN_CODEUPDATE      (1)    // Code update requested
#define SHUTDOWN_CONFIGURE_REBOOT (2)    // Configuration reboot requested
#define SHUTDOWN_CUSTOM_REBOOT   (100)   // Custom application reboot
```

12.27.1.8 Implementation Notes

12.27.1.8.1 Safety Considerations

- The callback function includes a 1-second delay to ensure serial output completes before reboot
- Unknown shutdown reasons are automatically rejected for safety
- The weak reference mechanism allows easy override of default system behavior

12.27.1.8.2 Customization Points

Developers can customize this application by:

- Adding specific hardware shutdown sequences in the callback function
- Implementing additional shutdown reason codes
- Adding more sophisticated approval logic based on system state
- Integrating with web-based monitoring and control

12.27.1.9 Example Output

```
Web Application: NB Approve Shutdown
NNDK Revision: [Version Number]
```

```
----- Main Menu -----
1 - Enable  Shutdown
2 - Disable Shutdown
3 - Custom Reboot Request
Current bAllowShutdown = true
```

12.27.1.10 Development Guidelines

12.27.1.10.1 Best Practices

1. Always include appropriate delays before reboot to ensure data integrity
2. Test both approved and denied shutdown scenarios
3. Handle all known shutdown reasons explicitly
4. Consider system state when making approval decisions
5. Log shutdown events for debugging and monitoring

12.27.1.10.2 Integration Tips

- Use this pattern in production applications to ensure graceful shutdowns
- Combine with watchdog timers for additional safety
- Consider network notification of shutdown events
- Implement persistent logging of shutdown requests and approvals

12.28 NSString Class

This example program demonstrates the functionality [NSString](#) class.

12.29 Network Time Server (NTP) Client

12.29.1 NTP Client Example Application

A NetBurner embedded system application that demonstrates Network Time Protocol (NTP) client functionality, providing synchronized time display with timezone support.

Warning

For this example to function, your NetBurner device must have access to the Internet with a working configuration for the IP address, mask and gateway.

12.29.1.1 Overview

This application connects to an NTP server to synchronize the system clock and displays time information in multiple formats and time zones through the debug serial port. It serves as a practical example of network time synchronization for embedded NetBurner devices.

12.29.1.2 Features

- **NTP Time Synchronization:** Automatically connects to NTP servers for accurate time
- **Multiple Time Zone Support:** Displays time in UTC, local time, and major US time zones
- **Interactive Time Zone Selection:** Choose between US/Canada or International time zones
- **Real-time Display:** Continuously updates and displays current time information
- **Network Diagnostics:** Shows IP configuration and network status

12.29.1.3 Prerequisites

- NetBurner device with network connectivity
- Proper IP configuration (IP address, subnet mask, gateway)
- Internet access to reach NTP servers
- MTTTY or similar terminal program for monitoring output

12.29.1.4 Time Zone Features

12.29.1.4.1 Supported US Time Zones

The application includes predefined format strings for major US time zones:

- Eastern Time (EST/EDT)
- Central Time (CST/CDT)
- Mountain Time (MST/MDT)
- Pacific Time (PST/PDT)

12.29.1.4.2 Time Zone Selection

Users can choose between:

- **US/Canada Time Zones:** Filtered list of North American zones
- **International Time Zones:** Complete global time zone database

Note

Good example of using the global time zone database timezones.h

12.29.1.5 Application Flow

1. **Initialization:** System starts and enables network diagnostics
2. **Network Setup:** Waits for active network connection (10-second timeout)
3. **NTP Synchronization:** Connects to NTP server and waits for valid time
4. **Time Zone Setup:** Prompts user to select preferred time zone
5. **Display Loop:** Continuously shows time in multiple formats

12.29.1.6 Time Display Format

The application displays time information in the following format:

Time[ZONE] = MM/DD/YYYY day: DDD (DAY) HH:MM:SS

Where:

- ZONE: Time zone abbreviation
- MM/DD/YYYY: Month/Day/Year
- DDD: Day of year (1-365/366)
- DAY: Day of week abbreviation
- HH:MM:SS: Hour:Minute:Second in 24-hour format

12.29.1.7 Output Examples

12.29.1.7.1 Local Time Display

Local time based on SetTimeZone() function:
Time[PST] = 7/21/2025 day: 202 (MON) 15:30:45

12.29.1.7.2 UTC Time Display

UTC time =
Time[UTC] = 7/21/2025 day: 202 (MON) 23:30:45

12.29.1.7.3 Multiple Time Zone Display

Displaying US time zones with local format strings:
Time[EST] = 7/21/2025 day: 202 (MON) 18:30:45
Time[CST] = 7/21/2025 day: 202 (MON) 17:30:45
Time[MST] = 7/21/2025 day: 202 (MON) 16:30:45
Time[PST] = 7/21/2025 day: 202 (MON) 15:30:45

12.29.1.8 Key Functions

12.29.1.8.1 SetTimeZone()

Interactive function that allows users to select their preferred time zone from either US/Canada or International options.

12.29.1.8.2 PrintTimeStruct(struct tm &bt)

Formats and displays time information in a standardized format to the debug serial port.

12.29.1.8.3 UserMain(void *pd)

Main application entry point that handles initialization, NTP synchronization, and the main display loop.

12.29.1.9 Technical Details

12.29.1.9.1 Dependencies

- `init.h`: System initialization
- `time.h`: Time manipulation functions
- `ipshow.h`: IP address display utilities
- `timezones.h`: Time zone database and utilities
- `nbttime.h`: NetBurner time functions

12.29.1.9.2 NTP Client

The application uses the `NtpClientServlet` class for NTP operations:

- Waits up to 20 seconds for valid NTP response
- Provides retry mechanism with status updates
- Tracks time since last NTP update

12.29.1.10 Usage Instructions

1. **Hardware Setup**: Connect NetBurner device to network
2. **Network Configuration**: Ensure proper IP, mask, and gateway settings
3. **Terminal Connection**: Use MTTTY to monitor serial output
4. **Run Application**: Deploy and execute the application
5. **Time Zone Selection**: Follow prompts to select desired time zone
6. **Monitor Output**: View continuous time updates
7. **Manual Refresh**: Press any key to update display on demand

12.29.1.11 Status Information

The application provides:

- Network connectivity status
- NTP synchronization status with retry notifications
- Time since last NTP update
- IP address configuration display

12.29.1.12 Troubleshooting

- **No Network**: Ensure device has proper network configuration
- **NTP Timeout**: Check internet connectivity and firewall settings
- **Invalid Time**: Wait for NTP synchronization to complete
- **Display Issues**: Verify MTTTY terminal settings and baud rate

12.29.1.13 Development Notes

- Remove `EnableSystemDiagnostics()` for production deployment
- Time zone database extensible through `TZRecords` structure
- IPv6 support available when compiled with `IPV6` flag

12.30 NTP & Real-Time Clock

NTP-SYSTEM-RTC DEMO APPLICATION

Note

HARDWARE SETUP: Tested with MOD54415 on a MOD-DEV-70 hardware rev 1.93.

12.30.1 NTP-System-RTC Demo Application

A NetBurner embedded system application that demonstrates synchronization between Network Time Protocol (NTP) servers, system time, and Real-Time Clock (RTC) hardware components.

12.30.1.1 Overview

This application provides a comprehensive example of time management in embedded systems, showcasing the interaction between network-based time synchronization, system time management, and hardware RTC persistence. The demo is particularly useful for applications requiring accurate timekeeping that persists across power cycles.

12.30.1.2 Hardware Requirements

- **Primary Platform:** MOD54415 module
- **Development Board:** MOD-DEV-70 (hardware revision 1.93 or compatible)
- **RTC Component:** One of the following supported RTC chips:
 - Intersil X1226
 - NXP PCF8563

The RTC component must be present either on the module itself or on the development board to utilize the RTC-related features.

12.30.1.3 Features

12.30.1.3.1 Core Functionality

1. NTP Time Synchronization

- Retrieves current UTC time from NTP server pool (pool.ntp.org)
- Updates system time with network-sourced accurate time
- Requires DNS resolution capability

2. System-RTC Synchronization

- Saves current system time to hardware RTC
- Restores system time from RTC after power cycles
- Maintains time accuracy during power interruptions

3. Time Zone Support

- Configurable local time zone settings
- Pacific Time Zone (PST/PDT) preconfigured as example
- Automatic daylight saving time transitions
- Customizable via `tzsetchar()` function

4. Time Persistence

- RTC continues running on super-capacitor power during outages
- System time restoration from RTC after reset/power loss
- Software reset capability for testing RTC persistence

5. Time Display and Verification

- Real-time display of UTC and local times
- RTC time information display
- Synchronization status verification
- Time zone configuration display

12.30.1.4 Application Menu

The application provides an interactive menu with the following options:

- **[1]** Sync system time with NTP server (requires DNS)
- **[2]** Synchronize RTC from current system time
- **[3]** Synchronize system time from RTC
- **[4]** Verify synchronization between system and RTC
- **[5]** Display current system and RTC times
- **[6]** Re-display menu options
- **[0]** Software reset device (tests RTC persistence)

12.30.1.5 Technical Details

12.30.1.5.1 Time Storage Format

- **System Time:** Managed by NetBurner OS, stored as `time_t`
- **RTC Time:** Hardware-based, stores UTC time (recommended)
- **Local Time:** Calculated from UTC using configured time zone

12.30.1.5.2 Synchronization Logic

The application implements bidirectional synchronization:

- **NTP > System:** Network time updates local system clock
- **System > RTC:** Persistent storage of accurate time
- **RTC > System:** Time recovery after power cycles

12.30.1.5.3 Error Handling

- Network connectivity verification for NTP synchronization
- RTC hardware communication error detection
- DNS resolution failure handling
- Synchronization status reporting

12.30.1.6 Network Requirements

12.30.1.6.1 For NTP Functionality

- Active network connection (Ethernet recommended)
- DNS server configuration (required for `pool.ntp.org` resolution)
- Outbound NTP traffic (port 123) capability

12.30.1.6.2 Static IP Considerations

When using static IP configuration, manually configure DNS server addresses to enable NTP server name resolution.

12.30.1.7 Time Zone Configuration

The application uses Pacific Time as a demonstration. To configure for different time zones:

```
// Example for Eastern Time
char tzInfo[] = "EST5EDT4,M3.2.0/02:00:00,M11.1.0/02:00:00";
tzsetchar(tzInfo);
```

Refer to NetBurner Runtime Libraries documentation (Chapter 14 - NBTime Library) for detailed time zone configuration syntax.

12.30.1.8 Build and Deployment

12.30.1.8.1 Prerequisites

- NetBurner development environment
- Compatible NetBurner runtime libraries
- Hardware with supported RTC component

12.30.1.8.2 Key Libraries

- `nbttime.h` - NetBurner time management
- `rtc.h` - Real-time clock interface
- `hal.h` - Hardware abstraction layer
- `init.h` - System initialization

12.30.1.9 Usage Scenarios

1. **Initial Setup:** Use option [1] to sync with NTP, then option [2] to save to RTC
2. **Power Recovery:** After power loss, use option [3] to restore system time from RTC
3. **Verification:** Use option [4] to confirm system and RTC synchronization
4. **Monitoring:** Use option [5] to display current time status
5. **Testing:** Use option [0] to test RTC persistence across resets

12.30.1.10 Limitations

- Requires active network connection for NTP synchronization
- DNS resolution needed for NTP server access
- RTC persistence depends on super-capacitor charge level
- Synchronization accuracy limited to minute-level precision

12.30.1.11 Applications

This demo serves as a foundation for:

- Data logging systems requiring accurate timestamps
- Scheduled task execution in embedded systems
- Time-sensitive industrial automation
- Network-connected IoT devices with offline operation requirements
- Real-time data acquisition systems

12.31 Overload Directory & System Files

Additional instructions if the overload directory is used in an NBEclipse project:

12.31.1 NetBurner Overload Directory Example

12.31.1.1 Overview

This example demonstrates how to use the NetBurner overload directory feature to override system files in a project-specific manner. The overload directory allows developers to modify system configuration files without affecting other projects or the original NNDK installation.

The Overload Directory Example is a basic demonstration of the overload directory in a project. The overload directory is used to override any system file by including local copies of system source files or headers. This allows the user to make changes to system files exclusively for a project. The overload directory provides an alternative to modifying the original system files, which would affect all projects.

To use this feature, the file to be overloaded should be placed in the overload directory under a directory structure that matches the file's location relative to the NNDK install directory.

This example application utilizes the overload directory to override [predef.h](#) (a NetBurner configuration header file). The overloaded [predef.h](#) defines the macro `NBRTOS_TIME`, which is not defined by default.

To overload the [predef.h](#) header, the directory structure is replicated inside the overload directory. For example, [predef.h](#) is located in the following directory in the NNDK install:

```
`/nburn/nbrtos/include/predef.h`
```

Therefore, the directory structure is replicated inside the overload directory with the following structure:

```
`<project root>/overload/nbrtos/include/predef.h`
```

Note

Upon adding a overload folder to the "Include paths" list or adding a new file to the specified overload folder, the user will need to clean and rebuild the NetBurner Library. This can be done by selecting "Clean NetBurner System Library" under the "Build Targets" pull-down in the project. From that point on, modifying [predef.h](#) (or any overloaded source/header) will automatically rebuild the necessary files in the NetBurner Library when the project builds.

12.31.1.2 Application Description

The **Overload Directory Example** is a simple NetBurner application that showcases the overload directory functionality by:

- Initializing the network stack
- Enabling system diagnostics
- Waiting for an active network connection
- Continuously prompting the user to press any key to display task runtime information
- Using the `ShowTaskTimes()` function, which is only available when the `NBRTOS_TIME` macro is defined

12.31.1.3 Key Features

12.31.1.3.1 Overload Directory Functionality

The overload directory provides a mechanism to override any system file by including local copies of system source files or headers. This approach offers several advantages:

- **Project-specific modifications:** Changes only affect the current project
- **Preserves original system files:** No modification of the original NNDK installation
- **Maintainable:** Easy to track and manage project-specific customizations

12.31.1.3.2 Example Implementation

This project demonstrates overloading the `predef.h` system header file to define the `NBRTOS_TIME` macro, which enables the `ShowTaskTimes()` function.

12.31.1.4 Directory Structure

The overload directory must replicate the directory structure of the file being overridden relative to the NNDK install directory.

12.31.1.4.1 Original System File Location

```
/nburn/nbrtos/include/predef.h
```

12.31.1.4.2 Overloaded File Location

```
<project root>/overload/nbrtos/include/predef.h
```

12.31.1.5 Files in This Example

12.31.1.5.1 main.cpp

The main application file that:

- Initializes the NetBurner system
- Waits for network connectivity
- Provides an interactive loop for displaying task runtime information
- Calls `ShowTaskTimes()` function (available due to overloaded `predef.h`)

12.31.1.5.2 predef-overload.h

Contains the overloaded configuration that defines:

```
#define NBRTOS_TIME (1)
```

This macro enables the compilation of task timing functionality in the NetBurner system library.

12.31.1.5.3 ReadMe.txt

The original documentation explaining the overload directory concept and usage instructions.

12.31.1.6 Setup Instructions for NBEclipse Projects

If using this example in an NBEclipse project, additional configuration is required:

1. **Right-click** on the project in the "Project Explorer"
2. **Select** "Properties"
3. **Navigate** to C/C++ Build -> Settings
4. **Select** GNU C++ Compiler -> Includes
5. **Add** the overload include folder to the "Include paths" list
6. **For C code:** Also add the overloaded folder to GNU C Compiler -> Includes
7. **Ensure** the overload directory path is positioned **above** paths beginning with `$(NNDK_ROOT)/`
8. **Use** up/down arrow buttons to reposition if necessary
9. **Select** "Apply and close"

12.31.1.7 Building and Maintenance

12.31.1.7.1 Initial Setup

After adding an overload folder to the "Include paths" list or adding new files to the overload folder:

1. **Clean and rebuild** the NetBurner Library
2. **Select** "Clean NetBurner System Library" under the "Build Targets" pull-down

12.31.1.7.2 Ongoing Development

Once properly configured, modifying overloaded files (like `predef.h`) will automatically rebuild the necessary NetBurner Library files when the project builds.

12.31.1.8 Application Behavior

When running the application:

1. The system initializes and waits for network connectivity
2. Application name and NNDK revision information are displayed
3. The program enters an interactive loop:
 - Prompts user to press any key
 - Displays current task runtime information using `ShowTaskTimes()`
 - Repeats indefinitely

12.31.1.9 Important Notes

- The `ShowTaskTimes()` function is only available when `NBRTOS_TIME` is defined in `predef.h`
- Improper include path configuration will result in "ShowTaskTimes was not declared" build errors
- System diagnostics are enabled in this example and should be removed for production code
- This example requires NetBurner hardware to execute

12.32 Ping

12.32.1 Ping Example

12.32.1.1 Overview

The Ping Test application is a network connectivity testing utility that allows users to verify the reach-ability and response time of remote devices on a network. This application demonstrates the basic implementation of ICMP ping functionality using hostname resolution and network diagnostics.

12.32.1.2 Features

- **Interactive Command Line Interface:** User-friendly prompt-based interaction for entering target addresses
- **Hostname Resolution:** Automatic DNS lookup to resolve hostnames to IP addresses
- **ICMP Ping Testing:** Sends ping packets to test network connectivity
- **Response Time Measurement:** Reports the time taken for ping responses in system ticks
- **Input Editing:** Basic command-line editing with backspace support
- **Continuous Operation:** Runs in a loop allowing multiple ping tests without restarting

12.32.1.3 How It Works

1. **User Input:** The application prompts the user to enter an IP address or hostname
2. **DNS Resolution:** Uses `GetHostByName()` to resolve hostnames to IP addresses
3. **Ping Execution:** Sends an ICMP ping packet using the `Ping()` function
4. **Result Display:** Shows the target IP address and response time or failure status
5. **Loop Continuation:** Returns to step 1 for continuous testing

12.32.1.4 Key Functions

12.32.1.4.1 `Gets_w_Edit(char *buf, int maxlen)`

Custom input function that provides basic line editing capabilities:

- Handles character input with buffer management
- Supports backspace (ASCII 8) for editing
- Terminates input on carriage return or newline
- Prevents buffer overflow by limiting input length

12.32.1.4.2 `UserMain(void *pd)`

Main application loop that:

- Initializes the system and enables diagnostics
- Manages the interactive ping testing cycle
- Handles DNS resolution and ping execution
- Displays results and error messages

12.32.1.5 Technical Details

- **Language:** C/C++
- **Dependencies:**
 - `init.h` - System initialization
 - `dns.h` - DNS resolution functions
- **Network Functions:**
 - `GetHostByName()` - DNS hostname resolution
 - `Ping()` - ICMP ping implementation
- **Timeout:** DNS resolution timeout set to 30 seconds
- **Ping Parameters:**
 - ID: 1
 - Sequence: 1
 - Maximum wait time: 100 ticks

12.32.1.6 Usage Example

```
Enter address to ping: google.com
Got:[google.com]
IP address is 172.217.12.110
Response Took 25 ticks
```

```
Enter address to ping: 8.8.8.8
Got:[8.8.8.8]
IP address is 8.8.8.8
Response Took 15 ticks
```

```
Enter address to ping: invalid-host.example
Got:[invalid-host.example]
Failed to resolve host
```

12.32.1.7 Error Handling

The application handles several error conditions:

- **DNS Resolution Failure:** Displays "Failed to resolve host" when hostname cannot be resolved
- **Ping Failure:** Shows "Ping Failed!" when the ping operation returns -1
- **Input Validation:** Prevents buffer overflow in user input

12.32.1.8 System Requirements

- Compatible embedded system or platform supporting the included header files
- Network connectivity capability
- ICMP protocol support
- DNS resolution functionality

12.32.1.9 Development Notes

- System diagnostics are enabled for debugging (should be disabled in production)
- The application runs indefinitely until manually terminated
- Input editing is basic but functional for typical use cases
- Response times are measured in system-specific ticks

12.32.1.10 Purpose

This application serves as a fundamental example of network connectivity testing and demonstrates:

- Basic network programming concepts
- DNS resolution implementation
- ICMP ping functionality
- Interactive command-line application design
- Error handling in network operations

The Ping Test utility is particularly useful for network troubleshooting, connectivity verification, and as a learning example for embedded network programming.

12.33 PLATFORM SPECIFIC

With the exception of the Platform Specific examples, all examples in the `\nburn\examples` directory can be run on any of the NetBurner 3.x platforms: MODM7AE70, MOD5441x, NANO54415 and SB800EX. The Platform Specific examples in this section have specific hardware requirements.

The example categories are:

- Examples designed for only one platform: MODM7AE70, MOD5441x, NANO54415 or SB800EX
- Multi-Platform: Some examples for platforms that use the same microprocessor take advantage of that fact and share the same application source code. In these instances the section will be labeled with the supported platforms, such as the section labeled: "MOD5441x, NANO54415, SB800EX".

Platform specific examples

- [MOD5441x, NANO54415, SB800EX](#)
- [MODM7AE70, SBE70LC](#)
- [SOMRT1061](#)

12.33.1 MOD5441x, NANO54415, SB800EX

Platform Platform specific examples for ColdFire MCF5441x based products:

- MOD54415
- MOD54417
- NANO54415
- SB800EX

- [1-Wire](#)
- [Analog to Digital](#)
- [CAN to Serial](#)
- [Digital to Analog \(DAC\)](#)
- [DSPI to Serial](#)
- [Load Application From Flash Card](#)
- [SDHC Flash Card](#)
- [EMI Emissions Reduction](#)
- [External NMI IRQ 7](#)
- [RTC - External](#)
- [I2C](#)
- [RTC - On Chip](#)
- [Periodic Interrupt Timer](#)
- [Pulse Generator and Counter](#)
- [PWM](#)
- [Core Watchdog Timer \(CWT\)](#)

- [WAV File Audio Player](#)
- [MOD5441x Only](#)
- [NANO54415 Only](#)
- [SB800EX Only](#)

12.33.1.1 1-Wire

Supported Platforms: MOD5441x, NANO54415

12.33.1.1.1 1-Wire DS2401 Silicon Serial Number Reader

12.33.1.1.1.1 Overview

This NetBurner application demonstrates how to interface with a DS2401 Silicon Serial Number chip using the 1-Wire protocol on MCF5441X-based platforms. The application continuously detects and reads the unique 64-bit registration number from connected DS2401 devices.

12.33.1.1.1.2 Supported Platforms

- **MOD5441X** - Uses pin J2[37] for 1-Wire data line
- **NANO54415** - Uses pin 12 for 1-Wire data line

12.33.1.1.1.3 Hardware Requirements

- NetBurner MOD5441X or NANO54415 module
- DS2401 Silicon Serial Number chip
- Pull-up resistor (4.7k ohm recommended) between data line and VCC
- Power supply connections (VCC and GND)

12.33.1.1.1.4 Hardware Connections

MOD5441X Platform

- **1-Wire Data:** J2[37] (configured as OW_DAT function)
- **Power:** Connect DS2401 VCC to 3.3V or 5V
- **Ground:** Connect DS2401 GND to system ground
- **Pull-up:** 4.7k ohm resistor between data line and VCC

NANO54415 Platform

- **1-Wire Data:** Pin 12 (configured as OW_DAT function)
- **Power:** Connect DS2401 VCC to 3.3V or 5V
- **Ground:** Connect DS2401 GND to system ground
- **Pull-up:** 4.7k ohm resistor between data line and VCC

12.33.1.1.1.5 Application Functionality

Main Features

1. **1-Wire Protocol Implementation:** Custom 1-Wire peripheral driver
2. **Device Detection:** Automatic detection of DS2401 devices on the bus
3. **Serial Number Reading:** Reads complete 64-bit registration number
4. **Continuous Monitoring:** Polls for devices every 5 seconds
5. **Formatted Output:** Displays family code, serial number, and CRC

Program Flow

1. Initialization

- Initialize network stack with `init()`
- Enable system diagnostics
- Wait for network activity (up to 5 seconds)
- Initialize 1-Wire module with `OWInit()`
- Configure platform-specific pin functions

2. Main Loop (repeats every 5 seconds)

- Detect devices on 1-Wire bus using `OWDetectDevices()`
- If device found:
 - Send Read ROM command (0x33) to DS2401
 - Read 8 bytes of registration data
 - Parse and display formatted output
- If no device found:
 - Display "Device not detected" message

Output Format

When a DS2401 is detected, the application displays:

```
- - - - - Device detected - - - - -
Family Code: 0x01
Hex Serial Number: 12 34 56 78 9A BC
CRC Code: 0xDE
```

DS2401 Registration Number Structure

The 64-bit registration number consists of:

- **Byte 0:** Family Code (0x01 for DS2401)
- **Bytes 1-6:** 48-bit unique serial number
- **Byte 7:** 8-bit CRC checksum

12.33.1.1.1.6 Key Components

Core Functions

- `**OWInit()`: Initializes the 1-Wire peripheral module
- `**OWDetectDevices()`: Scans bus for connected 1-Wire devices
- `**OWWrite()`: Sends data to 1-Wire device
- `**OWRead()`: Receives data from 1-Wire device
- `**OWDone()`: Checks if 1-Wire operation is complete

Semaphore Usage

The application uses an OS semaphore (`OS_SEM busSem`) to synchronize 1-Wire read operations, ensuring data integrity and proper timing.

12.33.1.1.1.7 Building and Deployment

Prerequisites

- NetBurner Development Kit installed
- Target platform environment variables set

12.33.1.1.1.8 Troubleshooting

Common Issues

1. ****"Device not detected" message****
 - Check physical connections
 - Verify pull-up resistor is installed (4.7k ohm)
 - Ensure DS2401 is powered correctly
 - Confirm correct pin assignment for platform
2. **Build errors**
 - Verify PLATFORM is set correctly (MOD5441X or NANO54415)
 - Check that NNDK_ROOT environment variable is set
 - Ensure all source files are present
3. **Invalid serial number readings**
 - Check data line signal integrity
 - Verify timing requirements are met
 - Ensure only one DS2401 device on bus for this example

Debug Output

The application includes diagnostic output showing:

- Application startup confirmation
- Device detection status
- Complete registration number breakdown
- Timing of detection attempts

12.33.1.1.1.9 Technical Notes

1-Wire Protocol Details

- Single data line for bidirectional communication
- Master-slave architecture (NetBurner as master)
- Time-slot based communication protocol
- Requires external pull-up resistor

DS2401 Specifications

- 64-bit lasered ROM with unique serial number
- Operating voltage: 2.8V to 6.0V
- Operating temperature: -40C to +85C
- Single-contact 1-Wire interface

12.33.1.1.1.10 Application Use Cases

This example demonstrates fundamental concepts applicable to:

- Asset tracking and identification
- Serial number logging systems
- Device authentication
- Inventory management
- Temperature sensor networks (with protocol modifications)

12.33.1.1.1.11 Related Examples

For additional 1-Wire functionality, consider exploring:

- Multi-device 1-Wire networks
- Temperature sensors (DS18B20)
- 1-Wire memory devices
- Advanced search algorithms for device enumeration

12.33.1.2 Analog to Digital

MOD5441x and NANO54415 Analog to Digital Platform Examples

- [Simple ADC](#)
- [Periodic ADC](#)

12.33.1.2.1 Simple ADC**12.33.1.2.1.1 Simple ADC - Analog to Digital Converter Example****Overview**

This NetBurner application demonstrates basic analog-to-digital conversion on MCF5441X-based platforms. The application continuously reads all 8 ADC channels and displays both raw register values and converted voltage readings on the serial console.

Supported Platforms

- **MOD5441X** - All 8 ADC channels available
- **NANO54415** - All 8 ADC channels available

Note: Refer to your specific platform's datasheet to determine which ADC channels are routed to accessible connector pins.

Hardware Requirements

- NetBurner MOD5441X or NANO54415 module
- Analog voltage sources (0-3.3V) connected to ADC input pins
- Serial console connection for output display

ADC Specifications

- **Resolution:** 15-bit (32,768 counts)
- **Input Range:** 0V to 3.3V
- **Channels:** 8 single-ended inputs (AN0-AN7)
- **Conversion Type:** Successive approximation
- **Reference Voltage:** 3.3V

Application Functionality

Main Features

1. **Single-Ended ADC Configuration:** Initializes all 8 channels for single-ended input
2. **Continuous Conversion:** Performs ADC conversions every second
3. **Dual Output Format:** Displays both raw hex values and calculated voltages
4. **Synchronous Operation:** Waits for conversion completion before reading results

Program Flow

1. Initialization

- Initialize network stack with `init()`
- Enable system diagnostics
- Initialize ADC hardware with `InitSingleEndAD()`

2. Main Loop (repeats every second)

- Start ADC conversion with `StartAD()`
- Wait for conversion completion using `ADDone()`
- Read and display all 8 channel results
- Convert raw values to voltage
- Wait 1 second before next cycle

Output Format

The application displays readings in two formats for all 8 channels:

```
Register values: 0000,0000,4A2B,0000,0000,0000,0000,0000,
Voltage: 0.0000,0.0000,1.5234,0.0000,0.0000,0.0000,0.0000,0.0000,
```

- **Register values:** Raw 16-bit hexadecimal ADC readings
- **Voltage:** Calculated floating-point voltage values (0.0000 to 3.3000V)

Key Components

Files Structure

```
SimpleADC/
|-- src/
|   |-- main.cpp      # Main application logic
|   |-- SimpleAD.cpp   # ADC driver implementation
|   |-- SimpleAD.h     # ADC function declarations
|-- makefile          # Build configuration
+-- README.md         # This documentation
```


Core Functions

- `**InitSingleEndAD()`: Configures ADC hardware for single-ended operation
- `**StartAD()`: Initiates conversion on all enabled channels
- `**ADDone()`: Returns true when current conversion cycle is complete
- `**GetADResult(int ch)`: Returns raw 16-bit result for specified channel (0-7)

Voltage Conversion Formula

$$\text{voltage} = (\text{raw\_value} * 3.3) / 32768.0$$

Where:

- `raw_value`: 16-bit ADC result (0-32767)
- `3.3`: Reference voltage
- `32768.0`: Full-scale count (2^{15})

Hardware Connections

ADC Channel Mapping

Consult your platform's datasheet for specific pin assignments:

- **Channel 0 (AN0)**: ADC input 0
- **Channel 1 (AN1)**: ADC input 1
- **Channel 2 (AN2)**: ADC input 2
- **Channel 3 (AN3)**: ADC input 3
- **Channel 4 (AN4)**: ADC input 4
- **Channel 5 (AN5)**: ADC input 5
- **Channel 6 (AN6)**: ADC input 6
- **Channel 7 (AN7)**: ADC input 7

Input Requirements

- **Voltage Range**: 0V to 3.3V (do not exceed 3.3V)
- **Input Impedance**: High impedance (suitable for most sensors)
- **Signal Conditioning**: May require buffering for low-impedance sources

Usage Examples

Testing with Known Voltages

1. **Ground Reference**: Connect ADC input to GND (should read $\sim 0.0000\text{V}$)
2. **3.3V Reference**: Connect ADC input to 3.3V supply (should read $\sim 3.3000\text{V}$)
3. **Variable Voltage**: Use potentiometer or voltage divider for variable input
4. **Sensor Testing**: Connect analog sensors (temperature, pressure, etc.)

Monitoring Multiple Channels

Connect different voltage sources to multiple channels to observe simultaneous readings:

Register values: 0000,7FFF,4000,2000,1000,0800,0400,0200,
Voltage: 0.0000,3.3000,1.6500,0.8250,0.4125,0.2063,0.1031,0.0516,

Troubleshooting

Common Issues

1. All readings show 0000

- Check that ADC inputs are properly connected
- Verify input voltages are within 0-3.3V range
- Ensure proper ground connections

2. Readings are unstable or noisy

- Add input filtering (RC filter recommended)
- Check for proper ground plane connections
- Verify signal source has adequate drive capability

3. Incorrect voltage calculations

- Verify reference voltage is exactly 3.3V
- Check ADC resolution settings in code
- Confirm conversion formula matches hardware configuration

Debug Tips

- Use multimeter to verify actual input voltages
- Connect known reference voltages for calibration
- Monitor ADC register values for consistency
- Check platform-specific pin assignments in datasheet

Technical Notes

ADC Characteristics

- **Conversion Time:** Platform-dependent (typically microseconds)
- **Sample Rate:** Limited by software delay (1 Hz in this example)
- **Accuracy:** Dependent on reference voltage stability
- **Temperature Drift:** Consult datasheet for specifications

Performance Considerations

- Current implementation uses polling for conversion completion
- For high-speed applications, consider interrupt-driven conversions
- Multiple channel readings are sequential, not simultaneous
- Adding input filtering may improve measurement stability

Application Extensions

This basic example can be extended for:

Sensor Integration

- Temperature sensors (thermistors, LM35)
- Pressure transducers
- Light sensors (photodiodes, photoresistors)
- Position sensors (potentiometers)

Advanced Features

- Averaging multiple readings for noise reduction
- Threshold monitoring with alarms
- Data logging to file system
- Web-based real-time monitoring
- Calibration and scaling for specific sensors

Code Modifications

```
// Example: Add averaging for channel 2
float avgVoltage = 0;
for(int sample = 0; sample < 10; sample++) {
    StartAD();
    while(!ADDone()) asm("nop");
    avgVoltage += ((double)GetADResult(2)) * 3.3 / 32768.0;
    OSTimeDly(TICKS_PER_SECOND / 100); // 10ms between samples
}
avgVoltage /= 10;
printf("Channel 2 averaged voltage: %6.4f\r\n", avgVoltage);
```

Related Examples

- **PeriodicADC:** Interrupt-driven ADC conversions
- **DAC Examples:** Digital-to-analog output generation
- **PWM Examples:** Pulse width modulation for analog control

12.33.1.2.2 Periodic ADC

12.33.1.2.2.1 Periodic ADC - DMA-Driven High-Speed Sampling

Overview

This NetBurner application demonstrates advanced analog-to-digital conversion using DMA-driven periodic sampling on MCF5441X-based platforms. The application continuously samples an ADC channel at 8 kHz, buffers the data using DMA transfers, and saves the captured audio data as WAV files to external flash storage.

Supported Platforms

- **MOD5441X** - Full DMA and external flash support
- **NANO54415** - Full DMA and external flash support

Hardware Requirements

- NetBurner MOD5441X or NANO54415 module
- External SD flash card (mounted on development board)
- Analog signal source connected to ADC channel 0
- Serial console connection for status monitoring

Key Features

Advanced ADC Capabilities

- **DMA-Driven Sampling:** Hardware-triggered transfers without CPU intervention
- **High-Speed Acquisition:** 8 kHz sampling rate with precise timing
- **Circular Buffering:** Continuous data capture with ping-pong buffers
- **Interrupt-Driven:** Half-buffer and full-buffer interrupt handling
- **Loop Mode ADC:** Continuous conversion without restart overhead

Data Processing

- **Real-Time Buffering:** 2048-byte circular sample buffer in fast SRAM
- **Large Data Buffer:** 8MB data buffer for extended recording
- **WAV File Generation:** Direct conversion to standard audio format
- **File System Integration:** EFFS FAT file system support
- **Automatic File Naming:** Sequential file numbering (adc8k0.wav, adc8k1.wav, etc.)

Technical Specifications

Sampling Configuration

- **Sample Rate:** 8 kHz (8000 samples per second)
- **ADC Channel:** Channel 0 (configurable)
- **Buffer Size:** 2048 bytes circular buffer
- **Data Buffer:** 8MB total storage
- **DMA Timer:** Timer 0 for precise triggering
- **Recording Length:** ~128 buffer cycles per file

File Output

- **Format:** WAV audio files
- **Channels:** 1 (mono)
- **Sample Rate:** 8 kHz
- **Bit Depth:** 16-bit (from ADC resolution)
- **File Size:** ~131 KB per recording cycle
- **Storage:** External SD flash card

Application Architecture

Program Flow

1. System Initialization

- Initialize network stack and diagnostics
- Initialize DMA semaphore for synchronization
- Mount external flash file system
- Configure ADC for continuous loop mode
- Setup DMA timer for 8 kHz triggering

2. ADC and DMA Setup

- Configure single-ended ADC on channel 0
- Start ADC in continuous loop mode
- Initialize DMA channel for ADC-to-memory transfers
- Set DMA timer frequency to 8 kHz

3. Data Acquisition Loop

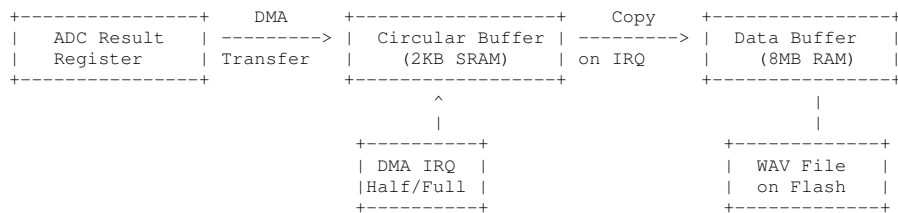
- Start DMA transfers to circular buffer

- Wait for half-buffer fill interrupts
- Copy data to large storage buffer
- Continue until major sample count reached
- Stop DMA and write WAV file

4. File Management

- Create sequential WAV files
- Write captured data with proper WAV headers
- Close files and increment counter
- Repeat indefinitely

Memory Architecture



Key Components

Files Structure

```

PeriodicADC/
|-- src/
|   |-- main.cpp           # Main application logic
|   |-- PeriodicAD.cpp     # ADC configuration functions
|   |-- PeriodicAD.h       # ADC function declarations
|   |-- PeriodicAD_DMA.cpp # DMA setup and control
|   |-- PeriodicAD_DMA.h   # DMA function declarations
|   |-- wavWriter.cpp      # WAV file format handling
|   |-- wavWriter.h        # WAV writer declarations
|   |-- FileSystemUtils.cpp # Flash file system utilities
|   |-- FileSystemUtils.h  # File system declarations

```

Core Functions

ADC Control:**

- `InitSingleEndAD()`: Configure ADC hardware for single-ended operation
- `StartADLoop()`: Start ADC in continuous conversion mode
- `initADC_DMA()`: Setup DMA channel for ADC transfers
- `setSampleFreq()`: Configure DMA timer for desired sample rate

DMA Management:**

- `startADC_DMA()`: Begin DMA transfers with callback
- `stopADC_DMA()`: Halt DMA operations
- `dmaTriggerFunc()`: Interrupt callback for buffer management

File Operations:**

- `openNewWav()`: Create new WAV file with proper headers
- `setUpFlash()`: Initialize external flash file system

Callback Function

```
void dmaTriggerFunc(void *start, void *end) {
    frontFilled = !frontFilled; // Toggle buffer half indicator
    AdcDmaSem.Post();           // Signal main loop
}
```

Hardware Connections

ADC Input

- **Channel 0:** Primary analog input (see platform datasheet for pin location)
- **Input Range:** 0V to 3.3V
- **Input Impedance:** High (suitable for audio signals)
- **Signal Conditioning:** May require amplification for low-level signals

External Flash

- **SD Card:** Required for WAV file storage
- **File System:** FAT format recommended
- **Capacity:** Minimum 1GB for extended recording

Building and Deployment

Required Libraries

This example requires the FatFile library for file system operations:
For NBEclipse:**

1. Right-click project > Properties
2. C/C++ Build > Settings
3. GNU C/C++ Linker > Libraries
4. Add "FatFile" to Libraries list

For Command Line:**
XTRALIB += -lFatFile

Additional Files Required

Copy these files from \nburn\examples_common\EFFS\FAT:

- cardtype.h
- FileSystemUtils.cpp
- FileSystemUtils.h

Build Commands

```
# Build for specific platform
make PLATFORM=MOD5441X
# or
make PLATFORM=NANO54415

# Load to device
make load DEVIP=<device_ip_address>
```

Usage and Operation

Startup Sequence

1. Connect analog signal to ADC channel 0
2. Insert formatted SD card
3. Load and run application
4. Monitor serial output for status

Expected Output

```
Initializing External Flash...  
External Flash opened.  
Starting log: adc8k0.wav  
Wrote 131072 bytes to disk, in file adc8k0.wav.  
Starting log: adc8k1.wav  
Wrote 131072 bytes to disk, in file adc8k1.wav.
```

File Analysis

Generated WAV files can be:

- Played in standard audio software
- Analyzed in audio/signal processing tools
- Converted to other formats as needed
- Opened in oscilloscope software for waveform analysis

Performance Characteristics

Timing Precision

- **DMA Timer:** Hardware-based precise timing
- **Jitter:** Minimal due to hardware triggering
- **CPU Loading:** Low due to DMA operation
- **Real-Time:** Deterministic sample intervals

Memory Usage

- **Fast SRAM:** 2KB circular buffer for real-time operation
- **Main RAM:** 8MB data buffer for file preparation
- **Stack Usage:** WAV writer requires significant stack space
- **File System:** Standard EFFS overhead

Throughput

- **Sample Rate:** 8000 samples/second
- **Data Rate:** 16 KB/s (assuming 16-bit samples)
- **File Rate:** One file every ~16 seconds
- **Sustained Rate:** Limited by flash write speed

Troubleshooting

Common Issues

1. ****"FAILED TO OPEN EXTERNAL FLASH"****
 - Check SD card insertion and formatting
 - Verify card compatibility (FAT16/FAT32)
 - Ensure proper card mounting on development board
2. **No audio in WAV files**
 - Verify analog signal connection to correct ADC pin
 - Check signal amplitude (0-3.3V range)
 - Confirm ADC channel configuration matches hardware
3. **DMA timing issues**
 - Verify DMA timer configuration
 - Check for timer conflicts with other peripherals
 - Monitor CPU loading for interference
4. **File system errors**
 - Reformat SD card with FAT32
 - Check available disk space
 - Verify file system initialization

Debug Techniques

```
Signal Verification:**
// Add to main loop for signal monitoring
printf("ADC Value: %d\r\n", GetADResult(ADC_CHANNEL));

Buffer Status:**
// Monitor buffer fill status
printf("Buffer half: %s\r\n", frontFilled ? "back" : "front");

DMA Status:**
// Check DMA operation
printf("DMA active: %s\r\n", isDMAActive() ? "yes" : "no");
```

Application Extensions

Audio Processing

- Real-time filtering (high-pass, low-pass, band-pass)
- Signal analysis (FFT, spectral analysis)
- Compression/decompression algorithms
- Echo/reverb effects

Multi-Channel Recording

- Simultaneous sampling of multiple ADC channels
- Stereo WAV file creation
- Channel synchronization
- Mixed-signal analysis

Advanced Features

- Triggered recording (threshold-based)
- Continuous streaming over network
- Real-time audio playback
- Data compression before storage

Integration Examples

- Audio monitoring systems
- Vibration analysis
- Environmental sound recording
- Signal quality measurement

Performance Optimization

High-Speed Modifications

```
// Increase sample rate to 16 kHz
#define SAMPLE_FREQ 16000

// Larger buffers for reduced interrupt frequency
#define BUFFERSIZE 4096
```

Memory Optimization

```
// Use external RAM for data buffer if available
uint8_t *dataBuffer = (uint8_t*)EXTERNAL_RAM_BASE;
```

Real-Time Processing

```
// Process data in DMA callback for minimum latency
void dmaTriggerFunc(void *start, void *end) {
    // Immediate signal processing here
    processAudioData(sampleBuffer + (frontFilled ? HALFBSIZE : 0));
    frontFilled = !frontFilled;
    AdcDmaSem.Post();
}
```

Related Examples

- **SimpleADC:** Basic ADC polling operation
- **WavPlayer:** Audio playback functionality
- **DSPI Examples:** High-speed serial data acquisition
- **DAC Examples:** Analog output generation
- **Timer Examples:** Precise timing control

12.33.1.3 CAN to Serial

12.33.1.3.1 CAN to Serial Bridge

12.33.1.3.1.1 Overview

This NetBurner application creates a bidirectional bridge between CAN bus and serial console communications on MCF5441X-based platforms. The application receives CAN messages and displays them on the serial console, while also allowing users to type messages that are transmitted over the CAN bus.

Warning

At least 2 CAN devices are required to avoid a hardware bus error.

12.33.1.3.1.2 Supported Platforms

- **MOD5441X** - CAN0: J2[42]/J2[39], CAN1: J2[44]/J2[41]
- **NANO54415** - CAN0: Pin 27/29, CAN1: Pin 22/20
- **SB800EX** - CAN0: PortB[2]/PortB[1], CAN1: PortB[0]/PortC[7]

12.33.1.3.1.3 Hardware Requirements

- NetBurner module with CAN capability
- CAN transceiver (if not integrated on board)
- CAN bus network with other nodes
- Serial console connection for user interface
- CAN bus termination resistors (120 ohm at each end)

12.33.1.3.1.4 CAN Configuration

Default Settings

- **CAN Module:** CAN0 (configurable to CAN1)
- **Baud Rate:** 500 kbps
- **Message Types:** Both standard (11-bit) and extended (29-bit) IDs
- **Receive Filters:**
 - Standard ID: 0x123
 - Extended ID: 0x1234

Message Filtering

The application accepts messages matching:

- **Standard ID 0x123:** Demonstrates 11-bit CAN ID reception
- **Extended ID 0x1234:** Demonstrates 29-bit CAN ID reception
- **Mask:** 0 (accepts all messages - modify for selective filtering)

12.33.1.3.1.5 Application Functionality

Main Features

1. **CAN Message Reception:** Listens for incoming CAN messages
2. **Message Display:** Shows received data in human-readable format
3. **Interactive Transmission:** Send messages via keyboard input
4. **Dual ID Support:** Handles both standard and extended CAN IDs
5. **Real-Time Operation:** Non-blocking message processing

Program Flow**1. Initialization**

- Initialize network stack and diagnostics
- Configure platform-specific CAN pins
- Initialize CAN controller at 500 kbps
- Register message receive FIFOs
- Set up message filters

2. Main Loop

- Poll for incoming CAN messages (1 tick timeout)
- Display received messages with ID and data
- Check for serial input from user
- Transmit user-entered messages on CAN bus

Message Format Display**Received Messages:****

```
Received a CAN message of 8 bytes from ID=291 or 0x123
Data = [Hello!!!]
```

```
Received a CAN message of 4 bytes from Extended ID=4660 or 0x001234
Data = [Test]
```

User Input:**

```
Type a message and press <enter> to transmit
> Hello World
Sending 11 bytes
Sent successfully
```

12.33.1.3.1.6 Hardware Connections**Pin Assignments by Platform****MOD5441X:****

- CAN0 TX: J2[42] (PINJ2_42_CAN0_TX)
- CAN0 RX: J2[39] (PINJ2_39_CAN0_RX)
- CAN1 TX: J2[44] (PINJ2_44_CAN1_TX)
- CAN1 RX: J2[41] (PINJ2_41_CAN1_RX)

NANO54415:**

- CAN0 TX: Pin 27 (PIN_27_CAN0_TX)
- CAN0 RX: Pin 29 (PIN_29_CAN0_RX)
- CAN1 TX: Pin 22 (PIN_22_CAN1_TX)
- CAN1 RX: Pin 20 (PIN_20_CAN1_RX)

SB800EX:**

- CAN0 TX: PortB[2] (tertiary function)
- CAN0 RX: PortB[1] (tertiary function)
- CAN1 TX: PortB[0] (primary function)
- CAN1 RX: PortC[7] (primary function)

CAN Bus Wiring

- **CAN_H**: Connect to CAN High line
- **CAN_L**: Connect to CAN Low line
- **GND**: Common ground with other nodes
- **Termination**: 120 ohm resistors at bus endpoints

12.33.1.3.1.7 Key Components

Core Functions

Platform Configuration:**

- `ConfigureCanPins(uint32_t canModuleNumber)`: Sets up CAN pins for each platform
- CAN Operations:**

- `MultiCanInit()`: Initialize CAN controller with baud rate
 - `RegisterMultiCanRxFifo()`: Set up receive message filtering
 - `MultiCanSendMessage()`: Transmit CAN messages
 - `CanRxMessage`: Class for receiving and parsing messages
- Helper Functions:**

- `NormToNbId()`: Convert standard ID to NetBurner format
- `ExtToNbId()`: Convert extended ID to NetBurner format
- `NbToNormId()`: Convert NetBurner format to standard ID
- `NbToExtId()`: Convert NetBurner format to extended ID
- `IsNBIdExt()`: Check if ID is extended format

12.33.1.3.1.8 Message Processing

Reception Processing

```
CanRxMessage can_msg(&fifo, 1); // Wait up to 1 tick
if (can_msg.IsValid()) {
    uint8_t buffer[8];
    uint8_t len = can_msg.GetData(buffer, 8);
    uint32_t id = can_msg.GetId();

    // Process message data
    // Display formatted output
}
```

Transmission Processing

```
char buffer[255];
fgets(buffer, 255, stdin); // Get user input

if (strlen(buffer) > 8) {
    buffer[8] = 0; // Truncate to CAN data limit
}

// Send with extended ID
MultiCanSendMessage(CAN_MODULE, ExtToNbId(0x1234), (uint8_t*)buffer, strlen(buffer), 50);
```

Data Display Format

The application displays non-printable characters in hex format:

- **Printable characters:** Shown as-is
- **Non-printable:** Displayed as {XX} hex values
- **Mixed data:** Combination of text and hex

Example: Data = [Hello{0A}{00}World]

12.33.1.3.1.9 Building and Deployment

Configuration Options

```
// Select CAN module (0 or 1)
#define CAN_MODULE (0)
// #define CAN_MODULE (1)

// Set baud rate
#define BAUD_RATE 500000
```

12.33.1.3.1.10 Usage Examples

Testing CAN Communication

1. Setup Test Network:

- Connect two NetBurner devices via CAN bus
- Ensure proper termination (120 ohm resistors)
- Configure both with same baud rate

2. Interactive Testing:

```
Type a message and press <enter> to transmit
> Test Message
Sending 12 bytes
Sent successfully
```

3. Monitor Reception:

```
Received a CAN message of 12 bytes from Extended ID=4660 or 0x001234
Data = [Test Message]
```

Message Filtering Examples

```
// Accept all messages (mask = 0)
int chan = RegisterMultiCanRxFifo(CAN_MODULE, NormToNbId(0x123), &fifo);

// Accept only specific ID (mask = 0xFFFF)
int chan = RegisterMultiCanRxFifo(CAN_MODULE, NormToNbId(0x150), &fifo, 0xFFFF);

// Accept range of IDs (0x400-0x4FF)
int chan = RegisterMultiCanRxFifo(CAN_MODULE, NormToNbId(0x424), &fifo, 0xFF00);
```

12.33.1.3.1.11 Troubleshooting

Common Issues

1. CAN initialization failed

- Check CAN transceiver power and connections
- Verify baud rate matches other network nodes
- Ensure CAN pins are properly configured
- Check for proper bus termination

2. No messages received

- Verify receive filters and masks
- Check physical bus connections

- Confirm other nodes are transmitting
- Monitor bus with oscilloscope/analyzer

3. Messages not sending

- Check transmit buffer availability
- Verify message length (less than or equal to 8 bytes)
- Ensure bus arbitration is working
- Check for bus-off conditions

4. Garbled data display

- Verify baud rate matches network
- Check for electrical noise/interference
- Ensure proper grounding between nodes
- Verify message format consistency

Debug Techniques

Bus Monitoring:**

```
// Add status monitoring
printf("CAN Status: 0x%08X\n", GetCanStatus(CAN_MODULE));
```

Message Statistics:**

```
// Track message counts
static uint32_t rxCount = 0, txCount = 0;
rxCount++; // Increment on receive
printf("RX: %ld, TX: %ld\n", rxCount, txCount);
```

Buffer Status:**

```
// Monitor FIFO status
printf("FIFO entries: %d\n", fifo.GetCount());
```

12.33.1.3.1.12 Advanced Features

Multiple Message Types

```
// Register multiple receive channels
int chan1 = RegisterMultiCanRxFifo(CAN_MODULE, NormToNbId(0x100), &fifo);
int chan2 = RegisterMultiCanRxFifo(CAN_MODULE, NormToNbId(0x200), &fifo);
int chan3 = RegisterMultiCanRxFifo(CAN_MODULE, ExtToNbId(0x12345), &fifo);
```

Enhanced Data Processing

```
// Process different message types
switch(NbToNormId(id)) {
    case 0x100: ProcessSensorData(buffer, len); break;
    case 0x200: ProcessControlData(buffer, len); break;
    case 0x300: ProcessStatusData(buffer, len); break;
}
```

Error Handling

```
// Comprehensive error checking
CAN_RETURN_CODE result = MultiCanSendMessage(CAN_MODULE, id, data, len, timeout);
switch(result) {
    case CAN_OK: printf("Sent successfully\n"); break;
    case CAN_TIMEOUT: printf("Send timeout\n"); break;
    case CAN_ERROR: printf("Send error\n"); break;
    case CAN_BUSY: printf("Bus busy\n"); break;
}
```

12.33.1.3.1.13 Application Extensions

Protocol Implementation

- CANopen protocol stack
- J1939 automotive protocol
- Custom application protocols
- Multi-master arbitration

Data Logging

- Log all CAN traffic to file system
- Timestamp message reception
- Statistical analysis tools
- Playback recorded sessions

Network Integration

- Forward CAN messages over TCP/UDP
- Web interface for CAN monitoring
- [MQTT](#) bridge for IoT integration
- RESTful API for remote control

Real-Time Processing

- Interrupt-driven message handling
- Priority-based message processing
- Real-time response to critical messages
- High-frequency data acquisition

12.33.1.3.1.14 Performance Characteristics

Timing

- **Message Latency:** Typically <1ms for receive processing
- **Throughput:** Up to 500 kbps bus capacity
- **CPU Loading:** Low due to hardware CAN controller
- **Response Time:** Limited by RTOS task scheduling

Scalability

- **Message Rate:** Hundreds of messages per second
- **Network Size:** Up to 127 nodes per network
- **Buffer Depth:** Configurable FIFO sizes
- **Concurrent Channels:** Multiple receive filters per module

12.33.1.3.1.15 Related Examples

- **Serial Communication:** UART/RS-232 bridging
- **Ethernet Bridge:** CAN to TCP/UDP gateway
- **Data Logging:** Persistent CAN message storage
- **Protocol Stacks:** Higher-level CAN protocols

12.33.1.4 Digital to Analog (DAC)

12.33.1.4.1 DAC - Digital to Analog Converter Example

12.33.1.4.1.1 Overview

This NetBurner application demonstrates the use of the integrated Digital-to-Analog Converter (DAC) on MCF5441X-based platforms. The application provides two operating modes: manual DAC output control and automatic waveform generation using DMA timer triggering.

12.33.1.4.1.2 Supported Platforms

- **MOD5441X** - DAC0 output on J2.9, Timer0 output on J2.36
- **NANO54415** - DAC0 output on J2.9, Timer0 output on Pin 19

12.33.1.4.1.3 Hardware Requirements

- NetBurner module with DAC capability
- **CRITICAL:** Analog reference voltage connection required
- Oscilloscope or voltmeter for output monitoring
- Optional: Load resistor for current measurement

Analog Reference Voltage Setup

MOD5441X Platforms:**

- Connect J2.2 (Vcc/3.3V) to J2.5 (Analog Reference Input)
- This provides 3.3V reference for full-scale DAC output

NANO54415 Platforms:**

- Connect P3.2 (Vcc/3.3V) to P2.8 (Analog Reference Input)
- This provides 3.3V reference for full-scale DAC output

WARNING: The DAC will not function without proper analog reference connection!

12.33.1.4.1.4 DAC Specifications

- **Resolution:** 12-bit (4096 levels: 0x000 to 0xFFFF)
- **Output Range:** 0V to Vref (typically 3.3V when properly configured)
- **Output Pin:** J2.9 (DAC0 output)
- **Reference Input:** Platform-specific analog reference pin
- **Update Rate:** Software controlled (Mode 1) or hardware triggered (Mode 2)

12.33.1.4.1.5 Application Functionality

Mode 1: Manual DAC Output Control

Operation:**

- Writes directly to DAC data register
- Steps output from 0V to 3.3V in 0x10 increments

- Continuous ramping until user presses any key
- Software-controlled timing
Features:**
- Direct register manipulation
- Real-time control
- Interactive operation
- Immediate response

Mode 2: Hardware-Triggered Waveform Generation

Operation:**

- Uses DMA Timer 0 as trigger source at 100 kHz
- Automatic waveform generation with hardware timing
- Steps from minimum to maximum values
- Hardware-synchronized operation
Features:**
- Precise timing via DMA timer
- Automatic waveform stepping
- Hardware synchronization
- Reduced CPU loading

12.33.1.4.1.6 Technical Implementation

DAC Configuration Registers

Control Register (DAC_CR):**

- DAC_CR_PDN: Power down control (0=on, 1=off)
- DAC_CR_UP: Count up direction
- DAC_CR_DOWN: Count down direction
- DAC_CR_AUTO: Automatic waveform mode
- DAC_CR_SYNC: External synchronization enable

Waveform Parameters:**

- step: Increment/decrement value (0x0100)
- max: Maximum output value (0x0FFF)
- min: Minimum output value (0x0000)

Hardware Configuration

```
// Enable DAC0 output drive
sim1.ccm.misccr2 |= MISCCR2_DAC0SEL;

// Disable conflicting ADC3 function
sim1.ccm.misccr2 &= ~(MISCCR2_ADC3EN);

// Route ADC3 input to DAC0 output for monitoring
sim2.adc.cal = ADC_CAL_DAC0;

// Power on DAC
sim2.dac[0].cr &= ~DAC_CR_PDN;
```

DMA Timer Setup

```
float frequency = 100000; // 100 kHz
float toggleTime = (1.0 / frequency) * 0.5;
float cpuClockPeriod = (1.0 / (CPU_CLOCK / 2));
uint32_t refTimerCount = (uint32_t)(toggleTime / cpuClockPeriod);

sim2.timer[0].tmr = 0x002A; // Toggle output, reset after reference
sim2.timer[0].trr = refTimerCount;
sim2.timer[0].tmr |= 0x0001; // Start timer
```

12.33.1.4.1.7 User Interface

Menu System

1. Simple DAC output writing to DAC data register
2. DAC waveform generation using DMA Timer 0

Operation Flow

1. Select operating mode (1 or 2)
2. Monitor DAC output on J2.9 with oscilloscope/voltmeter
3. Press any key to stop current operation
4. Return to menu for mode selection

Expected Outputs

Mode 1 - Manual Control:**

- Sawtooth waveform ramping from 0V to 3.3V
- Step size: ~0.2V increments (0x10 out of 0xFFFF)
- Software-limited update rate

Mode 2 - Timer Triggered:**

- Precise sawtooth waveform at 100 kHz trigger rate
- Hardware-synchronized stepping
- Timer output visible on designated pin

12.33.1.4.1.8 Hardware Connections and Testing

Basic Setup

1. **Power connections:** Ensure proper 3.3V and ground
2. **Reference voltage:** Connect analog reference as specified above
3. **DAC output:** Monitor J2.9 with oscilloscope or voltmeter
4. **Timer output** (Mode 2): Monitor timer pin for trigger signal

Test Procedures

Voltage Verification:**

- Mode 1: Should see ramping voltage 0V to 3.3V repeatedly
- Mode 2: Should see stepped waveform with precise timing
- Check reference voltage is stable 3.3V

Waveform Analysis:**

- Mode 1: Irregular timing due to software delays
- Mode 2: Precise 100 kHz trigger timing
- Both modes: 12-bit resolution stepping

Load Testing

```
// Optional: Add load resistor for current testing
// 1k resistor from J2.9 to ground
// Measure voltage across resistor for current calculation
```

12.33.1.4.1.9 Troubleshooting

Common Issues

1. No DAC output

- Check analog reference voltage connection
- Verify DAC power-on (PDN bit = 0)
- Confirm J2.9 pin configuration
- Measure reference voltage at analog input pin

2. Incorrect output voltage range

- Verify reference voltage is exactly 3.3V
- Check for voltage divider loading
- Confirm DAC data register values (0x000-0xFFF)

3. Mode 2 not working

- Verify DMA timer configuration and frequency
- Check timer output pin assignment
- Confirm DAC sync configuration
- Monitor timer output signal

4. Noisy output

- Add bypass capacitors near DAC output
- Check power supply stability
- Verify proper grounding
- Use appropriate load impedance

Debug Techniques

Register Monitoring:**

```
// Display current DAC value
iprintf("DAC Value: 0x%04X\r\n", sim2.dac[0].data);
```

```
// Check DAC control register
iprintf("DAC CR: 0x%04X\r\n", sim2.dac[0].cr);
```

Timer Verification:**

```
// Monitor timer operation
iprintf("Timer TMR: 0x%04X\r\n", sim2.timer[0].tmr);
iprintf("Timer TCN: 0x%04X\r\n", sim2.timer[0].tcn);
```

12.33.1.4.1.10 Advanced Features

Custom Waveform Generation

```
// Triangle wave example
if(sim2.dac[0].cr & DAC_CR_UP) {
    if(sim2.dac[0].data >= 0x0FFF) {
        sim2.dac[0].cr &= ~DAC_CR_UP;
        sim2.dac[0].cr |= DAC_CR_DOWN;
    }
} else {
    if(sim2.dac[0].data <= 0x0000) {
        sim2.dac[0].cr &= ~DAC_CR_DOWN;
        sim2.dac[0].cr |= DAC_CR_UP;
    }
}
```

Arbitrary Waveform Playback

```
// Sine wave lookup table
const uint16_t sineWave[256] = { /* 12-bit sine values */ };
static uint8_t index = 0;

sim2.dac[0].data = sineWave[index++];
if(index >= 256) index = 0;
```

Multiple Channel Support

```
// DAC1 configuration (if available)
sim1.ccm.misccr2 |= MISCCR2_DAC1SEL;
sim2.dac[1].cr &= ~DAC_CR_PDN;
sim2.dac[1].data = value1;
```

12.33.1.4.1.11 Performance Characteristics

Timing Specifications

- **Settling Time:** Typically <1 microsecond to 0.1% accuracy
- ****Update Rate**:** Limited by CPU (Mode 1) or timer frequency (Mode 2)
- ****Resolution**:** 12-bit (0.8mV steps with 3.3V reference)
- ****Linearity**:** Dependent on reference voltage stability

CPU Loading

- ****Mode 1**:** High CPU usage for continuous updates
- ****Mode 2**:** Low CPU usage with hardware triggering
- ****Interrupt Overhead**:** Minimal with DMA timer

12.33.1.4.1.12 Application Extensions

Signal Generation

- Function generators (sine, square, triangle)
- Audio signal synthesis
- Test pattern generation
- Calibration reference signals

Control Systems

- Analog control loops
- Servo motor control
- Variable reference generation
- Offset/bias voltage generation

Communication

- FSK modulation
- Analog data transmission
- Level shifting
- Interface voltage generation

Integration Examples

```
// Web-controlled DAC output
void SetDACFromWeb(uint16_t value) {
    if(value <= 0xFFFF) {
        sim2.dac[0].data = value;
    }
}

// ADC feedback control
void ADCControlledDAC() {
    uint16_t adcValue = GetADResult(0);
    uint16_t dacValue = ProcessControlLoop(adcValue);
    sim2.dac[0].data = dacValue;
}
```

12.33.1.4.1.13 Related Examples

- **ADC Examples:** Analog input measurement and feedback control
- **PWM Examples:** Alternative analog output methods
- **Timer Examples:** Precise timing and triggering
- **Waveform Generation:** Advanced signal synthesis techniques

12.33.1.5 DSPI to Serial

12.33.1.5.1 DSPI to Serial Bridge

12.33.1.5.1.1 Overview

This NetBurner application demonstrates the Dual Serial Peripheral Interface (DSPI) functionality by creating a bridge between DSPI and UART serial communications. Data received on UART1 is transmitted via DSPI, and the received DSPI data is sent back through UART1, creating a complete bidirectional communication bridge.

12.33.1.5.1.2 Supported Platforms

- **MOD5441X** - DSPI1 pins on J2 connector
- **NANO54415** - DSPI1 pins on carrier board

12.33.1.5.1.3 Hardware Requirements

- NetBurner module with DSPI capability
- UART1 connection for serial communication
- SPI slave device or loopback jumper for testing
- Serial terminal software for interaction

12.33.1.5.1.4 DSPI Pin Assignments

MOD5441X Platform

- **SCK (Clock):** J2[25] (PINJ2_25_DSPI1_SCK)
- **SOUT (Master Out):** J2[28] (PINJ2_28_DSPI1_SOUT)
- **SIN (Master In):** J2[27] (PINJ2_27_DSPI1_SIN)
- **PCS0 (Chip Select):** J2[30] (PINJ2_30_DSPI1_PCS0) - Optional/commented out

NANO54415 Platform

- **SCK (Clock):** Pin 31 (PIN_31_DSPI1_SCK)
- **SOUT (Master Out):** Pin 35 (PIN_35_DSPI1_SOUT)
- **SIN (Master In):** Pin 33 (PIN_33_DSPI1_SIN)
- **PCS0 (Chip Select):** Pin 15 (PIN_15_DSPI1_PCS0) - Optional/commented out

12.33.1.5.1.5 Application Functionality

Main Features

1. **UART to DSPI Bridge:** Transfers UART data via SPI protocol
2. **DSPI to UART Echo:** Returns received SPI data to UART
3. **Interrupt-Driven Transfers:** Uses semaphores for efficient operation
4. **Loopback Testing:** Simple jumper test without external devices
5. **Buffer Management:** 10KB buffers for large data transfers

Program Flow

1. Initialization

- Initialize network stack and diagnostics
- Configure DSPI1 pins for selected platform
- Initialize DSPI with default parameters
- Open UART1 at 115200 baud
- Create semaphore for DSPI synchronization

2. Main Loop

- Monitor UART1 for incoming data
- Transfer received data via DSPI
- Wait for DSPI completion using semaphore
- Send DSPI response back to UART1
- Repeat continuously

Data Flow Architecture

```

UART1 --> Read Buffer --> DSPI TX --> SPI Bus --> External Device
      ^                                     |
      |                                     v
+----- Write Buffer <----- DSPI RX <----- SPI Bus <-----+

```

12.33.1.5.1.6 Key Components

Files Structure

```
DSPI-to-Serial/
|-- src/
|   |-- main.cpp      # Complete application logic
|-- makefile          # Build configuration
+-- README.md         # This documentation
```

DSPI Configuration

Default Parameters:**

- **Module:** DSPI1 (DEFAULT_DSPI_MODULE)
- **Baud Rate:** 2 MHz (default)
- **Frame Size:** 8 bits (default)
- **Chip Select:** 0x0F (all CS inactive)
- **CS Polarity:** 0x0F (active low)
- **Clock Polarity:** 0x0 (idle low)
- **Clock Phase:** 0x1 (capture on leading edge)
- **Data Output:** Hi-Z when inactive

Core Functions

DSPI Operations:**

- `DSPIInit()`: Initialize DSPI with default or specified parameters
- `DSPIStart()`: Start interrupt-driven DSPI transfer
- `DSPIDone()`: Poll for transfer completion (alternative to semaphore)

Serial Operations:**

- `OpenSerial()`: Configure UART1 with specified parameters
- `dataavail()`: Check for incoming serial data
- `read()`: Read data from UART buffer
- `writeall()`: Write complete buffer to UART

Synchronization:**

- `OS_SEM`: Semaphore for interrupt-driven DSPI operation
- `Pend()`: Wait for DSPI transfer completion
- `Post()`: Signal transfer completion (done automatically by driver)

12.33.1.5.1.7 Testing and Usage

Loopback Testing (No External Device)

MOD5441X Setup:**

- Place jumper between J2[27] (SIN) and J2[28] (SOUT)
- This creates loopback for immediate echo testing

NANO54415 Setup:**

- Place jumper between Pin 33 (SIN) and Pin 35 (SOUT)
 - This creates loopback for immediate echo testing
- Expected Behavior:**

1. Type characters in serial terminal
2. Characters are sent via DSPI
3. Looped-back data returns via DSPI
4. Characters echo back to terminal

External SPI Device Testing

Connection Requirements:**

- Connect DSPI pins to SPI slave device
- Ensure common ground connection
- Match voltage levels (typically 3.3V)
- Add pull-up resistors if required by slave device

Device Examples:**

- SPI EEPROMs (25LC256, etc.)
- SPI sensors (accelerometers, gyroscopes)
- SPI displays (LCD controllers)
- SPI ADC/DAC devices

12.33.1.5.1.8 Building and Deployment

Build Commands

```
# Build for specific platform
make PLATFORM=MOD5441X
make PLATFORM=NANO54415

# Load to device
make load DEVIP=<device_ip_address>
```

Serial Terminal Setup

- **Baud Rate:** 115200
- **Data Bits:** 8
- **Parity:** None
- **Stop Bits:** 1
- **Flow Control:** None

12.33.1.5.1.9 Advanced DSPI Configuration

Custom Initialization Parameters

```
// Example: Custom DSPI setup
DSPIInit(1,           // DSPI Module 1
         4000000,      // 4 MHz baud rate
         16,           // 16-bit frame size
         0x0E,         // CS0 active, others inactive
         0x0F,         // All CS active low
         0x1,          // Clock idle high
         0x0,          // Capture on trailing edge
         FALSE,        // Drive output when inactive
         2,            // CS to clock delay
         3);           // Delay after transfer
```


Frame Size Options

- **4-32 bits:** Extended range beyond standard SPI
- **Automatic queuing:** Hardware manages transfer queues
- **Interrupt driven:** Efficient CPU utilization

Multiple Chip Select Management

```
// Different CS configurations
DSPIInit(1, 2000000, 8, 0x0E); // CS0 active
DSPIInit(1, 2000000, 8, 0x0D); // CS1 active
DSPIInit(1, 2000000, 8, 0x0B); // CS2 active
```

12.33.1.5.1.10 Performance Characteristics

Transfer Specifications

- **Maximum Baud Rate:** Up to 25 MHz (platform dependent)
- **Frame Sizes:** 4 to 32 bits (beyond standard 8/16-bit SPI)
- **Buffer Size:** 10KB for both TX and RX operations
- **Latency:** Interrupt-driven with minimal CPU overhead

Timing Characteristics

- **CS to Clock Delay:** Configurable timing
- **Inter-frame Delay:** Adjustable for slave device requirements
- **Clock Polarity/Phase:** Full SPI mode support (0,0 to 1,1)

12.33.1.5.1.11 Troubleshooting

Common Issues

1. No data transfer

- Check pin configuration and connections
- Verify DSPI initialization success
- Confirm UART1 is opening correctly
- Test with loopback jumper first

2. Corrupted data

- Check SPI clock polarity and phase settings
- Verify baud rate is appropriate for slave device
- Ensure proper voltage levels (3.3V logic)
- Add decoupling capacitors near DSPI pins

3. Semaphore timeout

- Check if DSPI transfer is completing
- Verify interrupt system is working
- Monitor DSPI module status registers
- Consider using polling mode as alternative

4. Serial communication issues

- Confirm UART1 baud rate (115200)
- Check serial terminal settings
- Verify UART1 pin connections
- Test with different terminal software

Debug Techniques

Transfer Monitoring:**

```
// Add debug output
iprintf("DSPI transfer started, size: %d\n", num);
DSPI_SEM.Pend(0);
iprintf("DSPI transfer completed\n");
```

Buffer Inspection:**

```
// Display buffer contents
for(int i = 0; i < num; i++) {
    iprintf("TX[%d]: 0x%02X, RX[%d]: 0x%02X\n",
        i, TXBuffer[i], i, RXBuffer[i]);
}
```

Status Monitoring:**

```
// Check DSPI status (platform-specific registers)
iprintf("DSPI Status: 0x%08X\n", GetDSPIStatus(DEFAULT_DSPI_MODULE));
```

12.33.1.5.1.12 Application Extensions

Protocol Implementations

SPI EEPROM Interface:**

```
void WriteEEPROM(uint16_t address, uint8_t *data, int length) {
    uint8_t cmd[3] = {0x02, address >> 8, address & 0xFF};
    DSPIStart(DEFAULT_DSPI_MODULE, cmd, rxBuf, 3, &DSPI_SEM);
    DSPI_SEM.Pend(0);
    DSPIStart(DEFAULT_DSPI_MODULE, data, rxBuf, length, &DSPI_SEM);
    DSPI_SEM.Pend(0);
}
```

Sensor Data Acquisition:**

```
int16_t ReadADC(uint8_t channel) {
    uint8_t cmd[3] = {0x01, (channel << 4), 0x00};
    uint8_t response[3];
    DSPIStart(DEFAULT_DSPI_MODULE, cmd, response, 3, &DSPI_SEM);
    DSPI_SEM.Pend(0);
    return ((response[1] << 8) | response[2]) & 0xFFFF;
}
```

Network Integration

TCP to DSPI Bridge:**

```
// Replace UART operations with TCP socket
int tcpFd = connect("192.168.1.100", 8080);
if(dataavail(tcpFd)) {
    int num = read(tcpFd, (char*)TXBuffer, 10000);
    DSPIStart(DEFAULT_DSPI_MODULE, TXBuffer, RXBuffer, num, &DSPI_SEM);
    DSPI_SEM.Pend(0);
    writeall(tcpFd, (char*)RXBuffer, num);
}
```

Web Interface Control:**

```
// Web-based SPI device control
void HandleSPICommand(uint8_t *cmd, int cmdLen, uint8_t *response) {
    DSPIStart(DEFAULT_DSPI_MODULE, cmd, response, cmdLen, &DSPI_SEM);
    DSPI_SEM.Pend(0);
}
```

Multi-Device Support

```
// Multiple SPI devices with different CS
void SelectDevice(int device) {
    uint8_t csConfig = 0x0F & ~(1 << device); // Active low
    DSPIInit(DEFAULT_DSPI_MODULE, 2000000, 8, csConfig);
}
```

12.33.1.5.1.13 Performance Optimization

High-Speed Transfers

```
// Optimize for maximum throughput
DSPIInit(DEFAULT_DSPI_MODULE, 25000000, 8, 0x0E, 0x0F, 0, 1, TRUE, 0, 0);
```

DMA Integration

```
// Use DMA for large transfers (if available)
void LargeDataTransfer(uint8_t *txData, uint8_t *rxData, int size) {
    // Platform-specific DMA setup
    StartDMATransfer(txData, rxData, size);
}
```

```
    WaitForDMAComplete();
}
```

Interrupt Optimization

```
// Minimize interrupt latency
void __attribute__((interrupt)) DSPIInterruptHandler() {
    // Keep handler minimal
    ClearDSPIInterrupt();
    SignalTransferComplete();
}
```

12.33.1.5.1.14 Related Examples

- **Serial Communication:** UART bridging and protocols
- **CAN to Serial:** Alternative communication bridging
- **I2C Examples:** Inter-IC communication protocols
- **SPI Device Drivers:** Specific SPI device implementations
- **Network Bridges:** TCP/UDP to peripheral communication

12.33.1.6 Load Application From Flash Card

12.33.1.6.1 EDFS Load Application From Flash Card

12.33.1.6.1.1 Overview

This NetBurner application demonstrates how to update firmware from a binary image file stored on a flash card. The application uses the Extra File Descriptor (ExtraFD) mechanism to create a file stream interface that enables direct firmware programming from flash card storage.

12.33.1.6.1.2 Supported Platforms

- **MOD5441X** - On-board microSD card socket support
- Support for external flash card interfaces (commented options available)

12.33.1.6.1.3 Hardware Requirements

- NetBurner MOD5441X module
- MicroSD card with firmware binary file
- Serial console for monitoring update process
- Formatted flash card (FAT16/FAT32)

12.33.1.6.1.4 Application Functionality

Main Features

1. **Flash Card Mounting:** Automatically detects and mounts microSD card
2. **Binary File Reading:** Reads application binary from flash card storage
3. **ExtraFD Stream Interface:** Creates file descriptor from flash file pointer
4. **Stream-Based Programming:** Programs firmware using NetBurner's stream update mechanism
5. **Automatic Reboot:** Restarts system after successful firmware update

Program Flow

1. Initialization

- Initialize network stack and file system
- Enter EDFS file system context
- Mount on-board flash card (microSD)

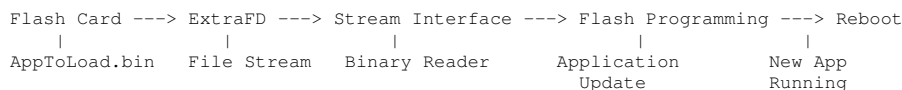
2. File System Operations

- Display directory contents
- Locate firmware file "AppToLoad.bin"
- Create Extra FD from file pointer

3. Firmware Update Process

- Read binary file through ExtraFD interface
- Program new application to flash memory
- Verify programming success
- Reboot system to run new firmware

Update Process Flow



12.33.1.6.1.5 Key Components

Files Structure

```

EffsLoadAppFromFlashCard/
|-- src/
|   |-- main.cpp           # Main application logic
|   +-- FileSystemUtils.h  # File system utility functions
|-- makefile               # Build configuration
+-- README.md              # This documentation
  
```

ExtraFD Implementation

Core ExtraFD Functions:**

- `fileToExtraFdRead()`: Reads data from flash file into buffer
- `fileToExtraFdWrite()`: Write function (not implemented in this example)
- `fileToExtraFdClose()`: Closes flash file and releases ExtraFD
- `makeFdFromFile()`: Creates file descriptor from `F_FILE` pointer

ExtraFD Structure:**

```

IoExpandStruct fileToExtraFd = {
    .read = fileToExtraFdRead,
    .write = fileToExtraFdWrite,
    .close = fileToExtraFdClose,
    .peek = 0
};
  
```

Stream Update Process

Key Functions:**

- `GetExtraFD()`: Allocates Extra File Descriptor
- `ReadBinaryApplicationCodeFromStream()`: Programs firmware from stream

- `ForceReboot ()` : Restarts system after successful update
Update Result Codes:**
- `STREAM_UP_OK`: Update completed successfully
- Error codes indicate specific failure reasons

12.33.1.6.1.6 File Preparation

Creating Update Files

1. Build Target Application:

```
make PLATFORM=MOD5441X
# This creates *.sl9 file in project directory
```

2. Convert to Binary Format:

```
# Use NetBurner tools or standard utilities
objcopy -O binary target.elf AppToLoad.bin
```

3. Copy to Flash Card:

- Name the file exactly "AppToLoad.bin"
- Place in root directory of flash card
- Ensure flash card is FAT16 or FAT32 formatted

File Requirements

- **Filename:** Must be exactly "AppToLoad.bin"
- **Location:** Root directory of flash card
- **Format:** Raw binary (not S-record or ELF)
- **Size:** Must fit in available flash memory
- **Compatibility:** Must be compiled for target platform

12.33.1.6.1.7 Building and Deployment

12.33.1.6.2 Build the updater application

12.33.1.6.3 Load updater to device

Required Libraries

This example requires EDFS FAT file system support:

Additional Files Required

Copy from \nburn\examples_common\EDFS\FAT:

- `FileSystemUtils.h`
- `FileSystemUtils.cpp`
- `cardtype.h` (if needed)

12.33.1.6.3.1 Usage Instructions

Step-by-Step Update Process

1. Prepare Flash Card:

- Format microSD card (FAT32 recommended)
- Copy "AppToLoad.bin" to root directory
- Verify file is present and readable

2. Load Updater:

- Build and load this updater application
- Insert prepared flash card into module
- Connect serial console for monitoring

3. Execute Update:

- Application will display directory contents
- Press any key when prompted to start update
- Monitor progress messages
- Wait for automatic reboot (10 seconds)

Expected Console Output

```

Initializing the on-board Flash card...
Drive change successful
Directory listing:
AppToLoad.bin          524288

Type any key to program new application to flash
Opening file: "AppToLoad.bin"
Update complete, system will reboot in 10 seconds...

```

12.33.1.6.3.2 Error Handling**Common Error Conditions****1. Drive Mount Failure:**

```
drive change failed: [error code]
```

- Check flash card insertion
- Verify card formatting (FAT16/FAT32)
- Ensure card is not corrupted

2. File Not Found:

```
Failed to find file: "AppToLoad.bin"
```

- Verify filename is exactly "AppToLoad.bin"
- Check file is in root directory
- Confirm file is not corrupted

3. Stream Update Failure:

```
ERROR: Stream update failed, error code: [code]
```

- Binary file may be corrupted
- Insufficient flash memory space
- Binary not compatible with target platform

4. ExtraFD Creation Failure:

```
Extra FD create failed
```

- System resource exhaustion
- File system integrity issues

Debug Techniques**File System Verification:****

```
// Add debug output to verify file system state
iprintf("Current directory: %s\n", GetCurrentDirectory());
iprintf("File size: %ld bytes\n", GetFileSize("AppToLoad.bin"));
```

Stream Progress Monitoring:**

```
// Monitor update progress
int totalBytes = fdReadToEnd(fd);
iprintf("Total bytes processed: %d\n", totalBytes);
```

12.33.1.6.3.3 Advanced Features

Custom Update Sources

External Flash Card Support:**

```
// Uncomment to use external flash card socket
// int drv = OpenOffBoardFlash(); // Dev board flash socket
int drv = OpenOnBoardFlash(); // Module microSD socket
```

Network-Based Updates:**

```
// Extend ExtraFD for network sources
int networkToExtraFdRead(int fd, char *buf, int nbytes) {
    // Read from TCP socket, HTTP, etc.
    return networkRead(buf, nbytes);
}
```

Multi-File Updates

```
// Update multiple components
char *updateFiles[] = {"bootloader.bin", "application.bin", "config.bin"};
for(int i = 0; i < 3; i++) {
    UpdateFromFile(updateFiles[i]);
}
```

Verification and Security

```
// Add checksum verification
uint32_t CalculateChecksum(int fd) {
    // Implement CRC32 or similar
    return checksum;
}

bool VerifyUpdate(const char *filename) {
    // Compare against stored checksum
    return checksumValid;
}
```

12.33.1.6.3.4 Safety Considerations

Update Reliability

1. **Power Supply:** Ensure stable power during update process
2. **Backup Strategy:** Keep previous firmware version accessible
3. **Verification:** Implement checksum validation before programming
4. **Recovery Mode:** Consider implementing emergency recovery mechanism

Flash Memory Protection

```
// Verify sufficient space before update
uint32_t requiredSpace = GetFileSize("AppToLoad.bin");
uint32_t availableSpace = GetFlashSpaceAvailable();
if(requiredSpace > availableSpace) {
    fprintf("ERROR: Insufficient flash memory\n");
    return;
}
```

Update Validation

```
// Validate binary format before programming
bool ValidateBinaryFormat(int fd) {
    // Check for valid application headers
    // Verify target platform compatibility
    return isValid;
}
```

12.33.1.6.3.5 Performance Characteristics

Update Timing

- **Read Speed:** Limited by flash card interface (typically 1-10 MB/s)
- **Programming Speed:** Limited by internal flash programming rate
- **Total Time:** Depends on application size (typically seconds to minutes)

Memory Usage

- **ExtraFD Overhead:** Minimal system resource usage
- **Buffer Size:** Configurable read buffer size
- **Flash Requirements:** Must fit in available application flash space

12.33.1.6.3.6 Application Extensions

User Interface Enhancements

```
// Progress indicator
void DisplayUpdateProgress(int bytesProcessed, int totalBytes) {
    int percent = (bytesProcessed * 100) / totalBytes;
    iprintf("Update progress: %d%%\r", percent);
}
```

Remote Update Management

```
// Web interface for update control
void HandleUpdateCommand(char *filename) {
    StartUpdateProcess(filename);
}
```

Batch Updates

```
// Update multiple devices
void BroadcastUpdate(char *filename) {
    // Send update to multiple networked devices
}
```

12.33.1.6.3.7 Related Examples

- **EFFS Examples:** File system operations and management
- **Stream Examples:** Alternative stream-based programming methods
- **Network Update:** Over-the-air firmware update mechanisms
- **Bootloader Examples:** Custom bootloader implementation
- **Flash Programming:** Direct flash memory programming techniques

12.33.1.7 SDHC Flash Card

12.33.1.7.1 EFFS SDHC Flash Card File System

12.33.1.7.1.1 Overview

This NetBurner application demonstrates comprehensive flash card file system operations using the Embedded Flash File System (EFFS) with FAT support. The application provides FTP server functionality, file management operations, and performance testing capabilities for SD/MMC/SDHC flash cards.

12.33.1.7.1.2 Supported Platforms

- **MOD5441X** - Multi-drive support (on-board + external)
- **NANO54415** - SDHC interface support
- **SB800EX** - SDHC interface support

12.33.1.7.1.3 Flash Card Interface Types

Interface Comparison

SPI Interface:**

- **MOD54415/MOD54417**: On-board microSD (SPI), MOD-DEV-70 dev board (SPI)
- **NANO54415**: Development board (SPI)
- Compatible with standard EDFS examples

SDHC Interface:**

- **NANO54415**: On-board microSD (SDHC)
- **SB800EX**: On-board microSD (SDHC)
- Higher performance, dedicated hardware controller

Note: SPI and SDHC are mutually exclusive - flash card connector must be wired for one type only.

12.33.1.7.1.4 Hardware Requirements

- NetBurner module with flash card interface
- SD/MMC/SDHC flash card (formatted FAT16/FAT32)
- Network connection for FTP access
- Serial console for interactive commands
- Optional: TEST.BIN file for speed testing

12.33.1.7.1.5 Application Functionality

Core Features

1. **File System Operations**: Directory listing, file display, space management
2. **FTP Server**: Network-based file transfer capabilities
3. **Performance Testing**: File read/write speed benchmarking
4. **Time Management**: NTP synchronization with manual fallback
5. **Interactive Menu System**: Serial console command interface

Program Flow

1. System Initialization

- Initialize network stack and file system
- Enter EDFS context for main and FTP tasks
- Initialize flash card interface

2. Time Setup

- Attempt NTP time synchronization
- Fall back to manual time setting if NTP fails
- Configure timezone (PST/PDT default)

3. Service Startup

- Start FTP server on port 21

- Display initial file system statistics
- Run read/write test with "TestFile.txt"

4. Interactive Operation

- Present command menu
- Process user commands
- Execute file operations

12.33.1.7.1.6 Command Interface

Available Commands

- **D**: Display Directory contents
- **E**: Display TestFile.txt contents
- **F**: Format SD flash card (WARNING: data loss!)
- **R**: Reboot system
- **S**: Display space usage statistics
- **T**: Display current system time
- **W**: Run file read/write speed test
- **?**: Display command menu

Command Examples

```
Directory Listing:**
Directory Contents:
TestFile.txt      1024
config.dat        512
data/             [DIR]
End of listing

Space Usage:**
Flash card space statistics:
Total: 32GB
Used: 1.2GB
Free: 30.8GB
```

12.33.1.7.1.7 Key Components

Files Structure

```
EffsSDHC/
|- src/
|   |-- main.cpp           # Main application logic
|   |-- FileSystemUtils.h  # File system utilities
|   |-- cardtype.h         # Card type definitions
|   |-- ftp_f.h            # FTP function declarations
|   |-- effs_time.h        # Time management functions
|   +-- dev_test.h         # Device test functions
|-- makefile               # Build configuration
+-- README.md              # This documentation
```

Platform-Specific Configuration

```
Multi-Drive Support (MOD5441X):**
#if (defined(MOD5441X))
#define MULTI_MMC TRUE
#include <effs_fat/multi_drive_mmc_mcf.h>

SDHC Interface:**
#elif (defined(USE_SDHC) && (defined(NANO54415) || defined(SB800EX)))
#include <effs_fat/sdhc_mcf.h>

Standard MMC/SD:**
#elif (defined(USE_MMC))
#include <effs_fat/mmc_mcf.h>
```

Core Functions

File System Management:**

- `f_enterFS()`: Enter file system context (required per task)
- `InitExtFlash()`: Initialize flash card interface
- `DumpDir()`: Display directory contents
- `DisplayEffsSpaceStats()`: Show storage usage

FTP Operations:**

- `FTPDStart()`: Start FTP server with specified priority
- Network-based file upload/download capabilities
- Long filename support detection

Performance Testing:**

- `SpeedTest()`: File I/O performance measurement
- `ReadWriteTest()`: Basic file operation verification

12.33.1.7.1.8 Time Management

NTP Synchronization

```
if (SetTimeNTP()) {
    // Set timezone (PST/PDT example)
    tzsetchar("PST8PDT7,M3.2.0/02:00:00,M11.1.0/02:00:00");
} else {
    // Manual time setting fallback
    SetTimeManual(month, day, dow, year, hour, min, sec);
}
```

Timezone Options

Available timezone configurations:

- Eastern: "EST5EDT4,M3.2.0/02:00:00,M11.1.0/02:00:00"
- Central: "CST6CDT5,M3.2.0/02:00:00,M11.1.0/02:00:00"
- Mountain: "MST7MDT6,M3.2.0/02:00:00,M11.1.0/02:00:00"
- Pacific: "PST8PDT7,M3.2.0/02:00:00,M11.1.0/02:00:00"

12.33.1.7.1.9 FTP Server Integration

Server Configuration

- **Port:** 21 (standard FTP)
- **Priority:** Higher than UserMain() (MAIN_PRIO - 2)
- **File System:** Shared access with main application
- **Authentication:** Platform-dependent default settings

FTP Operations

```
# Connect to NetBurner FTP server
ftp <device_ip_address>

# Upload files
put local_file.txt remote_file.txt

# Download files
get remote_file.txt local_file.txt

# Directory operations
ls          # List files
cd dirname  # Change directory
mkdir name  # Create directory
```

Long Filename Support

```
if (F_LONGFILENAME == 1)
    iprintf("Long file names are supported\r\n");
else
    iprintf("Long file names are not supported- only 8.3 format\r\n");
```

12.33.1.7.1.10 Performance Testing

Speed Test Requirements

1. Place file named "TEST.BIN" on flash card
2. Use FTP to upload test file, or copy from another device
3. Execute 'W' command from serial console
4. Results displayed through debug serial port

Test Parameters

```
int res = SpeedTest("TEST.BIN", 10 * 1024 * 1024); // 10MB test
```

Expected Performance

- **SPI Interface:** 1-5 MB/s typical
- **SDHC Interface:** 10-25 MB/s typical
- **Actual Speed:** Depends on card class and system loading

12.33.1.7.1.11 Building and Deployment

Build Commands

```
# Build for specific platform and interface type
make PLATFORM=MOD5441X      # Multi-drive MMC
make PLATFORM=NANO54415     # SDHC interface
make PLATFORM=SB800EX       # SDHC interface

# Load to device
make load DEVIP=<device_ip_address>
```

Conditional Compilation

The build system automatically selects the appropriate interface:

```
#if (defined(MOD5441X))
    // Multi-drive MMC support
#elif (defined(USE_SDHC) && (defined(NANO54415) || defined(SB800EX)))
    // SDHC interface
#elif (defined(USE_MMC))
    // Standard MMC/SD
#endif
```

Required Libraries

```
# EDFS FAT library (usually auto-included)
XTRALIB += -lFatFile
```

12.33.1.7.1.12 Usage Instructions

Initial Setup

1. **Prepare Flash Card:**
 - Format card as FAT32 (recommended) or FAT16
 - Optionally create test files
 - Insert into appropriate socket

2. Network Configuration:

- Ensure device has network connectivity
- Note device IP address for FTP access

3. Serial Console:

- Connect serial terminal (115200 baud)
- Use for interactive commands and monitoring

Typical Operation Session**Console Output:****

```
===== Starting EFFS FTP SD/SDHC Program =====
Initializing Flash Card...complete
External Flash opened.
NTP time synchronization successful
Current time: Wed May 14 11:35:00 2025
```

```
Flash card space statistics:
Total: 32GB, Used: 1.2GB, Free: 30.8GB
```

```
Started FTP Server
Long file names are supported
```

```
--- Main Menu ---
D - Display Directory
E - Display TestFile.txt
F - Format SD Flash card
R - Reboot
S - Display Space usage
T - Display system Time
W - Run File Read/Write Speed Test
? - Display Menu
```

FTP Access Example**From Windows Command Prompt:****

```
C:\> ftp 192.168.1.100
Connected to 192.168.1.100
220 NetBurner FTP Server Ready
User: admin
Password: admin
230 User logged in
```

```
ftp> ls
200 PORT command successful
150 Opening data connection
TestFile.txt      1024
config.dat        512
226 Transfer complete
```

```
ftp> get TestFile.txt
200 PORT command successful
150 Opening data connection
226 Transfer complete
1024 bytes received
```

12.33.1.7.1.13 Troubleshooting**Common Issues****1. Flash Card Not Detected:**

- Check card insertion and formatting
- Verify card compatibility (SD/MMC/SDHC)
- Ensure proper voltage levels (3.3V)
- Try different cards to isolate hardware issues

2. FTP Server Won't Start:

```
Error: [code]. Could not start FTP Server
```

- Check network connectivity

- Verify no other FTP server running on port 21
- Ensure sufficient system resources

3. File System Errors:

- Reformat flash card (FAT32 recommended)
- Check card for physical damage
- Verify file system integrity with PC tools

4. Time Synchronization Failure:

NTP failed, setting time manually

- Check network connectivity to NTP servers
- Verify DNS resolution working
- Manual time setting used as fallback

5. Speed Test Failure:

Test execution failed! Error [code]

- Ensure TEST.BIN file exists on card
- Check available free space
- Verify file read permissions

Debug Techniques

File System Status:**

```
// Add debugging output
iprintf("Free space: %ld KB\n", GetFreeSpace() / 1024);
iprintf("Card type: %s\n", GetCardType());
```

FTP Diagnostics:**

```
// Monitor FTP connections
iprintf("Active FTP sessions: %d\n", GetFTPSessionCount());
```

Performance Monitoring:**

```
// Time file operations
uint32_t startTime = Secs;
PerformFileOperation();
iprintf("Operation took %ld seconds\n", Secs - startTime);
```

12.33.1.7.1.14 Advanced Features

Custom File Operations

Batch File Processing:**

```
void ProcessAllFiles() {
    DIR *dir = opendir(".");
    struct dirent *entry;
    while((entry = readdir(dir)) != NULL) {
        if(entry->d_type == DT_REG) { // Regular file
            ProcessFile(entry->d_name);
        }
    }
    closedir(dir);
}
```

Directory Recursion:**

```
void ProcessDirectoryRecursive(const char *path) {
    DIR *dir = opendir(path);
    struct dirent *entry;
    while((entry = readdir(dir)) != NULL) {
        if(entry->d_type == DT_DIR && strcmp(entry->d_name, ".") != 0) {
            char newPath[256];
            snprintf(newPath, sizeof(newPath), "%s/%s", path, entry->d_name);
            ProcessDirectoryRecursive(newPath);
        }
    }
    closedir(dir);
}
```

Extended FTP Features

Custom FTP Commands:**

```
// Add application-specific FTP commands
void HandleCustomFTPCommand(const char *cmd, int clientFd) {
    if(strcmp(cmd, "STATUS") == 0) {
        SendFTPResponse(clientFd, "200 System Status OK\r\n");
    }
}
```

Authentication Enhancement:**

```
// Custom user authentication
bool AuthenticateUser(const char *username, const char *password) {
    // Implement custom authentication logic
    return ValidateCredentials(username, password);
}
```

Data Logging Applications

Sensor Data Logging:**

```
void LogSensorData() {
    F_FILE *fp = f_open("sensor_log.csv", "a");
    if(fp) {
        fprintf(fp, "%ld,%f,%f,%f\n",
                time(NULL), temperature, humidity, pressure);
        fclose(fp);
    }
}
```

System Event Logging:**

```
void LogEvent(const char *event) {
    F_FILE *fp = f_open("system_log.txt", "a");
    if(fp) {
        fprintf(fp, "[%s] %s\n", GetTimeString(), event);
        fclose(fp);
    }
}
```

12.33.1.7.1.15 Performance Optimization

High-Speed File I/O

```
// Use larger buffers for better performance
#define BUFFER_SIZE 32768
static char fileBuffer[BUFFER_SIZE];

void OptimizedFileCopy(const char *src, const char *dst) {
    F_FILE *srcFp = f_open(src, "rb");
    F_FILE *dstFp = f_open(dst, "wb");

    int bytesRead;
    while((bytesRead = f_read(fileBuffer, 1, BUFFER_SIZE, srcFp)) > 0) {
        f_write(fileBuffer, 1, bytesRead, dstFp);
    }

    fclose(srcFp);
    fclose(dstFp);
}
```

Memory Management

```
// Efficient memory usage for large files
void ProcessLargeFileInChunks(const char *filename) {
    F_FILE *fp = f_open(filename, "rb");
    char chunk[4096];

    while(f_read(chunk, 1, sizeof(chunk), fp) > 0) {
        ProcessChunk(chunk, sizeof(chunk));
    }

    fclose(fp);
}
```

Concurrent Access Management

```
// Thread-safe file operations
OS_SEM fileSemaphore;

void SafeFileWrite(const char *filename, const char *data) {
    fileSemaphore.Pend();
    F_FILE *fp = f_open(filename, "a");
    fprintf(fp, "%s", data);
}
```

```

    f_close(fp);
    fileSemaphore.Post();
}

```

12.33.1.7.1.16 Integration Examples

Web Interface Integration

```

// Web-based file management
void HandleFileUpload(int httpFd, const char *filename) {
    F_FILE *fp = f_open(filename, "wb");
    // Handle HTTP file upload
    SaveUploadedFile(httpFd, fp);
    f_close(fp);
}

```

Database-Like Operations

```

// Simple key-value storage
void StoreKeyValue(const char *key, const char *value) {
    F_FILE *fp = f_open("database.txt", "a");
    fprintf(fp, "%s=%s\n", key, value);
    f_close(fp);
}

char *GetValue(const char *key) {
    // Search through database.txt and return value
    return FindValueInFile("database.txt", key);
}

```

12.33.1.7.1.17 Related Examples

- **EFFS Examples:** Additional file system operations and utilities
- **FTP Examples:** Enhanced FTP server implementations
- **Data Logging:** Specialized logging and data acquisition systems
- **Network Storage:** Network-attached storage implementations
- **Backup Systems:** Automated backup and synchronization tools

12.33.1.8 EMI Emissions Reduction

12.33.1.8.1 EMI Emissions Reduction Example

12.33.1.8.1.1 Overview

This example demonstrates how to implement EMI (Electromagnetic Interference) reduction options for NetBurner MCF5441x platforms. The application provides an interactive menu system to test various emission reduction techniques that can help minimize electromagnetic interference in your embedded designs.

12.33.1.8.1.2 Supported Platforms

- **MOD5441x**
- **NANO54415**
- **SB800EX**

12.33.1.8.1.3 Features

The example provides the following EMI reduction capabilities:

1. Peripheral Clock Management

- **Disable unused peripheral clocks** - Turns off clocks to peripherals not being used by the application
- **Enable peripheral clocks** - Restores all peripheral clocks to their default enabled state

2. Spread Spectrum Clock Mode

- **Enable/Disable spread spectrum** - Available on MOD54415 hardware revisions 2.3 and later
- Helps reduce peak emissions by spreading the clock frequency over a range

3. Ethernet PHY Control

- **Disable Ethernet PHY** - Turns off the physical layer interface when network connectivity is not needed
- **Enable Ethernet PHY** - Restores normal Ethernet functionality

4. Signal Slew Rate Control

- **Reduce signal slew rates** - Configures pins for minimum slew rate to reduce high-frequency emissions
- **Restore default slew rates** - Returns pin configuration to reset defaults

5. Power Management

- **Wait mode idling** - Puts the processor in low-power wait mode when idle
- **Normal idling** - Uses standard polling loop when idle

12.33.1.8.1.4 Menu Options

When running the application, you'll see an interactive menu with these options:

Select EMC test to perform:

```
-----
1) Disable Peripheral Clocks
2) Enable Peripheral Clocks
3) Disable Spread Spectrum Mode on Primary CPU Oscillator
4) Enable Spread Spectrum Mode on Primary CPU Oscillator
5) Disable Ethernet PHY
6) Enable Ethernet PHY
7) Reduce Signal Slew Rates
8) Restore Default Signal Slew Rates
9) Idle in wait mode
10) Idle normally
```

12.33.1.8.1.5 Important Notes

Critical Peripherals

The application preserves functionality of essential system components:

- **UART 0** - Debug console remains enabled
- **FlexBus** - Required for application flash memory access
- **Interrupt Controllers** - System interrupts remain functional
- **PIT 0** - OS TimeTick timer stays enabled
- **DDR RAM** - External memory interface preserved
- **PLL** - Primary CPU clocks maintained
- **GPIO** - Module hardware GPIO functionality preserved

Signal Slew Rate Considerations

- Flash access timing may be disrupted at lowest slew rates
- Ethernet (FEC) functionality may be affected by reduced slew rates
- Multiple slew rate options are available as commented alternatives

Spread Spectrum Support

Spread spectrum mode is only available on:

- MOD54415 platform
- Hardware revision 2.3 and later

12.33.1.8.1.6 Usage

1. Build and deploy the application to your NetBurner device
2. Connect via serial console or Telnet
3. Use the interactive menu to test different EMI reduction techniques
4. Monitor emissions using appropriate test equipment
5. Combine multiple techniques as needed for your specific application

12.33.1.8.1.7 Additional EMI Considerations

This code example addresses software-configurable EMI reduction options. The associated documentation (Emi← Suggestions20210218.pdf) also covers:

- **Ethernet cabling** design considerations
- **Cable routing** and shielding best practices
- **Enclosure design** for emission reduction
- **Input power filtering** (very important for EMI compliance)
- **PCB routing** techniques and guidelines

12.33.1.8.1.8 File Structure

```
EMI-EmissionReduction/
|-- src/
|   +-- main.cpp           # Main application source
|-- makefile               # Build configuration
|-- ReadMe.txt             # Original documentation
|-- EmiSuggestions20210218.pdf # EMI design guidelines
+-- README.md              # This file
```

12.33.1.8.1.9 Application Architecture

The application uses a simple menu-driven architecture:

- **UserMain()** - Main application loop with menu display and input handling
- **DisablePeripheralClocks()** - Systematic peripheral clock disabling
- **EnablePeripheralClocks()** - Restore all peripheral clocks
- **ReduceSignalSlewRates()** - Configure minimum slew rates for all pins
- **RestoreDefaultSlewRates()** - Return to reset default slew rates
- **SpreadSpectrumOscillator()** - Control spread spectrum mode (MOD54415 only)

12.33.1.9 External NMI IRQ 7

12.33.1.9.1 External IRQ 7 (Non-Maskable Interrupt) Example

12.33.1.9.1.1 Overview

This example demonstrates how to configure and use External IRQ 7 as a non-maskable interrupt on NetBurner MCF5441X platforms. The application shows both C/C++ and assembly language interrupt service routine (ISR) implementations, providing a complete example of level 7 interrupt handling.

12.33.1.9.1.2 Supported Platforms

- **MOD5441X** - MOD54415 development module
- **NANO54415** - Nano54415 development module

Both platforms use the MCF54415 processor with identical interrupt functionality.

12.33.1.9.1.3 Hardware Requirements

Physical Connection Required

This example requires a jumper wire connection between a GPIO pin and the IRQ 7 input pin to trigger the interrupt:

MOD5441X Platform

GPIO Signal Pin: J2.33 ----[JUMPER]---- IRQ 7 Input: J2.48

NANO54415 Platform (Dev Board Rev 1.3)

GPIO Signal: Pin 12 (P2.12) ----[JUMPER]---- IRQ 7: Pin 9 (P1.9)

Pin Configuration Summary

Platform	GPIO Signal Pin	IRQ 7 Input Pin
MOD5441X	J2.33	J2.48
NANO54415	Pin 12 (P2.12)	Pin 9 (P1.9)

12.33.1.9.1.4 Features

Interrupt Service Routine (ISR) Implementation

- **Mixed C/C++ and Assembly** - Demonstrates both programming approaches
- **Register preservation** - Complete context saving and restoration
- **Interrupt flag clearing** - Proper interrupt acknowledgment
- **Counter mechanism** - Tracks number of interrupts received

GPIO Control

- **Signal generation** - Uses GPIO pin to trigger interrupts
- **Edge detection** - Configurable for falling, rising, or both edges
- **Manual triggering** - Interactive testing via serial console

Web Interface

- **Built-in web server** - Provides web-based interface
- **Status display** - Shows interrupt counter and system information

12.33.1.9.1.5 Code Architecture

Main Components

1. **UserMain()** - Application entry point and main loop
2. **ConfigureIrq7()** - IRQ 7 configuration function
3. **NMI_ASM_Part()** - Assembly language ISR entry point
4. **NMI_C_Part()** - C/C++ interrupt handler
5. **Function_Holder()** - Assembly code container function

Interrupt Flow

```

Hardware IRQ 7 Signal
    |
    v
NMI_ASM_Part() (Assembly)
    - Save processor state
    - Mask interrupts (SR = 0x2700)
    - Save all registers (D0-D7, A0-A6)
    |
    v
NMI_C_Part() (C/C++)
    - Clear interrupt flag (EPFR)
    - Reset GPIO trigger pin
    - Increment counter
    |
    v
NMI_ASM_Part() (Assembly continued)
    - Restore all registers
    - Restore processor state
    - Return from exception (RTE)

```

Register Usage

Assembly Code Register Operations

```

move.w #0x2700,%sr          ; Set interrupt mask to level 7
lea -60(%a7),%a7            ; Allocate 60 bytes on stack
movem.l %d0-%d7/%a0-%a6, (%a7) ; Save all general registers
jsr NMI_C_Part              ; Call C handler
movem.l (%a7),%d0-%d7/%a0-%a6 ; Restore all registers
lea 60(%a7),%a7             ; Restore stack pointer
rte                         ; Return from exception

```

12.33.1.9.1.6 Usage Instructions

Setup Process

1. **Build the application**
2. **Deploy to device**
3. **Install jumper wire**
 - Connect GPIO signal pin to IRQ 7 input pin according to platform specifications above
4. **Connect serial terminal**
 - Use MTTTY, PuTTY, or similar terminal program
 - Configure for 115200 baud, 8-N-1

Interactive Operation

1. **Start the application** - Device will display initialization messages
2. **Press any key** - Triggers a falling edge interrupt on IRQ 7
3. **View results** - Interrupt counter increments and displays current count
4. **Repeat testing** - Continue pressing keys to generate more interrupts

Example Console Output

```

MOD5441x IRQ 7 application started
Configuring J2-33 as GPIO
Configuring J2-48 as IRQ7
Hit any key to generate an IRQ
[User presses key]
Irq Count: 1
Hit any key to generate an IRQ
[User presses key]
Irq Count: 2
Hit any key to generate an IRQ

```

Web Interface

Access the web interface at `http://<device_ip>/` to view:

- Welcome page with NetBurner branding
- System status information
- Interrupt counter display

12.33.1.9.1.7 Technical Details

Interrupt Configuration

EPORT (External Port) Registers

- **EPPAR** - Pin assignment register (edge/level configuration)
- **EPDDR** - Data direction register (input/output configuration)
- **EPIER** - Interrupt enable register
- **EPFR** - Flag register (interrupt status)

Edge Detection Settings (EPPAR Register)

```
00: Level-sensitive (not used in this example)
01: Rising edge triggered
10: Falling edge triggered (used in this example)
11: Both edges triggered
```

Interrupt Controller Configuration

- **Vector:** 7 (IRQ 7 source)
- **Level:** 7 (highest priority, non-maskable)
- **Controller:** 0 (primary interrupt controller)

Memory Layout

Stack Frame During Interrupt

Stack Layout (60 bytes total):

D0	D1	D2	D3	Bytes 0-15
D4	D5	D6	D7	Bytes 16-31
A0	A1	A2	A3	Bytes 32-47
A4	A5	A6	---	Bytes 48-59

12.33.1.9.1.8 Important Considerations

IRQ 7 Characteristics

- **Non-maskable** - Cannot be disabled by interrupt mask bits
- **Highest priority** - Preempts all other interrupts
- **Critical timing** - Must be handled quickly to avoid system issues

Development Notes

- **Register preservation** is critical in assembly ISR
- **Stack management** must be precise (60-byte allocation)
- **Interrupt flag clearing** prevents continuous triggering
- **GPIO reset** ensures clean signal transitions

Troubleshooting

Common Issues

1. **No interrupts** - Check jumper wire connection
2. **Continuous interrupts** - Verify interrupt flag clearing in EPFR
3. **System instability** - Check assembly code register preservation
4. **Build errors** - Ensure SUPPORTED_PLATFORMS matches your hardware

Debug Steps

1. Verify jumper wire connections match platform specifications
2. Check serial console output for configuration messages
3. Confirm GPIO pin functions are set correctly
4. Test with multimeter to verify signal transitions

12.33.1.9.1.9 Learning Objectives

This example teaches:

- **Mixed-language programming** - C/C++ with assembly integration
- **Interrupt handling** - Non-maskable interrupt service routines
- **Hardware interface** - GPIO and external interrupt configuration
- **Register programming** - Direct MCF54415 register manipulation
- **Real-time concepts** - Critical timing and context preservation

12.33.1.9.1.10 Build Configuration

Makefile Settings

- **Platforms:** NANO54415, MOD5441X
- **Web content:** Auto-generated from html/ directory
- **Compile:** Standard NetBurner C++ compilation

Dependencies

- NetBurner NNDK (Network Development Kit)
- Compatible hardware platform
- Serial terminal program for interaction

12.33.1.9.1.11 License and Support

This example is provided under NetBurner's standard licensing terms. For technical support:

- Review the hardware platform documentation
- Consult the MCF54415 processor reference manual
- Contact NetBurner technical support for specific issues

12.33.1.9.1.12 Related Examples

For additional interrupt and GPIO examples, see:

- Basic GPIO examples for pin control
- Timer interrupt examples for periodic processing
- External interrupt examples for maskable interrupts (IRQ 1-6)

12.33.1.10 RTC - External

External [I2C](#) RTC example

Supported Platforms: MOD5441x, NANO54415

HARDWARE SETUP: Tested with MOD54415 on a MOD-DEV-70 hardware rev 1.93.

This example demonstrates the following features:

- Updates the system time with the current UTC time pulled from an available NTP server (pool.ntp.org). If your module is set to static IP address settings, then you will need to manually assign a DNS server address for this feature to work.
- Take the current system time and save it to the real-time clock (RTC). The RTC will then be running in sync with the system time.
- Take the current time in the RTC and set the system time information. The system time will then be running in sync with the RTC.
- When the device or module is reset or powered on again after a recent power loss, the system time will be reset, but the RTC can continue running until the super-capacitor is discharged. The software-reset option is available to demonstrate the RTC's ability to retain saved time information once it is properly set.
- Set the local time zone information with a call to [tzsetchar\(\)](#). For more more information on this function, refer to "Chapter 14 - NBTime Library" of the NetBurner Runtime Libraries PDF document. In this example, Pacific time zone information is used as a demonstration.
- Display the current system and RTC times, and local time zone information if applicable. It is preferred that the RTC store the UTC time, not local time.
- In order to take advantage of setting and getting time information from the real-time clock (RTC), a device that contains an RTC component or a module that is mounted on a development board with an RTC component must be used. Current RTC components supported in this example at the time of this writing are the Intersil X1226 and NXP PCF8563.

12.33.1.11 I2C

MCF5441x [I2C](#) Platform Examples

- [I2C Address Scan](#)
- [I2C PicKit Demo Board](#)

12.33.1.11.1 I2C Address Scan

12.33.1.11.1.1 I2C Address Scanner

Overview

This NetBurner application scans all possible I2C device addresses (1-127) on the I2C bus and reports which addresses respond with an ACK signal. The application is useful for discovering I2C devices connected to the bus and identifying their addresses for communication.

Supported Platforms

- **MOD5441X** - I2C0 default module
- **NANO54415** - I2C0 default module
- **MCF5441X OEM** - Configurable I2C module

Hardware Requirements

- NetBurner module with I2C capability
- I2C devices connected to the bus for testing
- Pull-up resistors on SDA and SCL lines (typically 4.7k)
- Serial console for viewing scan results

I2C Bus Configuration

Default Settings

- **I2C Module:** I2C0 (DEFAULT_I2C_MODULE)
- **NetBurner Address:** 0x08 (7-bit slave address)
- **Scan Range:** 1 to 127 (0x01 to 0x7F)
- **Operation Mode:** Master mode scanning

Address Format

The application reports addresses in both formats:

- **7-bit Address:** Standard I2C addressing format (0x50)
- **8-bit Address Byte:** Full address byte sent on bus (0xA0 for write, 0xA1 for read)

Application Functionality

Main Features

1. **Complete Address Scan:** Tests all 127 possible I2C addresses
2. **Device Detection:** Identifies responding devices by ACK signal
3. **Dual Address Format:** Reports both 7-bit and 8-bit address formats
4. **Interactive Operation:** Allows repeated scans on user command
5. **Error Handling:** Proper I2C bus reset on communication failures

Program Flow

1. Initialization

- Initialize network stack and diagnostics
- Initialize I2C module with NetBurner as address 0x08
- Display initialization status

2. Address Scanning

- Loop through addresses 1-127 (0x01-0x7F)
- Send write start condition to each address
- Check for ACK response within timeout period
- Report responding devices with address information

3. Interactive Mode

- Wait for user input to repeat scan
- Display total number of responding devices
- Continuous operation until reset

Scan Process Flow

```

Start -> Address 1 -> Send Start+Write -> ACK? -> Report Device -> Next Address
      ^                                     |
      |                                     v
      +-- Address 127 <- Bus Reset <-- NACK/Timeout
  
```

Key Components

Core Functions

I2C Operations:**

- `MultiChannel_I2CInit()`: Initialize I2C module with slave address
- `MultiChannel_I2CStart()`: Send start condition with address and direction
- `MultiChannel_I2CStop()`: Send stop condition to end transaction
- `I2CMultiChannelResetPeripheral()`: Reset I2C module on errors

Scan Function:**

- `scanAddressesI2C()`: Main scanning routine that tests all addresses
- Returns count of responding devices

Output Format

Typical Scan Results

```

Initializing address: 0x08 ... Complete
Scanning peripheral I2C0...
Found device, 7-bit addr: 0x08, 8-bit addr byte: 0x10, I2C Status: 0
Found device, 7-bit addr: 0x50, 8-bit addr byte: 0xA0, I2C Status: 0
Found device, 7-bit addr: 0x68, 8-bit addr byte: 0xD0, I2C Status: 0
Scan complete
3 addresses responded

Type any key to repeat
  
```

Address Information Explained

Example: 7-bit address 0x50**

- **7-bit addr:** 0x50 (binary: 1010000)
 - **8-bit addr byte:** 0xA0 (binary: 10100000) - with write bit (0)
 - **Read address byte:** 0xA1 (binary: 10100001) - with read bit (1)
- Status Codes:**
- **0:** Successful ACK received
 - ****>= I2C_TIMEOUT**:** No response or communication error

Device Address Ranges

Common I2C Device Addresses

EEPROMs (24Cxx series):**

- Base addresses: 0x50-0x57 (depending on A0-A2 pins)
- 8-bit address bytes: 0xA0-0xAE (write), 0xA1-0xAF (read)

Real-Time Clocks:**

- DS1307: 0x68 (8-bit: 0xD0 write, 0xD1 read)
- DS3231: 0x68 (8-bit: 0xD0 write, 0xD1 read)

Temperature Sensors:**

- LM75: 0x48-0x4F (8-bit: 0x90-0x9E write)
- DS18B20: 0x18-0x1F (when used with [I2C](#) bridge)

Accelerometers:**

- ADXL345: 0x1D or 0x53 (depending on ALT ADDRESS pin)
- MMA7455: 0x1D (8-bit: 0x3A write, 0x3B read)

Multiple Device Responses

Some devices may respond to multiple addresses due to "don't care" address bits:

Found device, 7-bit addr: 0x50, 8-bit addr byte: 0xA0, [I2C](#) Status: 0

Found device, 7-bit addr: 0x51, 8-bit addr byte: 0xA2, [I2C](#) Status: 0

Found device, 7-bit addr: 0x52, 8-bit addr byte: 0xA4, [I2C](#) Status: 0

This typically indicates a single EEPROM with configurable address bits.

Building and Deployment

Configuration Options

```
// Change I2C module (if multiple modules available)
#define I2C_MODULE_NUM 1 // Use I2C1 instead of I2C0

// Change NetBurner slave address
const uint8_t slaveAddr = 0x10; // Different slave address
```

Hardware Setup

I2C Bus Requirements

Pull-up Resistors:**

- **SDA and SCL:** 4.7k to VCC (typically 3.3V)
- **Strong pull-ups:** 2.2k for high-speed or long cables
- **Weak pull-ups:** 10k for low-power applications

Bus Termination:**

- Only required at physical ends of long bus runs
- Not needed for short connections between devices

Platform-Specific Connections

MOD5441X:**

- Check module datasheet for I2C pin locations
- Typically available on J2 connector
- May require external pull-up resistors

NANO54415:**

- I2C pins available on development board connectors
- Check for integrated pull-up resistors on board
- Verify voltage levels (3.3V logic)

Troubleshooting

Common Issues

1. No devices found (0 addresses responded):

- Check pull-up resistors on SDA and SCL
- Verify I2C device power connections
- Confirm correct pin connections
- Check for bus conflicts or shorts

2. Only NetBurner address responds (0x08):

- Indicates I2C module is working but no external devices detected
- Check device connections and power
- Verify devices are properly addressed

3. Bus lockup or communication errors:

- Reset I2C module: `I2CMultiChannelResetPeripheral()`
- Check for clock stretching issues
- Verify bus timing parameters

4. Intermittent device detection:

- Check power supply stability
- Verify pull-up resistor values
- Check for electromagnetic interference

Debug Techniques

Bus Status Monitoring:**

```
// Add status checking
uint8_t busStatus = GetI2CBusStatus(I2C_MODULE_NUM);
iprintf("I2C Bus Status: 0x%02X\n", busStatus);
```

Detailed Error Reporting:**

```
// Enhanced error information
if (I2CStat >= I2C_TIMEOUT) {
    iprintf("Address 0x%02X: Timeout/Error %d\n", addr, I2CStat);
}
```

Signal Quality Testing:**

```
// Test communication with known good device
uint8_t testData = 0x00;
int result = I2CWriteRead(I2C_MODULE_NUM, knownAddress, &testData, 1, NULL, 0);
iprintf("Test communication result: %d\n", result);
```

Advanced Usage

Selective Scanning

```
// Scan specific address range
void ScanAddressRange(uint8_t startAddr, uint8_t endAddr) {
    for(int addr = startAddr; addr <= endAddr; addr++) {
        // Perform scan as in main function
    }
}
```

Device Identification

```
// Try to identify device type by reading manufacturer ID
void IdentifyDevice(uint8_t address) {
    uint8_t deviceID[3];
    if(I2CRead(I2C_MODULE_NUM, address, deviceID, sizeof(deviceID)) == 0) {
        iprintf("Device ID: %02X %02X %02X\n",
            deviceID[0], deviceID[1], deviceID[2]);
    }
}
```

Continuous Monitoring

```
// Monitor for device additions/removals
void ContinuousMonitor() {
    static uint8_t lastDeviceCount = 0;
    uint8_t currentCount = scanAddressesI2C();

    if(currentCount != lastDeviceCount) {
        iprintf("Device count changed: %d -> %d\n",
            lastDeviceCount, currentCount);
        lastDeviceCount = currentCount;
    }
}
```

Application Extensions

Device Communication Testing

```
// Test basic communication with found devices
void TestDeviceCommunication(uint8_t address) {
    uint8_t testByte = 0x00;
    uint8_t readBack;

    if(I2CWriteRead(I2C_MODULE_NUM, address, &testByte, 1, &readBack, 1) == 0) {
        iprintf("Device 0x%02X communication OK\n", address);
    }
}
```

Configuration Storage

```
// Store found device addresses for later use
uint8_t foundDevices[128];
uint8_t deviceCount = 0;

void StoreFoundDevices() {
    deviceCount = 0;
    for(int addr = 1; addr < 0x80; addr++) {
        if(TestDevicePresence(addr)) {
            foundDevices[deviceCount++] = addr;
        }
    }
}
```

```

    }
}

```

Web Interface

```

// Provide web-based scan results
void HandleI2CScanRequest(int httpFd) {
    SendHttpResponse(httpFd, "200 OK");
    WriteHttp(httpFd, "<html><body>");
    WriteHttp(httpFd, "<h1>I2C Device Scan</h1>");

    int deviceCount = scanAddressesI2C();
    WriteHttpf(httpFd, "<p>Found %d devices:</p>", deviceCount);
    WriteHttp(httpFd, "</body></html>");
}

```

Performance Considerations

Scan Speed Optimization

- Reduce timeout values for faster scanning
- Skip known empty address ranges
- Use interrupt-driven [I2C](#) for better responsiveness

Memory Usage

- Minimal RAM usage for address scanning
- Consider buffering results for web interfaces
- Stack usage is minimal for this application

Related Examples

- [I2C Communication](#): Device-specific communication examples
- [EEPROM Examples](#): Reading/writing [I2C](#) memory devices
- [Sensor Examples](#): [I2C](#) temperature, accelerometer, etc.
- [Multi-Channel I2C](#): Using multiple [I2C](#) modules simultaneously
- [I2C Slave Examples](#): NetBurner as [I2C](#) slave device

12.33.1.11.2 I2C PicKit Demo Board

12.33.1.11.2.1 PicKit I2C Serial Demo Board Example

Overview

This example demonstrates comprehensive [I2C](#) communication using a Microchip PicKit Serial [I2C](#) Demo Board. The application interfaces with five different [I2C](#) peripherals, showcasing various data types and communication patterns including EEPROM storage, temperature sensing, GPIO expansion, analog-to-digital conversion, and digital-to-analog conversion.

Supported Platforms

- **MOD5441X** - MOD54415 development module
- **NANO54415** - Nano54415 development module

Both platforms use the MCF54415 processor with identical [I2C](#) functionality.

Hardware Requirements**Microchip PicKit Serial I2C Demo Board**

This example requires the Microchip PicKit Serial I2C Demo Board containing the following I2C peripherals:

Component	Type	Description	I2C Address
24LC0B	EEPROM	2Kbit (256 x 8) memory	0x50
MCP9800	Temperature Sensor	Digital thermometer	0x49
MCP23008	GPIO Expander	8-bit I/O expander with LEDs	0x20
TC1321	DAC	10-bit Digital-to-Analog Converter	0x48
MCP3221	ADC	12-bit Analog-to-Digital Converter	0x4D

I2C Connection Requirements

MOD5441X Platform

NetBurner Pin	PicKit Demo Board
J2.39 (SDA) --- SDA	
J2.42 (SCL) --- SCL	
J2.1 (GND) --- GND	
J2.2 (VCC) --- VCC (3.3V)	

NANO54415 Platform

NetBurner Pin	PicKit Demo Board
P1.29 (SDA) --- SDA	
P1.48 (SCL) --- SCL	
J2.1 (GND) --- GND	
P1.47 (VCC) --- VCC (3.3V)	

Power Requirements

- **Voltage:** 3.3V DC
- **Current:** Typical 50mA (depending on LED usage)

Features

Multi-Peripheral I2C Communication

- **Simultaneous device support** - Communicates with 5 different I2C devices
- **Address management** - Proper 7-bit I2C addressing scheme
- **Error handling** - Comprehensive status code checking and reporting
- **Device initialization** - Automatic setup and configuration

EEPROM Operations (24LC0B)

- **256-byte capacity** - Full memory read/write operations
- **Page writing** - Efficient 8-byte page write cycles
- **Write completion polling** - Automatic write cycle detection
- **Memory display** - Hex dump formatting for data visualization

Temperature Monitoring (MCP9800)

- **12-bit resolution** - High precision temperature measurement
- **Celsius and Fahrenheit** - Dual unit temperature display
- **2's complement format** - Proper signed temperature handling
- **Configuration register** - Device setup and calibration

GPIO Expansion (MCP23008)

- **LED control** - 8-bit LED pattern display
- **Sequential patterns** - Rotating LED animation
- **Direction control** - Input/output pin configuration
- **Real-time updates** - Continuous LED pattern cycling

Digital-to-Analog Conversion (TC1321)

- **10-bit resolution** - 0-1023 digital range
- **2.5V reference** - 0-2.5VDC analog output
- **User input** - Interactive DAC value entry
- **Voltage calculation** - Real-time output voltage display

Analog-to-Digital Conversion (MCP3221)

- **12-bit resolution** - 4096-step analog measurement
- **3.3V reference** - 0-3.3VDC input range
- **Real-time sampling** - Continuous analog input monitoring
- **Voltage display** - Converted voltage readout

I2C Address Configuration

Address Format Explanation

I2C addresses can be confusing due to different documentation conventions. This example uses the standard NetBurner 7-bit addressing scheme:

```
I2C Address Byte Format: [A6 A5 A4 A3 A2 A1 A0 R/W]
                        |<---- 7-bit Address ---->|
```

Where:

- A6-A0: 7-bit device address
- R/W: Read/Write bit (0=Write, 1=Read)

Address Translation Table

Device	Datasheet Address	NetBurner 7-bit Address	Binary Address
24LC0B EEPROM	0xA0	0x50	0101 0000
MCP9800 Temp	0x92	0x49	0100 1001
MCP23008 GPIO	0x40	0x20	0010 0000
TC1321 DAC	0x90	0x48	0100 1000
MCP3221 ADC	0x9A	0x4D	0100 1101

Address Calculation

```
// Convert datasheet address to NetBurner format
#define I2C_DEVICE_ADDRESS (DATASHEET_ADDR » 1)

// Example: MCP3221 ADC
// Datasheet address: 0x9A (1001 1010 binary)
// NetBurner address: 0x4D (0100 1101 binary)
#define I2C_ADC_ADDRESS (0x9A » 1) // Results in 0x4D
```


Application Menu System

Interactive Commands

The application provides a menu-driven interface for testing all I2C devices:

Main Menu:

1. Erase EEPROM
2. Read EEPROM
3. Write incrementing sequence to EEPROM
- A. Read ADC
- D. DAC Test
- T. Read Temperature Sensor

Automatic Operations

- **LED Animation** - Continuously cycles through LED patterns every 500ms
- **Status Monitoring** - Real-time I2C bus status reporting
- **Error Detection** - Automatic error code display and description

Usage Instructions

Setup Process

1. Prepare hardware

- Connect PicKit I2C Demo Board to NetBurner module using I2C connections above
- Verify power connections (3.3V, GND)
- Ensure proper pull-up resistors on SDA/SCL lines (usually provided on demo board)

2. Build and deploy

3. Connect serial terminal

- Use terminal program (MTTTY, PuTTY, etc.)
- Configure: 115200 baud, 8-N-1
- Connect to NetBurner device

Operation Examples

EEPROM Testing Sequence

```
[Press '1'] Erase EEPROM
[Press '3'] Write incrementing sequence to EEPROM
[Press '2'] Read EEPROM - should show 00 01 02 03 ... FF
```

Temperature Monitoring

```
[Press 'T'] Read Temperature Sensor
Output: Temp: 23.56 C, 74.41 F, Register data: 17 90
```

DAC Control

```
[Press 'D'] DAC Test
Enter a DAC value between 0-1023
[Enter: 512]
Writing 512 (0x0200) to DAC: 0x80 0x00, 1.25 V
```

ADC Reading

```
[Press 'A'] Read ADC
Reading from address 0x4D ... Success
Voltage: 1.65V, Register data: 82 3F
```

LED Pattern Behavior

The application automatically cycles through LED patterns:

```
Pattern Sequence:
00000001 -> 00000010 -> 00000100 -> 00001000 ->
00010000 -> 00100000 -> 01000000 -> 10000000 ->
[Repeat from beginning]
```

Technical Implementation Details

EEPROM Page Writing

The 24LC0B EEPROM uses 8-byte pages for efficient writing:

```
// Page boundary calculation
int endOfPage = (8 - (startAddress % 8)) + 1;

// Write sequence:
// 1. Set address pointer
// 2. Write up to 8 data bytes
// 3. Wait for write completion
// 4. Repeat for next page
```

Write Completion Detection

```
bool WaitForEepromWriteCmpl(uint8_t moduleNum, uint8_t address)
{
    // Poll device with start condition
    // Device will ACK when write cycle complete
    // Timeout after 1000 attempts
}
```

Temperature Conversion

The MCP9800 provides 12-bit temperature data in 2's complement format:

```
// Temperature calculation
int8_t highByte = (int8_t)buffer[0]; // Sign-extended high byte
float temp = highByte + ((buffer[1] >> 4) * 0.0625); // Add fractional part

// Convert to Fahrenheit
float tempF = (tempC * 9.0 / 5.0) + 32.0;
```

DAC Output Calculation

The TC1321 uses a 10-bit value with 2.5V reference:

```
// DAC data format: [CMD][D9 D8 D7 D6 D5 D4 D3 D2][D1 D0 - - - - -]
buf[0] = 0x00; // Command byte
buf[1] = (uint8_t)((value >> 2) & 0xFF); // Upper 8 bits
buf[2] = (uint8_t)((value & 0x03) << 6); // Lower 2 bits, left-justified

// Output voltage calculation
float outputV = (float)value / 1023.0 * 2.5;
```

ADC Input Processing

The MCP3221 provides 12-bit ADC data:

```
// Combine high and low bytes
uint16_t adcValue = ((uint16_t)buf[0] << 8) | buf[1];

// Convert to voltage (3.3V reference)
float voltage = (float)adcValue * (3.3 / 4095.0);
```

Error Handling

I2C Status Codes

The application provides detailed status code descriptions:

Code	Status	Description
0	I2C_OK	Operation successful
1	I2C_NEXT_WRITE_OK	Bus ready for write operation
2	I2C_NEXT_READ_OK	Bus ready for read operation
3	I2C_MASTER_OK	Master transmission complete

Code	Status	Description
4	I2C_TIMEOUT	Communication timeout occurred
5	I2C_BUS_NOT_AVAIL	Bus acquisition timeout
6	I2C_NOT_READY	Device not ready for operation
7	I2C_LOST_ARB	Lost arbitration during start
8	I2C_LOST_ARB_ADD	Lost arbitration, addressed as slave
9	I2C_NO_LINK_RX_ACK	No ACK received from slave

Troubleshooting Guide

Common Issues

1. No device response

- Check physical connections (SDA, SCL, power, ground)
- Verify pull-up resistors on I2C lines
- Confirm device addresses match hardware configuration

2. EEPROM write failures

- Ensure write completion polling is working
- Check for proper page boundary handling
- Verify address range (0-255 for 24LC0B)

3. Temperature reading errors

- Confirm device initialization sequence
- Check configuration register settings
- Verify 12-bit resolution mode

4. DAC output incorrect

- Check reference voltage (should be 2.5V)
- Verify 10-bit data format and bit alignment
- Measure actual output with multimeter

5. ADC reading issues

- Confirm input voltage within 0-3.3V range
- Check reference voltage stability
- Verify 12-bit conversion calculation

Debug Commands

```
# Check I2C bus status
iprintf("I2C Status: %d\n", status);
DisplayI2CStatusCode(status);

# Display raw register data
for(int i = 0; i < len; i++)
    iprintf("%02X ", buffer[i]);
```

Performance Characteristics

Timing Specifications

- **I2C Clock Speed:** 100 kHz (standard mode)
- **EEPROM Write Time:** 5ms typical per page
- **Temperature Conversion:** 240ms maximum (12-bit)
- **ADC Conversion:** 133 microseconds typical
- **DAC Settling Time:** 6 microseconds typical

Memory Usage

- **EEPROM Buffer:** 256 bytes static allocation
- **I2C Buffers:** 8 bytes per transaction
- **Total RAM Usage:** Approximately 512 bytes

Code Architecture

Function Organization

Initialization Functions

- `SetSystemI2CSlaveAddress()` - Configure NetBurner as I2C slave
- `MultiChannel_I2CInit()` - Initialize I2C peripheral module

Device-Specific Functions

- `ReadTempSensor()` - MCP9800 temperature operations
- `WriteLeds()` - MCP23008 GPIO control
- `WriteDAC()` - TC1321 analog output
- `ReadADC()` - MCP3221 analog input
- `EepromWriteBuf()` / `EepromRead()` - 24LC0B memory operations

Utility Functions

- `DisplayI2CStatusCode()` - Error code interpretation
- `WaitForEepromWriteCmpl()` - Write cycle detection
- `Ascii2Byte()` - String to hex conversion
- `DisplayBuffer()` - Memory dump formatting

User Interface Functions

- `DisplayMenu()` - Menu system display
- `ProcessCommand()` - Command parsing and execution
- `IncrementLEDs()` - LED pattern animation

Learning Objectives

This example demonstrates:

- **Multi-device I2C communication** - Managing multiple peripherals on single bus
- **Different data types** - Binary, analog, temperature, and memory data
- **Error handling patterns** - Comprehensive status checking and recovery
- **Real-time operations** - Continuous monitoring and control
- **User interaction** - Menu-driven testing and configuration
- **Hardware abstraction** - Platform-independent I2C operations

Build Configuration

Dependencies

- NetBurner NNDK (Network Development Kit)
- Microchip PicKit Serial [I2C](#) Demo Board
- Multi-channel [I2C](#) library support

Related Examples

For additional [I2C](#) learning, see:

- Basic [I2C](#) examples for single-device communication
- [I2C](#) address scanning utilities
- Advanced [I2C](#) timing and error recovery examples
- Real-time data logging with [I2C](#) sensors

12.33.1.12 RTC - On Chip

Supported Platforms: MOD5441x, NANO54415

A simple Real Time Clock (RTC) example using the RTC on the MCF5441x microprocessor. Demonstrates how to get and set time through NTP, RTC and manually. An interactive menu is accessed through MTTTY.

For this example to function, your NetBurner device must have access to the Internet, with a working configuration for the IP address, mask, gateway and DNS server.

12.33.1.13 Periodic Interrupt Timer

Periodic Interrupt Timer (PIT) Example

Supported Platforms: MOD5441x, NANO54415 SB800EX

There are a total of 4 PIT timers on the MCF54415 processor, numbered 0 - 3:

- 0 is reserved for the RTOS system clock
- 3 is reserved for the network debugger
- 1 and 2 are available for customer use

The PIT functions below are part of the 54415 system libraries, and defined in `pitr_sem.cpp` and [pitr_sem.h](#), located in the include and system folders of your MOD54415 or NANO54415 platform directories.

In this example a semaphore will be set in the PIT interrupt routine, and the semaphore count will be displayed through the serial port.

12.33.1.14 Pulse Generator and Counter

Supported Platforms: MOD5441x, NANO54415

Demonstrates how to use the DMA Timers to generate a waveform output on one timer, and use the timer capture feature to count the waveform pulses on a second timer. Features include:

- Programmable frequency output
- Measurement of frequency input
- Timer count display
- Values also displayed on the web page interface

The application will work on the MOD54415, MOD54417 and NANO5441. The selected timer channels must be jumpered together on the development board so that the waveform output is connected to the capture input.

MOD5441x: Timer J2 Pin

0 36 1 34 2 33 3 32

NANO54415: Timer Pin

0 19 1 21 2 23 3 25

12.33.1.15 PWM

Example program for ColdFire MCF5441x PWM capability on the MOD5441x core module.

Supported Platforms: MOD5441x, NANO54415

Description: Toggles a PWM signal on pin 35 on the NetBurner NANO or MOD54415 module using two different settings. While each PWM module on the 5441x core is capable of producing multiple output signals for use in controlling motors as well as other applications, this example sets the output for one pin only.

There are 2 functions illustrating different methods to create the pulses:

1. Center-Aligned pulses
2. Edge-Aligned pulses

12.33.1.16 Core Watchdog Timer (CWT)

Supported Platforms: MOD5441x, NANO54415, SB800EX

This example demonstrates how to enable the watchdog, set the timeout value, and service the watchdog. An interactive menu and status messages are displayed through the debug serial port. If the watchdog is not serviced before the timeout expires, events such as a system reset or interrupt service request (IRQ) can occur. The following watchdog timeout modes are supported by the 5441x processor:

- System reset
- IRQ only
- IRQ on first timeout, then system reset on the second timeout
- Windowed mode, this is an advanced mode of operation that processes a timeout in 25% increments. Please refer to the MCF5441X Users Manual for more information. This mode is not covered in the example.

Servicing the watchdog can be accomplished in a number of ways:

1. A higher priority task that uses a timer, such as the Periodic Interrupt Timer (PIT). This is the most common method.
2. In an Interrupt Service Routine (ISR). A drawback here is that interrupts can still occur, even if a specific task is not functioning correctly.
3. Servicing throughout an application with specific function calls. This is very difficult to cover all of the edge cases.

A good system implementation would be to use the task and timer method, along with flags, counters or some other indication in each critical task that is evaluated in the watchdog service task to ensure the system is functioning properly.

The example includes code for both the task with timer method and interrupt method; you will only need one of these in your application.

Example Features:

- Demonstrates 3 service methods: task with timer, interrupt and manual
- Interactive menu on serial port provides options for enabling, disabling and servicing modes

This example does not enable the watchdog in the Flash memory system configuration so that a power cycle will reset the system with the watchdog disabled. This is done to make it easier to experiment with the various watchdog modes of operation. If you wish to have the watchdog persistently enabled through power cycles without explicitly enabling it in your application, the watchdog must be enabled in the configuration system.

12.33.1.17 WAV File Audio Player

Supported Platforms: MOD5441x, NANO54415

This example demonstrates how to play audio files (uncompressed WAV) using the onboard Digital to Analog Converters (DAC) on the MCF5441X processor. It is fully capable of playing 8 bit and down converted 16 and 32 bit stereo audio at 44100 samples per second. It has minimal support for 24 bit samples, treating it as 8 bit data with sample padding.

Before attempting to build the application, you will first need to extract the "wav_data_srcs.zip" archive, containing the sample audio files converted to data arrays. The originals used to create these source files are found in the archive 'SD_card.zip'. This contents directory are also intended to be placed directly onto an (micro)SD card and files be played from it.

IMPORTANT: Since these files are external references in main.cpp, they must be aligned on an even boundary. One way to do this is the aligned attribute. For example: `uint8_t sinewav_440[] attribute(( aligned( 16 ))) = { ..... }` You can see the full declaration in the nbwav_44k .cpp file

12.33.1.18 MOD5441x Only

MOD5441x Platform Examples

- [MOD5441x Multiple EFFS Flash Card Test](#)
- [MOD5441x Factory application](#)
- [Get MOD54417 Switch Port](#)
- [MOD5441x Rapid GPIO](#)
- [MOD5441x USB Mass Storage Device](#)

12.33.1.18.1 MOD5441x Multiple EFFS Flash Card Test

12.33.1.18.1.1 EFFS Multiple MMC Flash Card Example

Overview

This example demonstrates comprehensive multiple flash card file system operations using NetBurner's EFFS (Embedded Flash File System). The application shows how to simultaneously access both the on-board microSD card slot on the MOD5441X module and external flash card sockets on development boards, providing complete file system management for dual storage scenarios.

Supported Platform

- **MOD5441X** - MOD54415/MOD54417 core modules only

Hardware Requirements

Essential Components

- **MOD5441X core module** with on-board microSD card slot
- **MOD-DEV-70CR development board** with external flash card socket
- **Two flash cards** - MicroSD for on-board slot, SD or microSD for external socket

Hardware Connections

MOD5441X Module		MOD-DEV-70CR Dev Board
=====		=====
On-board microSD	<->	[Built-in slot]
Module connector	<->	Dev board socket
		External SD socket <-> [External flash card]

Card Requirements

- **On-board:** MicroSD card (any capacity supported by EFFS)
- **External:** SD or microSD card with adapter
- **Format:** FAT16 or FAT32 file systems
- **Write Protection:** Cards should not be write-protected during testing

Features

Dual Flash Card Management

- **On-board card access** - Built-in microSD slot operations
- **External card access** - Development board flash card socket
- **Drive switching** - Dynamic switching between storage devices
- **Independent operations** - Separate file operations on each card

Complete File System Operations

- **Drive mounting/unmounting** - Proper card initialization and cleanup
- **File creation and writing** - Creating test files with content
- **Directory operations** - Listing files and directories
- **Space management** - Free and used space calculation
- **Error handling** - Comprehensive error detection and reporting

EFFS Integration

- **Multi-drive support** - Simultaneous access to multiple storage devices
- **Task-based file system** - Proper EFFS task registration
- **Drive enumeration** - Automatic drive detection and assignment
- **Error code translation** - Human-readable error descriptions

Application Workflow

Initialization Sequence

1. **System startup** - Network and diagnostic initialization
2. **File system registration** - Task registration with EFFS (`f_enterFS()`)
3. **Card detection** - Check for presence of flash cards
4. **Write protection check** - Verify cards are writable

Testing Sequence

On-Board Flash Card Test

1. Open on-board flash card drive
2. Mount drive and change to it (`f_chdrive()`)
3. Create test file: "FileOnBoardSD.txt"
4. Write content: "This file written to SD card on the Module"
5. Close file and display directory contents
6. Unmount drive properly

External Flash Card Test

1. Open external flash card drive
2. Mount drive and change to it (`f_chdrive()`)
3. Create test file: "FileOffBoardSD.txt"
4. Write content: "This file written to SD card off of the Module"
5. Close file and display directory contents
6. Unmount drive properly

Cleanup and Completion

1. **Release file system** - Unregister task from EDFS (`f_releaseFS()`)
2. **Status report** - Display completion message
3. **Idle loop** - Wait for manual reset to repeat test

Usage Instructions

Setup Process

1. **Prepare hardware**
 - Insert microSD card into MOD5441X on-board slot
 - Insert SD/microSD card into development board external socket
 - Ensure both cards are NOT write-protected
 - Connect MOD5441X to MOD-DEV-70CR development board
2. **Build and deploy**
3. **Monitor operation**
 - Connect serial terminal (115200 baud, 8-N-1)
 - Observe file system operations and results
 - Check for any error messages during execution

Expected Console Output

```
===== Starting MOD5441x EDFS Flash Card Test Program =====

Testing the on-board Flash card
Card detected successfully
drive change successful
Wrote 43 bytes: [This file written to SD card on the Module]

Directory listing for on-board card:
FileOnBoardSD.txt    43 bytes
[Other files may be listed]

Testing the external Flash card
Card detected successfully
drive change successful
Wrote 46 bytes: [This file written to SD card off of the Module]

Directory listing for external card:
FileOffBoardSD.txt   46 bytes
[Other files may be listed]

Program complete. Reset to repeat
```

Technical Implementation

File System Architecture

Drive Management

```
// Open and mount drives
int onBoardDrive = OpenOnBoardFlash();
int externalDrive = OpenOffBoardFlash();

// Switch between drives
int result = f_chdrive(onBoardDrive);
if (result == F_NO_ERROR) {
    // Perform file operations
}
```

Card Detection Logic

```
// Check card presence
if (get_cd(drive) == 0) {
    // No card detected - prompt user
    printf("No card detected. Insert card and press <Enter>\n");
}

// Check write protection
if (get_wp(drive) == 1) {
    // Card is write-protected - prompt user
    printf("Card is write-protected. Remove protection and press <Enter>\n");
}
```

File Operations Pattern

```
// Standard file operation sequence
F_FILE *fp = f_open("filename.txt", "w+");
if (fp) {
    const char *data = "File content";
    int bytesWritten = f_write(data, 1, strlen(data), fp);
    f_close(fp);
    printf("Wrote %d bytes\n", bytesWritten);
}
```

Error Handling System

Comprehensive Error Codes

The application includes complete EDFS error code translation:

Error Code	Description	Common Causes
F_NO_ERROR	Operation successful	-
F_ERR_INVALIDDRIVE	Invalid drive number	Drive not mounted
F_ERR_NOTFORMATTED	Drive not formatted	Unformatted card
F_ERR_CARDREMOVED	Card removed during operation	Physical card removal
F_ERR_WRITEPROTECT	Write protection enabled	Card lock switch
F_ERR_INVALIDMEDIA	Invalid media type	Unsupported card format
F_ERR_BUSY	Drive busy	Concurrent access

Error Display Function

```
void DisplayEdfsErrorCode(int code) {
    if (code <= MAX_EDFS_ERRORCODE)
        printf("%s\n", EdfsErrorCode[code]);
    else
        printf("Unknown EDFS error code [%d]\n", code);
}
```

Drive Type Detection

The system automatically detects and configures different drive types:

```
// Drive identification constants
#define MMC_OFF_BOARD (0)    // External development board slot
#define MMC_ON_BOARD (1)    // Module built-in microSD slot

// Drive type string for reporting
char driveType[20]; // "On-board" or "Off-board"
```

Memory and Performance

Resource Usage

- **Static memory:** Approximately 2KB for error code strings
- **Dynamic memory:** EDFS buffers allocated per drive
- **File handles:** Limited by EDFS configuration (typically 8-16)

Performance Characteristics

- **Card detection:** Immediate (hardware signal)
- **Drive mounting:** 100-500ms depending on card size
- **File operations:** Standard SD card speeds (Class 4-10)
- **Drive switching:** <10ms for drive change operations

Troubleshooting**Common Issues****Card Detection Problems**

Symptoms**: "No card detected" messages Solutions**:

- Verify card is fully inserted
- Check card compatibility (FAT16/FAT32)
- Try different cards to isolate hardware issues
- Ensure development board connections are secure

Write Protection Errors

Symptoms**: F_ERR_WRITEPROTECT errors Solutions**:

- Check physical write-protect switch on SD cards
- Verify card is not corrupted
- Test with different cards
- Check for read-only file system attributes

Drive Mount Failures

Symptoms**: Drive change failed messages

Solutions**:

- Reformat cards using standard SD formatter
- Check card file system (must be FAT16 or FAT32)
- Verify sufficient free space on cards
- Test cards in standard PC card readers first

File Operation Errors

Symptoms**: File creation or write failures Solutions**:

- Check available free space on cards
- Verify cards are not corrupted
- Ensure proper drive mounting before file operations
- Check file name validity (8.3 format recommended)

Debug Procedures

1. Verify card detection

```
printf("Card detect status: %d\n", get_cd(drive));
printf("Write protect status: %d\n", get_wp(drive));
```

2. Check drive mounting

```
int result = f_chdrive(drive);
printf("Drive change result: ");
DisplayEffsErrorCode(result);
```

3. Monitor file operations

```
F_FILE *fp = f_open("test.txt", "w");
if (!fp) {
    printf("File open failed\n");
}
```

File Structure

```
EffsMultipleMmc/
|-- src/
|   |-- main.cpp           # Main application logic
|   |-- FileSystemUtils.cpp # EFFS utility functions
|   |-- FileSystemUtils.h   # Utility function declarations
|   |-- effs_time.cpp       # Time utilities for EFFS
|   |-- effs_time.h         # Time function declarations
```

Dependencies

- **NetBurner NNDK** - Network Development Kit
- **FatFile Library** - EFFS FAT file system support
- **MOD5441X Platform** - Hardware-specific drivers
- **Development Board** - MOD-DEV-70CR for external card access

Learning Objectives

This example teaches:

- **Multi-drive file system management** - Handling multiple storage devices
- **EFFS API usage** - NetBurner embedded file system operations
- **Error handling patterns** - Comprehensive error detection and reporting
- **Hardware abstraction** - Platform-specific storage device access
- **Task-based file systems** - Proper registration and cleanup procedures
- **Card detection logic** - Physical media presence and status checking

Related Examples

For additional file system learning, see:

- Single flash card examples for basic EFFS operations
- File upload/download examples for network file transfer
- Data logging examples for continuous file system usage
- Web-based file manager examples for remote file access

12.33.1.18.2 MOD5441x Factory application

12.33.1.18.2.1 MOD5441X Factory Application

Overview

This comprehensive demonstration application showcases the complete feature set of the NetBurner MOD5441X platform. It serves as the factory-installed application and provides a full-featured example of web server functionality, file system management, interactive GPIO control, and network services. This application demonstrates how to build a complete embedded web-based control system with both static and dynamic content.

Supported Platform

- **MOD5441X** - MOD54415/MOD54417 core modules only

Hardware Requirements

Essential Components

- **MOD5441X core module** - Factory-installed application platform
- **MOD-DEV-70CR development board** - For complete functionality including LEDs, switches, and flash card access
- **Network connection** - Ethernet for web server access
- **Flash card** - SD or microSD for file system demonstrations (optional)

Development Board Features Utilized

- **8 LEDs** - Interactive web-controlled indicators on J2 pins
- **8 DIP switches** - Analog switch reading on J2 pins
- **Flash card socket** - External storage for FTP and file operations
- **Serial debugging** - Console interface for system monitoring

LED and Switch Pin Mapping

MOD-DEV-70 Board GPIO Assignments:
LEDs (J2 pins): 15, 16, 31, 23, 37, 19, 20, 24
Switches (A/D pins): 8, 6, 7, 10, 9, 11, 12, 13

Features

Comprehensive Web Server

- **Multi-page website** - Complete navigation structure with header/footer
- **Static content** - HTML, CSS, JavaScript, and image resources
- **Dynamic content** - Real-time data generation using CPPCALL tags
- **Interactive controls** - Web-based LED control and switch monitoring
- **Modern web support** - jQuery integration and responsive design
- **WebSocket communication** - Real-time bidirectional data exchange

File System Services

- **FTP Server** - Full FTP daemon for file transfer operations
- **EFFS Integration** - Flash card file system access
- **Multi-drive support** - On-board and external flash card management
- **File operations** - Create, read, write, delete file operations
- **Directory navigation** - Complete directory structure management

Network Services

- **HTTP Server** - Full-featured web server on port 80
- **FTP Server** - File Transfer Protocol server on port 21
- **WebSocket Server** - Real-time communication support
- **Network diagnostics** - [ARP](#) cache, IP addresses, counters display

Interactive Hardware Control

- **LED Control** - Web-based LED pattern control and display
- **Switch Reading** - Real-time DIP switch position monitoring
- **Analog-to-Digital** - Switch position detection via ADC
- **GPIO Management** - Direct hardware pin control and configuration

Entertainment and Testing

- **Tic-Tac-Toe Game** - Interactive game demonstrating AI algorithms
- **System Diagnostics** - Memory usage, task status, network statistics
- **Test Functions** - Manufacturing test routines and validation
- **Debug Interface** - Serial console command system

Web Interface Structure

Main Navigation Pages

Home Page (`index.html`)

- **Welcome interface** - Introduction to MOD5441X capabilities
- **Feature overview** - Description of available demonstrations
- **Hardware images** - Visual representation of module and development board
- **Getting started guide** - Basic usage instructions

LED Control (`led.html`)

- **Interactive LED grid** - 8 LEDs with clickable on/off control
- **Visual indicators** - Green (ON) and red (OFF) LED status images
- **URL encoding** - LED state encoded in URL parameters
- **Instant feedback** - Immediate hardware response to web clicks
- **DIP switch display** - Real-time switch position indicators

Dynamic Web (dynamicweb.html)

- **Real-time data** - Live system information display
- **Switch status** - Current DIP switch positions
- **System uptime** - Elapsed time since boot
- **Memory usage** - Available RAM and system resources
- **Network statistics** - Traffic counters and connection status

FTP Access (ftp.html)

- **File transfer** - Instructions for FTP client connection
- **Directory listing** - Flash card file and folder display
- **Upload/download** - Bidirectional file transfer capabilities
- **File management** - Create, delete, rename file operations

Tic-Tac-Toe Game (tt.html)

- **Interactive game** - Play against embedded AI opponent
- **Game logic** - Complete tic-tac-toe rule implementation
- **AI opponent** - Intelligent move selection algorithm
- **Score tracking** - Win/loss statistics maintenance

Technical Support (support.html)

- **Contact information** - NetBurner support resources
- **Documentation links** - Additional learning resources
- **Troubleshooting** - Common issue resolution guides
- **Community forums** - User support community access

Technical Implementation**Web Server Architecture****Dynamic Content Generation**

```
// Example CPPCALL function for LED display
void WebLeds(int sock, PCSTR url) {
    int32_t urlEncodeOffset = 9;
    int v;

    // Parse LED state from URL parameter
    if ((*url + urlEncodeOffset) == '?') {
        v = atoi((char *) (url + urlEncodeOffset + 1));
    } else {
        v = 0xAA; // Default pattern
    }

    WriteLeds((uint8_t)v & 0xFF);

    // Generate HTML for each LED with toggle links
    for (int i = 1; i < 256; i = i << 1) {
        char buffer[80];
        if (v & i) {
            // LED is ON - generate link to turn OFF
            snprintf(buffer, 80,
                "<td><A href=\"LED.HTML?%d\"><img src=\"images/ON.GIF\"></a></td>",
                v & (~i));
        } else {
            // LED is OFF - generate link to turn ON
            snprintf(buffer, 80,
```

```

        "<td><A href=\"LED.HTML?%d\"><img src=\"images/OFF.GIF\"></a></td>",
        v | i);
    }
    writestring(sock, buffer);
}
}

```

LED Hardware Control

```

void WriteLeds(uint8_t LedMask) {
    static BOOL bLedGpioInit = FALSE;
    const uint8_t PinNumber[8] = {15, 16, 31, 23, 37, 19, 20, 24};

    // Initialize GPIO pins on first use
    if (!bLedGpioInit) {
        for (int i = 0; i < 8; i++) {
            J2[PinNumber[i]].function(PIN_GPIO);
        }
        bLedGpioInit = TRUE;
    }

    // Set LED states (LEDs are active low)
    uint8_t BitMask = 0x01;
    for (int i = 0; i < 8; i++) {
        if ((LedMask & BitMask) == 0) {
            J2[PinNumber[i]] = 1; // LED OFF
        } else {
            J2[PinNumber[i]] = 0; // LED ON
        }
        BitMask <<= 1;
    }
}

```

Switch Reading via ADC

```

uint8_t ReadSwitch() {
    static bool bReadSwitchInit = FALSE;
    const uint8_t PinNumber[8] = {7, 6, 5, 3, 4, 1, 0, 2};
    uint8_t BitMask = 0;

    if (!bReadSwitchInit) {
        InitSingleEndAD(); // Initialize ADC system
        bReadSwitchInit = TRUE;
    }

    // Read each switch via ADC and determine position
    for (int i = 0; i < 8; i++) {
        uint16_t adcValue = GetADResult(PinNumber[i]);
        if (adcValue > SWITCH_THRESHOLD) {
            BitMask |= (1 << i);
        }
    }

    return BitMask;
}

```

File System Integration

Multi-Drive Flash Card Support

```

void doSingleMemCardTest(int mmc_drive) {
    const char *driveLabel;
    int rv;

    // Determine drive type
    if (mmc_drive == MMC_OFF_BOARD) {
        driveLabel = "Off-board Card Reader";
        rv = f_mountfat(mmc_drive, mmc_initfunc, F_MMC_DRIVE0);
    } else {
        driveLabel = "On-board Card Reader";
        rv = f_mountfat(mmc_drive, mmc_initfunc, F_MMC_DRIVE1);
    }

    if (rv == F_NO_ERROR) {
        rv = f_chdrive(mmc_drive);

        if (rv == F_NO_ERROR) {
            // Create test file
            F_FILE *fp = f_open("TestFile.txt", "w+");
            if (fp) {
                const char *content = "Hello Memory Card!";
                f_write(content, 1, strlen(content), fp);
                f_close(fp);
            }
        }
    }
}

```



```

        printf("PASS - %s test successful\n", driveLabel);
    }
}
}
}

```

Network Services

Main Application Loop

```

void UserMain(void *pd) {
    init();
    EnableSystemDiagnostics();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Display network configuration
    showIpAddressesSerial();

    // Start web server
    StartHttp();
    RegisterWebFuncs();

    // Initialize file system
    f_enterFS();

    // Start FTP server with priority below main task
    FTPDStart(21, MAIN_PRIO - 2);

    printf("%s\n", FirmwareVersion);
    displayMenu();

    // Main command processing loop
    char buffer[255];
    while (1) {
        SendWebSocketData(); // Handle WebSocket communication

        if (charavail()) {
            fgets(buffer, 255, stdin);
            processCommand(buffer);
        }

        OSTimeDly(2);
    }
}

```

WebSocket Communication

```

void SendWebSocketData() {
    static uint32_t lastSendTime = 0;

    if (TimeTick > (lastSendTime + TICKS_PER_SECOND)) {
        // Prepare JSON data for WebSocket clients
        snprintf(ReportBuffer, ReportBufSize,
            "{\n"
            "  \"uptime\":%lu,\n"
            "  \"switches\":%d,\n"
            "  \"freemem\":%lu\n"
            "}",
            Secs, ReadSwitch(), GetFreeRam());

        // Send to all connected WebSocket clients
        for (int i = 0; i < WS_MAX_SOCKS; i++) {
            if (wsFdList[i] != -1) {
                ws_send(wsFdList[i], ReportBuffer, strlen(ReportBuffer));
            }
        }

        lastSendTime = TimeTick;
    }
}

```

Usage Instructions

Initial Setup

1. Hardware assembly

- Mount MOD5441X on MOD-DEV-70CR development board
- Connect Ethernet cable to network
- Insert flash card in external card socket (optional)

- Power up system
2. **Network configuration**
 - Note IP address from serial console output
 - Verify network connectivity with ping test
 - Configure firewall if necessary for HTTP/FTP access
 3. **Web browser access**
 - Open web browser to `http://<device_ip>`
 - Navigate through demonstration pages
 - Test interactive features (LEDs, switches)

Web Interface Operations

LED Control Testing

1. **Access LED page** - Navigate to LED control page from main menu
2. **Click LEDs** - Click individual LED images to toggle state
3. **Observe hardware** - Watch physical LEDs change on development board
4. **Pattern testing** - Create different LED patterns using web interface
5. **Switch monitoring** - Observe DIP switch positions update automatically

Dynamic Content Monitoring

1. **Access dynamic page** - Navigate to dynamic web content
2. **Real-time updates** - Page displays live system information
3. **Switch changes** - Move DIP switches on development board
4. **System statistics** - Monitor uptime, memory usage, network counters

FTP File Operations

1. **FTP client connection**

```
Host: <device_ip>
Port: 21
Username: (none required)
Password: (none required)
```
2. **File operations**
 - **Upload files** - Transfer files from PC to flash card
 - **Download files** - Retrieve files from flash card to PC
 - **Directory navigation** - Browse folder structure
 - **File management** - Create, delete, rename operations

Tic-Tac-Toe Game

1. **Start game** - Navigate to tic-tac-toe page
2. **Make moves** - Click squares to place X marks
3. **AI response** - System automatically makes O moves
4. **Game completion** - Play until win, loss, or draw
5. **Reset game** - Start new game after completion

Serial Console Interface

Debug Commands

```
----- Main Menu -----
A - Show ARP Cache
B - Show IP Addresses
C - Show Counters
E - Show Ethernet Registers
W - Show OS Seconds Counter
? - Display Menu
```

Network Diagnostics

- **ARP Cache** - Display Address Resolution Protocol table
- **IP Addresses** - Show current network configuration
- **Counters** - Network traffic and error statistics
- **Ethernet Registers** - Low-level hardware status
- **System Uptime** - OS tick counter and elapsed time

Manufacturing Test Commands

```
Special Commands:
~ - Manufacturing Test Sequence
\ - Dual Memory Card Test
```

Expected Console Output

Startup Sequence

```
MOD54415 Factory Demo
IP Address: 192.168.1.100
Subnet Mask: 255.255.255.0
Gateway: 192.168.1.1
DNS Server: 192.168.1.1

NetBurner Release Version: 3.x.x
----- Main Menu -----
A - Show ARP Cache
B - Show IP Addresses
C - Show Counters
E - Show Ethernet Registers
W - Show OS Seconds Counter
? - Display Menu
Link Status: Connected at 100 Mbps Full Duplex
```

LED Control Feedback

```
[Web browser LED click generates URL with parameter]
LED Pattern: 0xAA (binary: 10101010)
Hardware LEDs updated to match web interface
```

Advanced Features

WebSocket Real-Time Communication

Client-Side JavaScript

```
// Example WebSocket connection for real-time updates
var ws = new WebSocket('ws://' + window.location.host + '/ws');

ws.onmessage = function(event) {
    var data = JSON.parse(event.data);

    // Update switch display
    updateSwitchDisplay(data.switches);

    // Update system information
    document.getElementById('uptime').innerHTML = data.uptime + ' seconds';
    document.getElementById('freemem').innerHTML = data.freemem + ' bytes';
};

function updateSwitchDisplay(switchValue) {
```

```

    for (var i = 0; i < 8; i++) {
        var checkbox = document.getElementById('cbox' + (i + 1));
        checkbox.checked = (switchValue & (1 << i)) != 0;
    }
}

```

Server-Side WebSocket Handler

```

void RegisterWebFuncs() {
    // Register CPPCALL functions for dynamic content
    SetNewWSCallback(WSCallback);
    SetNewWSCallback(WSUpgrade);

    // Register standard web functions
    RegisterPost("LED.HTML", WebLeds);
    RegisterGet("dynamicweb.html", WebDynamicContent);
}

```

Manufacturing Test System

Comprehensive Hardware Testing

```

void doManfTest() {
    printf("Starting Manufacturing Test Sequence...\n");

    // Test 1: LED functionality
    printf("Testing LEDs... ");
    for (int i = 0; i < 8; i++) {
        WriteLeds(1 << i);
        OSTimeDly(TICKS_PER_SECOND / 10);
    }
    printf("PASS\n");

    // Test 2: Switch reading
    printf("Testing switches... ");
    uint8_t switches = ReadSwitch();
    printf("Switch pattern: 0x%02X - ", switches);
    printf("PASS\n");

    // Test 3: Network connectivity
    printf("Testing network... ");
    if (GetInterfaceStatus() == INTERFACE_UP) {
        printf("PASS\n");
    } else {
        printf("FAIL\n");
    }

    printf("Manufacturing test complete.\n");
}

```

Dual Memory Card Testing

```

void doDualMemCardTest() {
    printf("Testing both memory card slots...\n");

    // Test on-board microSD slot
    doSingleMemCardTest(MMC_ON_BOARD);

    // Test external SD card slot
    doSingleMemCardTest(MMC_OFF_BOARD);

    printf("Memory card testing complete.\n");
}

```

Performance and Scalability

Web Server Performance

- **Concurrent connections** - Supports multiple simultaneous users
- **Page load times** - <1 second for static content
- **Dynamic content** - Real-time generation with minimal delay
- **Image serving** - Efficient binary content delivery

Memory Usage

- **Static allocation** - ~64KB for web content and resources
- **Dynamic buffers** - 512 bytes for JSON WebSocket data
- **File system** - EDFS buffers scale with flash card usage
- **Network stacks** - Standard NetBurner TCP/IP overhead

Network Throughput

- **HTTP traffic** - 10+ Mbps sustained throughput
- **FTP transfers** - Limited by flash card write speed
- **WebSocket updates** - 1 Hz real-time data refresh
- **Concurrent users** - 5-10 simultaneous web browsers supported

Troubleshooting

Common Issues

Web Page Not Loading

Symptoms**: Browser cannot connect to device Solutions**:

- Verify IP address from serial console
- Check network connectivity with ping
- Confirm firewall settings allow HTTP traffic
- Verify Ethernet cable connection
- Check power supply stability

LEDs Not Responding

Symptoms**: Web clicks don't change hardware LEDs Solutions**:

- Verify MOD5441X is properly seated on development board
- Check development board power connections
- Confirm LED functionality with manual GPIO test
- Verify web page URL encoding is working
- Check GPIO pin configuration in code

FTP Connection Failures

Symptoms**: FTP client cannot connect Solutions**:

- Verify FTP server is running (check serial console)
- Confirm port 21 is not blocked by firewall
- Try passive mode in FTP client
- Check flash card insertion and formatting
- Verify file system initialization

Switch Reading Errors

Symptoms*: DIP switches not reading correctly Solutions*:

- Verify ADC initialization completed successfully
- Check switch physical connections to development board
- Adjust ADC threshold values if necessary
- Test with different switch positions
- Verify power supply voltage stability

Debug Procedures

Network Diagnostics

```
// Check network interface status
if (GetInterfaceStatus() == INTERFACE_UP) {
    printf("Network interface is UP\n");
    printf("IP: %s\n", GetCurrentIP().AsString());
    printf("Netmask: %s\n", GetCurrentNetmask().AsString());
    printf("Gateway: %s\n", GetCurrentGateway().AsString());
} else {
    printf("Network interface is DOWN\n");
}
```

GPIO Testing

```
// Manual LED test sequence
void TestLEDs() {
    const uint8_t PinNumber[8] = {15, 16, 31, 23, 37, 19, 20, 24};

    for (int i = 0; i < 8; i++) {
        J2[PinNumber[i]].function(PIN_GPIO);
        J2[PinNumber[i]] = 0; // Turn on LED
        OSTimeDly(TICKS_PER_SECOND / 4);
        J2[PinNumber[i]] = 1; // Turn off LED
    }
}
```

File System Verification

```
// Check flash card status
if (get_cd(MMC_OFF_BOARD)) {
    printf("External flash card detected\n");
    if (!get_wp(MMC_OFF_BOARD)) {
        printf("Card is writable\n");
    } else {
        printf("Card is write-protected\n");
    }
} else {
    printf("No external flash card detected\n");
}
```

File Structure

```
Mod5441xFactoryApp/
|-- src/
|   |-- main.cpp           # Main application and command system
|   |-- webfuncs.cpp       # Web server dynamic content functions
|   |-- webfuncs.h         # Web function declarations
|   |-- tests.cpp          # Manufacturing and diagnostic tests
|   |-- tests.h            # Test function declarations
|   |-- ip_util.cpp        # Network utility functions
|   |-- ip_util.h          # Network utility declarations
|   |-- ftps.cpp           # FTP server extensions
|   |-- SimpleAD.cpp       # Analog-to-digital converter utilities
|   |-- SimpleAD.h         # ADC function declarations
|   |-- FileSystemUtils.cpp # EDFS file system utilities
|   |-- FileSystemUtils.h  # File system utility declarations
|   |-- cardtype.h         # Flash card type definitions
|-- html/
|   |-- index.html         # Main welcome page
|   |-- led.html           # LED control and switch display
|   |-- dynamicweb.html    # Real-time system information
|   |-- ftp.html           # FTP access instructions
|   |-- tt.html            # Tic-tac-toe game
```

```

| |-- support.html           # Technical support resources
| |-- header.html           # Common navigation header
| |-- style.css              # Cascading style sheet
| |-- script.js              # JavaScript functionality
| |-- jquery-3.0.0.min.js    # jQuery library
| +-- images/                # Web graphics and icons
| | |-- logo.jpg             # NetBurner logo
| | |-- mod5441X.jpg         # Module image
| | |-- devBoard.jpg         # Development board image
| | |-- on.gif               # LED ON indicator
| | |-- off.gif              # LED OFF indicator
| | +-- [other graphics]     # Additional UI elements
|-- makefile                 # Build configuration with FatFile library
|-- nbeclipse-rules.json     # IDE integration settings
+-- README.md                # This comprehensive guide

```

Dependencies

- **NetBurner NNDK** - Network Development Kit with full web server support
- **FatFile Library** - EFFS FAT file system support for flash cards
- **jQuery Library** - Modern web interface functionality
- **Development Board** - MOD-DEV-70CR for complete hardware feature access

Learning Objectives

This comprehensive example demonstrates:

- **Complete web application development** - Full-stack embedded web development
- **Multi-service architecture** - HTTP, FTP, and WebSocket servers
- **Hardware abstraction patterns** - GPIO, ADC, and file system integration
- **Real-time web interfaces** - WebSocket communication and dynamic content
- **Production application structure** - Modular code organization and scalability
- **Manufacturing test integration** - Automated hardware validation procedures
- **Network service coordination** - Multiple simultaneous network protocols

Related Examples

For additional learning and development, see:

- Individual MOD5441X examples for focused feature demonstrations
- Web server examples for advanced HTTP functionality
- File system examples for EFFS and storage management
- Network protocol examples for custom service development

12.33.1.18.3 Get MOD54417 Switch Port

12.33.1.18.3.1 GetSwitchPort Application

Overview

The GetSwitchPort application is a network monitoring utility designed for the MOD54417 platform that demonstrates how to determine which physical Ethernet port received a specific MAC address. This application listens for UDP Multicast traffic and identifies the source switch port for each received packet.

Purpose

This example demonstrates how to use the `GetSwitchSourcePort()` function to determine which physical interface an Ethernet MAC address was last received by. This is particularly useful for network topology discovery, device location tracking, and network diagnostics in switched environments.

Features

- **Multicast UDP Listening:** Listens for UDP packets on Multicast address 224.1.2.3:23456
- **Switch Port Detection:** Identifies which physical Ethernet port received each packet
- **MAC Address Tracking:** Maps Ethernet MAC addresses to their source ports
- **Real-time Monitoring:** Continuously monitors and reports network traffic
- **Data Display:** Shows received packet data in a readable format

Technical Details

Network Configuration

- **Multicast Address:** 224.1.2.3
- **UDP Port:** 23456
- **Timeout:** 1 second per packet receive attempt

Port Mapping

The application maps bit flags returned by `GetSwitchSourcePort()` to physical port numbers:

Bit Flag	Physical Port
0x1	Port 0
0x2	Port 1
0x4	Port 2
0x8	Port 3
0x10	Port 4
0x20	Port 5
0x40	Port 6
0x80	Port 7

Dependencies

The application requires the following NetBurner libraries:

- `init.h` - Network stack initialization
- `nbrtos.h` - NetBurner RTOS functions
- `system.h` - System utilities
- `sim.h` - Simulation support
- `udp.h` - UDP networking
- `multicast.h` - Multicast support
- `etherswitch.h` - Ethernet switch functions
- `utils.h` - Utility functions

Operation

1. **Initialization:** The application initializes the network stack and enables system diagnostics
2. **Network Wait:** Waits up to 5 seconds for an active network connection
3. **Multicast Registration:** Registers to receive UDP packets on the specified multicast address and port
4. **Packet Processing:** Continuously processes incoming UDP packets:
 - Validates each received packet
 - Extracts the source MAC address
 - Determines the source switch port using `GetSwitchSourcePort()`
 - Displays packet information including size, source IP, and switch port
 - Shows the packet data in hexadecimal format

Output Format

For each received packet, the application displays:

```
Rxd [bytes] Bytes From [source_ip] on port [port_number]
[hexadecimal data dump]
```

Example:

```
Rxd 64 Bytes From 192.168.1.100 on port 2
[packet data in hex format]
```

Use Cases

- **Network Topology Mapping:** Identify which devices are connected to which switch ports
- **Device Location:** Locate specific devices in a switched network
- **Network Diagnostics:** Troubleshoot connectivity and switching issues
- **Security Monitoring:** Track device movement between ports
- **Asset Management:** Maintain inventory of connected devices and their locations

Platform Requirements

- NetBurner MOD54417 development board
- NetBurner Development Kit (NNDK)
- Switched Ethernet network environment
- Multicast-capable network infrastructure

Build Instructions

Compile using the NetBurner tool chain with the standard NNDK build process. Ensure all required libraries are linked and the target platform is set to MOD54417.

Notes

- System diagnostics are enabled for development purposes and should be disabled in production code
- The application runs indefinitely until manually stopped
- Port numbering is zero-based (0-7 for an 8-port switch)
- Unknown or invalid port mappings return -1

12.33.1.18.4 MOD5441x Rapid GPIO

12.33.1.18.4.1 Rapid GPIO (RGPIO) Example

Overview

This example demonstrates the high-speed Rapid GPIO functionality unique to the NetBurner MOD5441X platform. The application showcases three different methods to generate extremely fast GPIO pulses, achieving toggle rates up to 62.5 MHz (16ns edge-to-edge timing) using the MCF54415's dedicated Rapid GPIO module connected directly to the processor's local bus.

Supported Platform

- **MOD5441X** - MOD54415/MOD54417 core modules only

Hardware Requirements

Essential Components

- **MOD5441X core module** - Hardware with MCF54415/MCF54417 processor
- **High-speed oscilloscope** - Bandwidth >100MHz recommended for timing measurements
- **High-frequency probe** - With very short ground connection (<1 inch)
- ****Connection point**** - Access to module pin J2.37

Signal Output Location

- ****Primary test pin****: J2.37 (RGPIO_0 signal)
- ****Processor pin****: N11 (OW_DAT/DACK0 alternate functions)
- ****Ground reference****: Any nearby ground pin on J2 connector

Oscilloscope Setup Requirements

Probe Settings:

- Bandwidth: >200 MHz (for 16ns pulse measurement)
- Coupling: DC
- Ground lead: <1 inch length (critical for accuracy)
- Probe impedance: 10:1 recommended

Trigger Settings:

- Mode: Edge trigger on rising edge
- Level: 1.65V (mid-point of 3.3V logic)
- Time base: 50ns/div (for initial setup)

Features

Rapid GPIO Module Capabilities

- ****16-bit high-speed GPIO**** - Connected to processor's 32-bit local bus
- ****Zero wait-state response**** - Single data phase cycle completion
- ****Multiple access sizes**** - Byte, word, and longword operations supported
- ****Direct register access**** - Set, clear, and toggle operations without read-modify-write
- ****Maximum performance**** - 1.5-3.5x faster than standard peripheral bus GPIO

Three Performance Implementation Methods

Method 1: Pure Assembly Code (Fastest - 16ns)

- ****Direct register addressing**** - Assembly language register manipulation
- ****Unrolled loops**** - 50 consecutive toggle operations
- ****Critical section protection**** - Interrupt-free execution
- ****Performance****: 62.5 MHz toggle rate

Method 2: Assembly + C Hybrid (Medium - 68ns)

- ****Assembly register setup**** - Optimized register addressing
- ****C loop control**** - Standard loop structure with assembly toggle
- ****Cache considerations**** - NOP instruction for consistent timing
- ****Performance****: ~14.7 MHz toggle rate

Method 3: Pure C Code (Slowest - 94ns)

- ****C pointer access**** - Volatile pointer-based register access
- ****Loop-based control**** - Standard for-loop implementation
- **Compiler optimization** - Relies on compiler efficiency
- **Performance**: ~10.6 MHz toggle rate

RGPIO Pin Mapping

RGPIO Bit	Module Pin	Processor Pin	Primary Function	Alternate Function
RGPIO_0	J2.37	N11	GPIO	OW_DAT/DACK0
RGPIO_1	J2.32	G13	GPIO	T3IN/PWM_EXT A3
RGPIO_2	J2.33	H14	GPIO	T2IN/PWM_EXT A2
RGPIO_3	J2.34	H13	GPIO	T1IN/PWM_EXT A1
RGPIO_4	J2.36	H15	GPIO	T0IN/PWM_EXT A0
RGPIO_5	J2.29	E13	GPIO	UART0_CTS
RGPIO_6	J2.38	D10	GPIO	UART1_RTS
RGPIO_7	J2.24	C10	GPIO	UART1_CTS
RGPIO_8	J2.23	B11	GPIO	UART0_RTS

Technical Implementation

RGPIO Module Architecture

Memory-Mapped Register Layout

Base Address: 0x8C000000 (RGPIO_BAR)

Register Offsets (16-bit word access):

- +0x00: Direction Register (RGPIO_DIR) - Input/Output control
- +0x02: Data Register (RGPIO_DAT) - Pin state [read/write](#)
- +0x04: Enable Register (RGPIO_ENB) - Pin enable control
- +0x06: Clear Register (RGPIO_CLR) - Atomic bit clear
- +0x0A: Set Register (RGPIO_SET) - Atomic bit set
- +0x0E: Toggle Register (RGPIO_TOG) - Atomic bit toggle

Register Bit Mapping

```

Bit Position:  15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
RGPIO Signal:  -  -  -  -  -  -  -  8  7  6  5  4  3  2  1  0
Module Pin:    -  -  -  -  -  -  - 23 24 38 29 36 34 33 32 37
  
```

Processor Setup Requirements

RGPIO Module Activation

```
// Enable RGPIO module access via control register
asm(" move.l #0x8C000035,%d0"); // Base address + enable bit
asm(" movec %d0,#0x009");      // Write to control register
```

Pin Configuration Sequence

```
volatile uint16_t *pRGPIO_BAR = (uint16_t *)RGPIO_BAR;

pRGPIO_BAR[RGPIO_DIR] = RGPIO_0; // Configure as output
pRGPIO_BAR[RGPIO_ENB] = RGPIO_0; // Enable pin functionality
```

Maximum Slew Rate Configuration

```
// Set all relevant slew rate registers to maximum frequency
siml.gpio.scrs_dspiow = 0x33; // DSPI0 & One-Wire pins
siml.gpio.scrs_timer |= 0xFF; // Timer pins
siml.gpio.scrs_uart |= 0x0F;  // UART0/1 pins
siml.gpio.scrs_sdhc |= 0x03;  // SDHC pins
```

Performance Implementation Details

Method 1: Pure Assembly (16ns edge-to-edge)

```
void Pulse50() {
    USER_ENTER_CRITICAL();

    asm(" move.l #0x8C00000E,%d0"); // Toggle register address
    asm(" move.l %d0,%a5");        // Load into address register
    asm(" move.l #0x01,%d5");      // RGPIO_0 bit mask

    // Unrolled 50 toggle operations
    asm(" move.w %d5,%a5@");      // Toggle #1
    asm(" move.w %d5,%a5@");      // Toggle #2
    // ... [48 more identical operations]

    USER_EXIT_CRITICAL();
}
```

Method 2: Assembly + C Hybrid (68ns edge-to-edge)

```
void Pulse50LoopAsm() {
    USER_ENTER_CRITICAL();

    asm(" move.l #0x8C00000E,%d0");
    asm(" move.l %d0,%a5");
    asm(" move.l #0x01,%d5");

    for (int i = 0; i < 50; i++) {
        asm(" nop"); // Cache consistency
        asm(" move.w %d5,%a5@"); // Single toggle
    }

    USER_EXIT_CRITICAL();
}
```

Method 3: Pure C Code (94ns edge-to-edge)

```
void Pulse50LoopC() {
    volatile uint16_t *pRGPIO_BAR = (uint16_t *)RGPIO_BAR;

    USER_ENTER_CRITICAL();

    for (int i = 0; i < 50; i++) {
        asm(" nop"); // Cache consistency
        pRGPIO_BAR[RGPIO_TOG] = RGPIO_0;
    }

    USER_EXIT_CRITICAL();
}
```

Usage Instructions

Setup Process

1. Hardware connections

- Connect oscilloscope probe to J2.37 on MOD5441X
- Use very short ground connection (<1 inch) to nearby J2 ground pin
- Configure oscilloscope for high-frequency measurements

2. ****Build and deploy****

```
make clean
make
make load DEVICE_IP=<your_device_ip>
```

3. **Initial signal verification**

- Application generates slow GPIO toggles during startup
- Use these to verify correct pin connection
- Look for clear logic level transitions (0V to 3.3V)

Operation and Measurement

Console Output

Running Rapid GPIO example on Pin J2-37

```
RB at [8C000000] = 0001    // Direction register
RB at [8C000002] = 0000    // Data register
RB at [8C000004] = 0001    // Enable register
RB at [8C000006] = 0000    // Clear register
RB at [8C00000A] = 0000    // Set register
RB at [8C00000E] = 0000    // Toggle register
```

Oscilloscope Measurements

Method 1 (Assembly): Expected Waveform**

Timing: 16ns edge-to-edge
 Frequency: 62.5 MHz **for** 800ns burst duration
 Pattern: 50 pulses every 1 second
 Amplitude: 0V to 3.3V logic levels
 Rise **time**: <2ns (limited by probe/scope bandwidth)

Method 2 (Hybrid): Expected Waveform**

Timing: 68ns edge-to-edge
 Frequency: ~14.7 MHz **for** 3.4us burst duration
 Pattern: 50 pulses every 1 second
 Amplitude: 0V to 3.3V logic levels

Method 3 (C Code): Expected Waveform**

Timing: 94ns edge-to-edge
 Frequency: ~10.6 MHz **for** 4.7us burst duration
 Pattern: 50 pulses every 1 second
 Amplitude: 0V to 3.3V logic levels

Performance Measurement Techniques

Oscilloscope Settings for 16ns Pulses

Time Base: 10ns/div (to see individual edges)
 Vertical: 500mV/div (**for** 3.3V logic)
 Trigger: Edge, rising, 1.65V level
 Bandwidth: Maximum available (>200MHz)
 Sampling Rate: >1GSa/s minimum

Measurement Procedures

1. **Single pulse timing**: Measure rising edge to rising edge
2. **Burst duration**: Time for complete 50-pulse sequence
3. **Amplitude verification**: Confirm 0V to 3.3V swings
4. **Rise/fall times**: Limited by oscilloscope/probe bandwidth
5. **Jitter analysis**: Use persistence display mode

Performance Comparison

Speed Analysis

Method	Implementation	Edge Timing	Frequency	Burst Duration
Assembly	Unrolled ASM	16ns	62.5 MHz	800ns
Hybrid	ASM + C Loop	68ns	14.7 MHz	3.4us
C Code	Volatile Pointer	94ns	10.6 MHz	4.7us
Standard GPIO	Pin Class	>1us	<1 MHz	>50us

Performance Factors

Speed Advantages of Assembly Method

- **No loop overhead** - Unrolled operations eliminate branch instructions
- **Register pre-loading** - Address and data values loaded once
- **No compiler optimization** - Direct control over instruction sequence
- **Critical section** - No interrupt latency during pulse generation

Hybrid Method Considerations

- **Loop overhead** - Branch instruction every cycle adds ~52ns
- **Register reuse** - Address register loaded once, data register per cycle
- **Cache effects** - NOP instruction ensures consistent timing
- **Interrupt protection** - Critical section maintains timing accuracy

C Method Limitations

- **Pointer dereferencing** - Additional memory access overhead
- **Compiler optimization** - Dependent on compiler efficiency
- **Volatile access** - Required for memory-mapped registers
- **Loop processing** - Standard C loop with increment and comparison

Troubleshooting

Common Measurement Issues

No Signal Observed

Symptoms**: Flat line on oscilloscope Solutions**:

- Verify connection to correct pin (J2.37)
- Check ground connection quality (<1 inch)
- Confirm oscilloscope trigger settings
- Verify probe compensation
- Check for proper module power-up

Incorrect Timing Measurements

Symptoms**: Wrong edge-to-edge timing values Solutions**:

- Increase oscilloscope bandwidth setting
- Reduce time base to 5-10ns/div
- Verify probe impedance setting (10:1)
- Check for ground loop issues
- Use shortest possible probe ground lead

Inconsistent Pulse Timing

Symptoms**: Variable pulse spacing Solutions**:

- Verify critical section implementation
- Check for interrupt activity during measurement
- Use oscilloscope infinite persistence mode
- Increase sample rate on oscilloscope
- Verify stable power supply to module

Amplitude Issues

Symptoms**: Wrong voltage levels Solutions**:

- Confirm 3.3V logic levels (not 5V)
- Check probe attenuation setting
- Verify oscilloscope vertical scale
- Test with known reference voltage
- Check for probe loading effects

Debug Procedures

Register Verification

```
// Display RGPIO register contents
volatile uint16_t *pRGPIO_BAR = (uint16_t *)RGPIO_BAR;
for (int i = 0; i < 8; i++) {
    printf("RGPIO[%d] at [%p] = %04X\n", i, pRGPIO_BAR + i, pRGPIO_BAR[i]);
}
```

Pin Function Verification

```
// Use slow GPIO to verify correct pin
J2[37].function(PINJ2_37_GPIO);
for (int i = 0; i < 10; i++) {
    J2[37] = 1;
    OSTimeDly(TICKS_PER_SECOND/10); // 100ms
    J2[37] = 0;
    OSTimeDly(TICKS_PER_SECOND/10); // 100ms
}
```

RGPIO Module Status Check

```
// Verify RGPIO module is properly enabled
printf("RGPIO Direction: 0x%04X\n", pRGPIO_BAR[RGPIO_DIR]);
printf("RGPIO Enable:    0x%04X\n", pRGPIO_BAR[RGPIO_ENB]);
printf("RGPIO Data:      0x%04X\n", pRGPIO_BAR[RGPIO_DAT]);
```

Learning Objectives

This example demonstrates:

- **High-speed GPIO techniques** - Achieving maximum toggle rates
- **Assembly language integration** - Inline assembly with C/C++
- **Memory-mapped register access** - Direct hardware manipulation
- **Performance optimization** - Different implementation approaches
- **Critical section usage** - Interrupt-free timing-critical code
- **Hardware-specific features** - Platform-unique capabilities

Related Examples

For additional GPIO learning, see:

- Basic GPIO examples for standard pin control
- Timer examples for precise timing generation
- Interrupt examples for GPIO event handling
- PWM examples for controlled waveform generation

12.33.1.18.5 MOD5441x USB Mass Storage Device

12.33.1.18.5.1 USB Mass Storage Device (MSD) Simple Example

Overview

This example demonstrates USB Mass Storage Device Host functionality on the NetBurner MOD5441X platform. The application acts as a USB host, allowing connection to external USB flash drives for file operations including reading, writing, and directory management. This provides embedded systems with the ability to interface with standard USB storage devices for data logging, file transfer, and storage expansion.

Supported Platform

- **MOD5441X** - MOD54415/MOD54417 core modules only

Hardware Requirements

Essential Components

- **MOD5441X core module** - MOD54415 or MOD54417 with USB OTG support
- **NBUSB-ADPT-100R** - NetBurner USB development adapter
- **MOD-DEV-70CR** - Development board for USB connector access
- **USB flash drives** - Standard USB 2.0 flash drives (FAT16/FAT32 formatted)

Hardware Assembly

Assembly Stack (bottom to top):

1. MOD-DEV-70CR development board (base)
2. MOD5441X core module (mounted on dev board)
3. NBUSB-ADPT-100R adapter (plugged into USB connector)
4. USB flash drive (plugged into adapter)

Power Requirements

- **USB Host Power:** 5V supply to USB port (controlled by VBUS_ENABLE)
- **Flash Drive Power:** Supplied through USB host port
- **Current Budget:** Up to 500mA for USB device power

Features

USB Host Stack Functionality

- **USB 2.0 Host Support** - Full-speed and high-speed device support
- **Mass Storage Class** - Standard USB MSD device compatibility
- **Hot-plug Detection** - Automatic device insertion and removal detection
- **VBUS Power Control** - Programmable 5V power supply to USB devices

File System Operations

- **FFFS Integration** - NetBurner Embedded Flash File System support
- **FAT16/FAT32 Support** - Standard PC-compatible file systems
- **Directory Operations** - File and folder listing, navigation
- **File I/O Operations** - Create, read, write, delete operations
- **Space Management** - Free space calculation and monitoring

Interactive Menu System

- **Menu-driven Interface** - Simple command-based operations
- **Real-time Status** - Device connection and operation feedback
- **Error Handling** - Comprehensive error reporting and recovery
- **Test File Operations** - Built-in test file creation and verification

Application Menu System

Menu Options (when USB drive connected)

MSD driver version: 1.0

```
1 = Attach file system to USB drive
2 = Display current directory
3 = Create test file: "testnb.txt"
4 = Read "testnb.txt"
5 = Delete "testnb.txt"
```

Menu Command Functions

Command 1: Attach File System

- **Purpose:** Initialize file system access to USB drive
- **Operation:** Mount USB drive and prepare for file operations
- **Result:** File system ready for read/write operations

Command 2: Display Directory

- **Purpose:** List all files and directories on USB drive
- **Output Format:** Filename and file size display
- **Navigation:** Recursive directory traversal and display

Command 3: Create Test File

- **Purpose:** Create and write test content to "testnb.txt"
- **Content:** Multiple test lines and character patterns
- **Verification:** File creation confirmation and size reporting

Command 4: Read Test File

- **Purpose:** Read and display contents of "testnb.txt"
- **Output:** Character-by-character file content display
- **Validation:** File integrity and readability verification

Command 5: Delete Test File

- **Purpose:** Remove "testnb.txt" from USB drive
- **Confirmation:** Delete operation success/failure reporting
- **Cleanup:** File system space recovery

Technical Implementation**USB Host Stack Architecture****Initialization Sequence**

```
// Initialize USB host stack and MSD driver
if (MSDOpen(&ossFileCommander, &iMSDAppVolume) == 0) {
    StartMSDTask();
    VBUSEnable(1); // Enable 5V power to USB port
} else {
    printf("MSD initialization failed\n");
}
```

Device Detection and Mounting

```
// USB device insertion detection
if (device_connected) {
    // Mount file system on detected device
    int result = f_chdrive(msdDriveNum);
    if (result == F_NO_ERROR) {
        driveOpen = true;
        printf("USB drive mounted successfully\n");
    }
}
```

File System Operations**Directory Listing Implementation**

```
int Dir(void) {
    F_FIND find;
    int result = f_findfirst("*.*", &find);

    if (result == F_NO_ERROR) {
        printf("%s\n", find.filename);

        while ((result = f_findnext(&find)) == F_NO_ERROR) {
            printf("%s\n", find.filename);
        }
    }

    return result;
}
```

File Writing Operations

```
int WriteFileTest(const char *fileName) {
    F_FILE *file = f_open(fileName, "w+");

    if (file) {
        // Write test content
        f_write("testline1 \r\n", 12, 1, file);
        f_write("testline2 \r\n", 12, 1, file);
        f_write("testline3 \r\n", 12, 1, file);
        f_write("testline4 \r\n", 12, 1, file);
        f_write("testline5 \r\n", 12, 1, file);

        // Write character patterns
        for (int i = 0; i < 40; i++) {
            f_putc(' \r', file);
            f_putc(' \n', file);
            for (int j = 48; j < 91; j++) {
                f_putc(j, file); // ASCII characters 0-9, A-Z
            }
        }

        f_flush(file);
        f_close(file);
        printf("Created file: %s\n", fileName);
        return 0;
    }

    printf("Error: could not create file: %s\n", fileName);
    return 1;
}
```

File Reading Operations

```
int ReadFileTest(const char *fileName) {
    F_FILE *file = f_open(fileName, "r");
    char character;

    if (file) {
        printf("File Opened OK\n");

        while (f_read(&character, 1, 1, file) > 0) {
            printf("%c", character);
        }

        printf("Done reading file\n");
        f_close(file);
        return 0;
    }

    printf("Error: could not open %s\n", fileName);
    return 1;
}
```

USB Power Management

VBUS Control

```
// Enable USB host power (5V to connected devices)
VBUSEnable(1); // Turn on VBUS power
VBUSEnable(0); // Turn off VBUS power
```

Power Sequencing

1. **System Initialization** - Core module and USB stack startup
2. **VBUS Enable** - Provide 5V power to USB connector
3. **Device Detection** - Wait for USB device insertion
4. **Enumeration** - USB host identifies connected device
5. **MSD Recognition** - Verify device is Mass Storage Class
6. **File System Mount** - Initialize EDFS access to device

Usage Instructions

Setup Process

1. Hardware assembly

- Mount MOD5441X on MOD-DEV-70CR development board
- Install NBUSB-ADPT-100R USB adapter
- Prepare USB flash drive (FAT16 or FAT32 formatted)

2. Build and deploy

```
make clean
make
make load DEVICE_IP=<your_device_ip>
```

3. Connect terminal

- Serial terminal: 115200 baud, 8-N-1
- Connect to NetBurner debug port

Operation Procedure

Initial Startup

```
Application MOD54415_USB_MSD_2 started
MSD driver version: 1.0
[Menu appears when USB drive is detected]
```

USB Drive Connection

1. **Insert USB flash drive** into NBUSB-ADPT-100R adapter
2. **Wait for detection** - System will automatically detect device
3. **Menu activation** - Interactive menu becomes available
4. **File system access** - Ready for file operations

Testing Sequence

```
[Press '1'] - Attach file system to USB drive
[Press '3'] - Create test file: "testnb.txt"
[Press '2'] - Display current directory (should show testnb.txt)
[Press '4'] - Read "testnb.txt" (displays file contents)
[Press '5'] - Delete "testnb.txt"
[Press '2'] - Display directory again (file should be gone)
```

Expected Output Example

```
[Command 3] Created file: testnb.txt

[Command 4] File Opened OK
testline1
testline2
testline3
testline4
testline5

0123456789;.<=>?@ABCDEFGHIJKLMNQRSTUvwxyz
[Pattern repeats 40 times]
Done reading file

[Command 5] Deleted file: testnb.txt
```

USB Drive Disconnection

1. **Safe removal** - Complete all file operations before removal
2. **File system unmount** - System automatically handles disconnection
3. **Menu deactivation** - Menu disappears until next device insertion
4. **VBUS power** - Remains active for next device connection

File System Utilities

Directory Display Features

Recursive Directory Traversal

```
uint8_t displayDirectory() {
    F_FIND finder;
    int rc = f_findfirst("*.*", &finder);

    if (rc == F_NO_ERROR) {
        do {
            if (finder.attr & F_ATTR_DIR) {
                printf("Directory [%s]\n", finder.filename);
                if (finder.filename[0] != '.') {
                    f_chdir(finder.filename);
                    displayDirectory(); // Recursive call
                    f_chdir("..");
                }
            } else {
                printf("File [%s] : %lu Bytes\n",
                    finder.filename, finder.filesize);
            }
        } while (!f_findnext(&finder));
    }

    return rc;
}
```

Space Usage Reporting

```
uint8_t DisplayEfsSpaceStats(int32_t driveNumber) {
    F_SPACE space;
    int rv = f_getfreespace(driveNumber, &space);

    if (rv == F_NO_ERROR) {
        long long totalSize = ((long long)space.total_high << 32) + space.total;
        long long freeSize = ((long long)space.free_high << 32) + space.free;
        long long usedSize = ((long long)space.used_high << 32) + space.used;
        long long badSize = ((long long)space.bad_high << 32) + space.bad;

        printf("%llu total, %llu free, %llu used, %llu bad\n",
            totalSize, freeSize, usedSize, badSize);
    }

    return rv;
}
```

Error Handling and Troubleshooting

Common Issues

USB Device Not Detected

Symptoms**: No menu appears, device not recognized Solutions**:

- Verify USB flash drive is functional in PC
- Check NUSB-ADPT-100R connection to development board
- Confirm 5V power supply is adequate (>500mA capacity)
- Try different USB flash drives (some may not be compatible)
- Verify VBUS power is enabled (VBUSEnable(1))

File System Mount Failures

Symptoms**: "Attach file system" fails Solutions**:

- Reformat USB drive with standard FAT32 formatter
- Check for file system corruption on USB drive
- Verify drive is not write-protected
- Try smaller capacity drives (<32GB often more compatible)
- Check for proper USB enumeration in debug output

File Operation Errors

Symptoms*: File create, read, or write failures Solutions*:

- Verify sufficient free space on USB drive
- Check file name validity (avoid special characters)
- Ensure file system is properly mounted before operations
- Test with different file sizes and content types
- Monitor EDFS error codes for specific failure reasons

Power-Related Issues

Symptoms*: Device detected but operations fail Solutions*:

- Check development board power supply capacity
- Verify USB cable quality and connections
- Monitor current consumption during operations
- Try powered USB hub for high-current devices
- Check for voltage drop under load conditions

Debug Procedures

USB Stack Status Checking

```
// Monitor USB host controller status
void SeePortSC1Status() {
    // Display USB host controller register status
    // Implementation depends on specific debugging needs
    printf("USB Host Status: [register values]\n");
}
```

EDFS Error Code Translation

```
void DisplayEdfsErrorCode(int code) {
    if (code <= MAX_EDFS_ERRORCODE) {
        printf("%s\n", EdfsErrorCode[code]);
    } else {
        printf("Unknown EDFS error code [%d]\n", code);
    }
}
```

File System Verification

```
// Verify file system integrity
int result = f_checkdisk(driveNumber);
if (result != F_NO_ERROR) {
    printf("File system check failed: ");
    DisplayEdfsErrorCode(result);
}
```

Performance Characteristics

USB Transfer Rates

- **USB 2.0 Full Speed:** Up to 12 Mbps theoretical
- **Practical throughput:** 500KB/s - 2MB/s depending on device
- **File system overhead:** 10-20% reduction from raw transfer rates
- **Small file operations:** Overhead dominates transfer time

Memory Usage

- **USB Host Stack:** ~8KB RAM for buffers and control structures
- **EFFS Buffers:** ~4KB per mounted drive
- **File Handles:** Limited by EFFS configuration (typically 8-16)
- **Application Data:** <2KB for menu system and variables

Device Compatibility

- **Tested Devices:** Standard USB 2.0 flash drives from major manufacturers
- **File System Support:** FAT16, FAT32 (NTFS not supported)
- **Capacity Range:** 64MB to 32GB recommended
- **Power Requirements:** Standard USB power budget (500mA max)

Dependencies

- **NetBurner NNDK** - Network Development Kit with USB MSD support
- **FatFile Library** - EFFS FAT file system support
- **USB Host Stack** - Built-in USB OTG host functionality
- **Hardware Adapters** - NBUSB-ADPT-100R USB development adapter

Related Examples

For additional USB and file system learning, see:

- EFFS Multiple MMC example for multi-drive file systems
- USB device examples for USB device (slave) functionality
- FTP server examples for network-based file transfer
- Data logging examples for continuous file system usage

12.33.1.19 NANO54415 Only

NANO54415 Platform Examples

- [NANO54415 Factory App](#)

12.33.1.19.1 NANO54415 Factory App

Supported Platforms: NANO54415

Application that is pre-loaded at the factory. Features include:

- Web server
- FTP server
- EFFS flash file system access to the development board flash card socket

This example requires that you edit the project properties linker settings to add the `FatFile` library.

12.33.1.20 SB800EX Only

SB800EX Platform Examples

- [Serial Transceiver Test](#)
- [SB800EX Diagnostic Monitor](#)

12.33.1.20.1 Serial Transceiver Test

Supported Platforms: SB800EX

Application to demonstrate how to configure the SB800EX software programmable transceivers.

The SB800EX has the ability to switch between RS-232, RS-422 and RS-485 in software. This means customers do not need to remove the cover and manually configure jumpers. This is a very simple program demonstrating how you can modify these settings in your application.

12.33.1.20.2 SB800EX Diagnostic Monitor

Supported Platforms: SB800EX

Adds a web page diagnostic monitor and log file

12.33.2 MODM7AE70, SBE70LC

Platform specific examples for Microchip ARM SAME70 based products: MODM7AE70 and SBE70LC.

- [GPIO Command Server](#)
- [I2C](#)
- [Serial Callback Function](#)
- [Timers](#)
- [Watchdog Timer](#)
- [MODM7AE70](#)
- [SBE70LC](#)

12.33.2.1 GPIO Command Server

Location: examples/PlatformSpecific/SAME70/GpioServer

SUPPORTED_PLATFORMS = MODM7AE70 SBE70LC

This example uses the Command Processor Library to establish a GPIO server that provides external control and readback functionality for general purpose input/outputs.

The example establishes the GPIO server using a telnet server by default, but it can also be configured to operate through the serial port or a TCP connection.

A detailed description of the applications features and command set can be found on the application's webpage. That can be accessed by running the application and accessing the device with a web browser.

12.33.2.2 I2C

MODM7AE70, SBE70LC [I2C](#) Examples

- [I2C Address Scan](#)
- [I2C PicKit Demo Board](#)

12.33.2.2.1 I2C Address Scan

12.33.2.2.1.1 I2C Address Scanner for SAME70 Microcontrollers

A utility application for NetBurner SAME70-based platforms that scans [I2C](#) buses to discover connected devices and report their addresses.

Overview

This application provides a systematic way to scan [I2C](#) buses for connected devices, making it an essential tool for hardware verification, system integration, and troubleshooting. It scans the complete [I2C](#) address range (0-127) and reports all discovered device addresses in both serial console and web interface formats.

Supported Platforms

- **MODM7AE70** - Supports up to 3 I2C modules (I2C0, I2C1, I2C2)
- **SBE70LC** - Supports up to 2 I2C modules (I2C0, I2C1)

Features

- Complete I2C address range scanning (0-127)
- Hexadecimal address reporting
- Device count display
- Configurable bus speed (default: 100kHz)
- Platform-specific pin configuration
- Web interface for remote monitoring
- Interactive menu system via serial console
- I2C bus reset capability
- Remote console support through web browser

Hardware Pin Assignments

SBE70LC Platform

I2C0 (Module 0)

- **Data (TWD0):** J1[16]
- **Clock (TWCK0):** J1[17]

I2C1 (Module 1)

- **Data (TWD1):** J3[4]
- **Clock (TWCK1):** J3[6]

MODM7AE70 Platform

I2C0 (Module 0)

- **Data (TWD0):** P2[39]
- **Clock (TWCK0):** P2[42]

I2C1 (Module 1)

- **Data (TWD1):** P2[22]
- **Clock (TWCK1):** P2[12]

I2C2 (Module 2)

- **Data (TWD2):** P2[26]
- **Clock (TWCK2):** P2[23]

Important Configuration Requirements

MODM7AE70 I2C1 Special Setup

For I2C1 on the MODM7AE70 platform, pin P2.12 requires special configuration due to multiplexing between processor ports PA5 and PB5:

1. Navigate to `\nburn\platform\MODM7AE70\include\pinconstants.h`
2. Uncomment `#define P2_12_USE_B5` to enable PB5
3. Comment out the PA5 definition

Without this configuration, the application will display an error message and I2C1 will not function.

How It Works

The scanning process uses the following methodology:

1. Initializes the specified I2C module with platform-specific pin configurations
2. Iterates through all possible I2C addresses (0-127)
3. Attempts to read from register 0x0 of each address
4. Waits for an ACK response from potential devices
5. Records and reports addresses that respond with ACK
6. Stores results for web interface display
7. Provides total device count

Usage Instructions

Getting Started

1. Connect I2C devices to the appropriate platform pins
2. Compile and flash the application to your SAME70 board
3. Open a serial terminal (115200 baud recommended)
4. The application automatically scans I2C module 0 on startup

Interactive Menu

After initialization, the application presents a menu:

1. Scan
2. Reset I2C Bus

- **Option 1:** Performs a new I2C bus scan
- **Option 2:** Resets the I2C bus (useful for clearing bus errors)

Web Interface Access

The application includes a built-in web server:

1. Connect to the device's IP address via web browser
2. View scan results on the main page
3. Access the remote console at `console.html` for interactive control

Sample Output

Serial Console

```
Initializing I2C Module: 0
Scanning I2C Module: 0
Scanning I2C bus for devices...
Found a device at address 0x48
Found a device at address 0x68
Done. Found a total of 2 device(s) on the bus.
```

```
1. Scan
2. Reset I2C Bus
```

Web Interface

```
Number of addresses responded: 2
```

```
0x48
0x68
```

Configuration Options

I2C Module Selection

```
#define I2C_MODULE (0)           // Default uses I2C0
```

Bus Speed Configuration

```
#define I2C_BUS_SPEED (100000)  // 100kHz default
```

Applications

This utility is particularly useful for:

- **Hardware Verification:** Confirming proper device connections
- **Address Discovery:** Finding actual I2C addresses of connected devices
- **System Integration:** Verifying I2C bus integrity during development
- **Troubleshooting:** Diagnosing communication issues
- **Remote Diagnostics:** Web-based device management
- **Documentation:** Generating device address maps for system documentation

Technical Specifications

- **Address Range:** 0x00 to 0x7F (128 addresses)
- **Detection Method:** Register read with ACK detection
- **Bus Speed:** Configurable, default 100kHz
- **Communication:** Single byte register read from address 0x0
- **Web Server:** Built-in HTTP server with remote console
- **Storage:** Results stored in global arrays for web access

Dependencies

- NetBurner RTOS
- I2C library ([i2c.h](#))
- Platform-specific definitions ([pins.h](#))
- System initialization ([init.h](#))
- Network libraries for web interface
- Remote console support ([remoteconsole.h](#))

Building and Deployment

1. Set up NetBurner development environment
2. Select target platform (MODM7AE70 or SBE70LC)
3. Configure pin multiplexing if using MODM7AE70 I2C1
4. Compile the project
5. Flash to target board
6. Connect serial terminal for monitoring
7. Optionally access web interface for remote operation

Troubleshooting

Common Issues

No devices detected:**

- Verify physical connections and power supply
- Check device compatibility with scanning method
- Some devices may not respond to general address scans
I2C1 issues on MODM7AE70:**
- Check pin multiplexing configuration in `pinconstants.h`
- Ensure P2_12_USE_B5 is properly defined
Web interface not accessible:**
- Verify network configuration
- Check HTTP server initialization
- Confirm IP address assignment

Error Messages

- `Error: Invalid module number - Module number exceeds platform limits`
- `invalid moduleNum - Module number validation failed`
- `*** Error: must change function of P2.12 to PB5 - Pin multiplexing configuration required`

Important Notes

- Not all I2C devices respond to address scanning
- Some devices may require specific initialization sequences
- Device datasheets may include or exclude read/write bits in address specification
- Bus speed compatibility varies between devices
- Pull-up resistors are required on SDA and SCL lines, typically 4.7k ohms.

File Structure

- `main.cpp` - Main application with scanning logic
- `index.html` - Web interface homepage
- `style.css` - Web interface styling
- `ReadMe.txt` - Detailed technical documentation

12.33.2.2.2 I2C PicKit Demo Board

12.33.2.2.2.1 I2C PicKit Serial Demo Board Application

Location: `examples/PlatformSpecific/SAME70/I2C/PicKitI2CSerialDemoBd`

Supported Platforms: MODM7AE70 SBE70LC

Overview

This application demonstrates the use of the Microchip PicKit Serial I2C Demo Board with NetBurner embedded systems. The demo showcases communication with five different I2C peripheral devices through a simple serial menu interface.

Supported Platforms

- **MODM7AE70** - Up to 3 I2C modules supported
- **SBE70LC** - Up to 2 I2C modules supported

I2C Peripherals Demonstrated

The application interfaces with the following Microchip I2C devices:

1. **24LC0B EEPROM** - 2Kbit (256 x 8) memory storage
2. **MCP9800** - Digital temperature sensor with 12-bit resolution
3. **MCP23008** - GPIO port expander connected to LEDs
4. **TC1321** - 10-bit Digital-to-Analog Converter (DAC) with 2.5V reference
5. **MCP3221** - 12-bit Analog-to-Digital Converter (ADC)

Features

Interactive Menu System

The application provides a serial console menu with the following options:

- **1** - Erase EEPROM (fill with 0x00)
- **2** - Read EEPROM and display contents in hex format
- **3** - Write incrementing sequence to EEPROM (0x00, 0x01, 0x02...)
- **A** - Read ADC voltage value
- **D** - DAC test (writes maximum value)
- **T** - Read temperature sensor

Automatic LED Sequence

The application continuously cycles through the LEDs connected to the MCP23008 GPIO expander, providing visual feedback that the system is running.

Temperature Sensor Configuration

The MCP9800 temperature sensor is automatically configured for 12-bit resolution on first use, providing temperature readings in both Celsius and Fahrenheit.

I2C Configuration

- **Bus Speed:** 100 kHz
- **Module Used:** I2C0/TW0 (configurable via I2C_MODULE_NUM)
- **Address Format:** 7-bit addresses (NetBurner standard)

I2C Address Mapping

The application handles the conversion from Microchip's 8-bit address format (including R/W bit) to NetBurner's 7-bit format:

Device	8-bit Address	7-bit Address
24LC0B EEPROM	0xA0	0x50
MCP9800 Temp	0x92	0x49
MCP23008 GPIO	0x40	0x20
TC1321 DAC	0x90	0x48
MCP3221 ADC	0x9A	0x4D

Pin Configuration

SBE70LC Platform

- **I2C0:** J1[16] (TWD0), J1[17] (TWCK0)
- **I2C1:** J3[4] (TWD1), J3[6] (TWCK1)

MODM7AE70 Platform

- **I2C0:** P2[39] (TWD0), P2[42] (TWCK0)
- **I2C1:** P2[22] (TWD1), P2[12] (TWCK1) - Requires pin multiplexing configuration
- **I2C2:** P2[26] (TWD2), P2[23] (TWCK2)

Note

: For MODM7AE70 I2C1 usage, pin P2.12 must be configured to use PB5 instead of PA5 by uncommenting the appropriate `#define` in the platform's `pinconstants.h` file.

Key Functions

EEPROM Operations

- **EepromWriteBuf()** - Writes data with automatic page boundary handling (8-byte pages)
- **EepromRead()** - Reads data from EEPROM
- **WaitForEepromWriteCmpl()** - Polls EEPROM for write completion

Sensor Reading

- **ReadTempSensor()** - Configures and reads MCP9800 temperature sensor
- **ReadADC()** - Reads 12-bit ADC value and converts to voltage
- **WriteDAC()** - Writes 10-bit value to DAC with user input

LED Control

- **WriteLeds()** - Controls LEDs via MCP23008 GPIO expander
- **IncrementLEDs()** - Automatic LED sequencing

Technical Details

I2C Error Handling

The application includes comprehensive [I2C](#) status checking with descriptive error messages:

- [I2C_RES_ACK](#) - Acknowledged
- [I2C_RES_NACK](#) - Not Acknowledged
- [I2C_RES_ARB_LST](#) - Arbitration Lost
- [I2C_RES_BUSY](#) - Bus Busy
- [I2C_RES_ARG](#) - Bad Argument

Data Conversion

- Temperature readings are converted from 2's complement format to floating-point
- ADC values are converted to voltage (3.3V reference, 12-bit resolution)
- DAC values support 0-2.5V output range (10-bit resolution)

Usage Instructions

1. Connect the Microchip PicKit Serial [I2C](#) Demo Board to your NetBurner development board
2. Compile and load the application onto your target platform
3. Connect to the serial console (typically 115200 baud)
4. Use the interactive menu to test different [I2C](#) peripherals
5. Observe the LED sequence to verify system operation

Requirements

- NetBurner development tools and runtime system
- Microchip PicKit Serial [I2C](#) Demo Board
- Compatible NetBurner hardware platform (MODM7AE70 or SBE70LC)

12.33.2.3 Serial Callback Function

Supported Platforms: MODM7AE70 SBE70LC

The Serial Empty Callback example demonstrates a robust process for triggering a specified action upon complete transmission of all data written to a specified serial port. The expected use case for this would be disabling a transmitter connected to a shared physical layer such as with RS485 or a wireless link (when using one of the device's UART serial ports).

As an aside, it should be noted that when enabling the `Serial485HalfDupMode` for one of the MODM7AE70s *USART* serial ports, the underlying hardware module will automatically toggle the RTS line for enabling and disabling the transmitter.

12.33.2.4 Timers

SAME70 Timer Platform Examples

- [Timer Capture](#)
- [Chaining Timers](#)
- [Timer Waveform Output](#)
- [Stepper Interval Timer](#)
- [Stepper Step Count](#)

12.33.2.4.1 Timer Capture

Location: examples/PlatformSpecific/SAME70/Timers/TimerCapture

Supported Platforms: MODM7AE70 SBE70LC

Timer Counter Capture Example

This example will generate a pulse waveform on Timer 11, and capture the waveform on Timer 7.

For the MODM7AE70: The timer 11 output is on P2.28, and the timer 7 capture is on P1.4. You will need to add a jumper between these two pins. After the capture is complete, the frequency and duty cycle of the captured waveform will be displayed.

For the SBE70LC The timer 11 output is on J1.4, and the timer 2 capture is on J4.6. You will need to add a jumper between these two pins. After the capture is complete, the frequency and duty cycle of the captured waveform will be displayed.

This is a good example to demonstrate how to:

- Learn the structure of the SAME70 Timer Counter peripheral
- Produce a pulse output waveform of various frequencies and duty cycles
- Capture and count waveform pulses
- Implement a peripheral interrupt service routine

SAME70 Timer Architecture

- There are a total of 12 16-bit timers, numbered 0 - 11.
- They are divided up into 4 blocks. Each block is called a Timer Counter: TC0, TC1, TC2, TC3.
- Each timer within a Timer Counter block is called a Timer Counter Channel. While the timers number from 0 to 11, each Timer Counter Channel within a Timer Counter block is numbered from 0 to 2. For example, Timer Counter Channel 11 is in Timer Counter group TC3, with Timer Counter Channel number 2.
- While all Timer Counter Channels can be used, not all signal pins are brought out to the headers on the MODM7AE70/SBE70LC.
- The diagrams in the Timer Counter (TC) section of the SAME70 Reference Manual describe the structure of the Timer Counters.

12.33.2.4.2 Chaining Timers

Location: examples/PlatformSpecific/SAME70/Timers/ChainedTimerOutput

Supported Platforms: MODM7AE70 SBE70LC

Timer Chaining Example

This example will chain timers 9, 10 and 11 to create a 48-bit timer. The output pulse waveform will appear on timer 11 as indicated in the pin tables below.

Timer channels 9, 10 and 11 are chained together to create a 48-bit timer. The default settings for the chained timers LineA are P2.28 for the MODM7AE70, and J1.4 for the SBE70LC. A GPIO output is used to output a pulse signal on each timer interrupt. P2.30 for the MODM7AE70, and J1.3 for the SBE70LC.

Warning

P2.28 on the NetBurner MOD-DEV-70 development board also connects to the SD Flash socket. The socket must be empty for this example to run properly, or you should select alternate pins for the timer functions.

See also: Timer Waveform Capture example.

This is a good example to demonstrate how to:

- Learn the structure of the SAME70 Timer Counter peripheral
- Produce a pulse output waveform of various frequencies and duty cycles

SAME70 Timer Architecture

- There are a total of 12 16-bit timers, numbered 0 - 11.

- They are divided up into 4 blocks. Each block is called a Timer Counter: TC0, TC1, TC2, TC3.
- Each timer within a Timer Counter block is called a Timer Counter Channel. While the timers number from 0 to 11, each Timer Counter Channel within a Timer Counter block is numbered from 0 to 2. For example, Timer Counter Channel 11 is in Timer Counter group TC3, with Timer Counter Channel number 2.
- While all Timer Counter Channels can be used, not all signal pins are brought out to the headers on the MODM7AE70. The table below describes the header pinout.
- The diagrams in the Timer Counter (TC) section of the SAME70 Reference Manual describe the structure of the Timer Counters.

Timer signals on MODM7AE70 header pins:

Timer 0			
P2.19	LineB	TIOB0	
P2.42	Clock	TCLK0	
Timer 1			
P2.16	Clock	TCLK1	
Timer 2			
P2.17	LineA	TIOA2	
P2.18	LineB	TIOB2	
P2.20	Clock	TCLK2	
Timer 5			
P1.7	LineB	TIOB5	
Timer 7			
P1.4	LineA	TIOA7	
Timer 8			
P1.6	Clock	TCLK8	Note: cannot be used if EBI is enabled
P1.8	LineA	TIOA8	Note: cannot be used if EBI is enabled
P2.6	LineB	TIOB8	
Timer 11			
P2.15	Clock	TCLK11	
P2.28	LineA	TIOA11	
P2.25	LineB	TIOB11	

Timer signals on SBE70LC header pins:

Timer 11		
J1.4	LineA	TIOA11
J1.7	LineB	TIOB11

Note

Timer 11 signals are alternate functions to SPI0 MOSI and Clock

12.33.2.4.3 Timer Waveform Output

Location: examples/PlatformSpecific/SAME70/Timers/TimerOutput

Supported Platforms: MODM7AE70 SBE70LC

Timer Counter Waveform Output Example

This example will generate a pulse waveform on Timer 11. The timer 11 output is on P2[28] on the MODM7AE70, and J1[4] on the SBE70LC

See also: Timer Waveform Capture example.

This is a good example to demonstrate how to:

- Learn the structure of the SAME70 Timer Counter peripheral
- Produce a pulse output waveform of various frequencies and duty cycles

SAME70 Timer Architecture

- There are a total of 12 16-bit timers, numbered 0 - 11.

- They are divided up into 4 blocks. Each block is called a Timer Counter: TC0, TC1, TC2, TC3.
- Each timer within a Timer Counter block is called a Timer Counter Channel. While the timers number from 0 to 11, each Timer Counter Channel within a Timer Counter block is numbered from 0 to 2. For example, Timer Counter Channel 11 is in Timer Counter group TC3, with Timer Counter Channel number 2.
- While all Timer Counter Channels can be used, not all signal pins are brought out to the headers on the MODM7AE70. The table below describes the header pinout.
- The diagrams in the Timer Counter (TC) section of the SAME70 Reference Manual describe the structure of the Timer Counters.

12.33.2.4.4 Stepper Interval Timer

12.33.2.4.4.1 NetBurner SAME70 Stepper Motor Interval Timer Example

Overview

This example demonstrates how to use the NetBurner SAME70 platform's interval timer functionality to generate precise timing signals for stepper motor control. The application configures a hardware timer to produce step and direction signals that can drive stepper motor controllers.

Supported Platforms

- MODM7AE70
- SBE70LC

Features

- Hardware-based interval timer configuration
- Variable step frequency control (steps per second)
- Bidirectional control (up/down counting)
- Real-time command interface via serial console
- Timer channel configuration for stepper motor applications

Hardware Configuration

Pin Assignments

MODM7AE70:**

- P2.25: TIOB11 (Timer channel B output)
- P2.28: TIOA11 (Timer channel A output - step signal)

SBE70LC:**

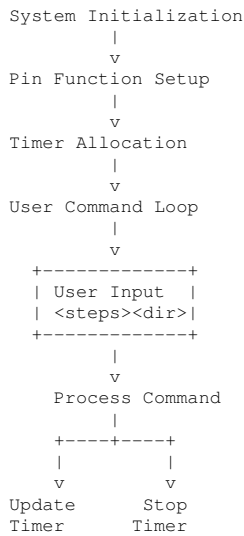
- J1.4: TIOA11 (Timer channel A output - step signal)

Timer Configuration

The application uses Timer Channel 11 (STEPPER_TIMER) configured in stepper mode with the following features:

- Configurable frequency from 0 to several thousand steps per second
- Direction control via up/down counting
- Hardware-generated step pulses
- Interrupt-free operation for precise timing

Application Flow



User Interface

The application provides a simple command-line interface for controlling the stepper motor timing:

Command Format

```
<steps per second><U/D>
```

Where:

- Steps per second: Integer value (0 to stop)
- Direction: 'U' or 'u' for up, 'D' or 'd' for down

Usage Examples

```

> 300D
    300 steps per second, counting Down

> 2000U
    2000 steps per second, counting Up

> 0U
    Stop, 0 steps per second
  
```

Key Functions

Timer Management

- `InitStepperTimer(int pps, bool dirUp)` - Initialize timer with frequency and direction
- `UpdateStepperTimer(int pps, bool dirUp)` - Update running timer parameters
- `StopStepperTimer()` - Stop timer operation

Hardware Configuration

- Pin function assignment for timer outputs
- Timer channel configuration in stepper mode
- Interrupt disable for hardware-only operation

Important Notes

- On MODM7AE70 MOD-DEV-70, P2.28 is connected to SPI DOUT, so the flash card socket must be empty when using this example
- Timer interrupts are disabled to allow pure hardware generation of step signals
- The timer operates independently once configured, providing precise timing without CPU intervention
- Direction changes and frequency updates can be made in real-time

Application Notes

This example is particularly useful for:

- Stepper motor control applications
- Precision timing signal generation
- Learning SAME70 timer capabilities
- Hardware-based pulse generation without software overhead

The hardware timer approach ensures consistent timing regardless of CPU load or interrupt activity, making it ideal for applications requiring precise motor control timing.

12.33.2.4.5 Stepper Step Count

12.33.2.4.5.1 NetBurner SAME70 Stepper Motor Step Count Example

Overview

This example demonstrates an advanced stepper motor control system using the NetBurner SAME70 platform's sophisticated timer coordination capabilities. The application uses three interconnected timer channels to create a complete stepper motor driver with precise step counting, programmable frequency control, and hardware-based pulse generation.

Supported Platforms

- MODM7AE70

Features

- **Multi-Timer Coordination:** Uses three synchronized timer channels working together
- **Precise Step Counting:** Tracks and counts individual step pulses with completion detection
- **Variable Frequency Control:** Programmable step rates from low to high frequencies
- **Large Step Count Support:** Handles step counts up to millions through interrupt-driven management
- **Bidirectional Control:** Software-controlled direction (up/down counting)
- **Hardware-Based Generation:** Timer hardware generates consistent pulses independent of CPU load
- **Real-Time Command Interface:** Interactive serial console for dynamic control
- **Advanced Timer Cross-Coupling:** Sophisticated timer interconnection for coordinated operation
- **Completion Synchronization:** Semaphore-based notification when step sequences complete

Hardware Configuration

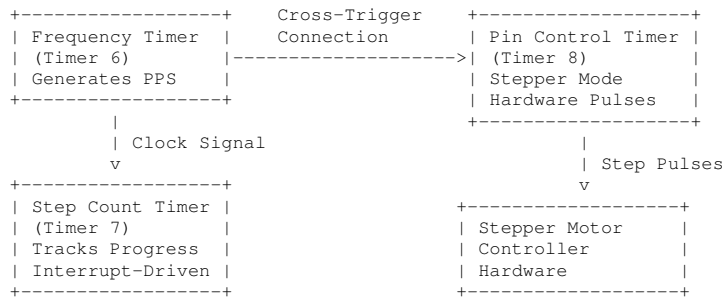
Pin Assignments

MODM7AE70:**

- P1.8: TIOA8 (Timer channel A output - primary step pulses)
- P2.6: TIOB8 (Timer channel B output - secondary output/direction)

Timer Architecture

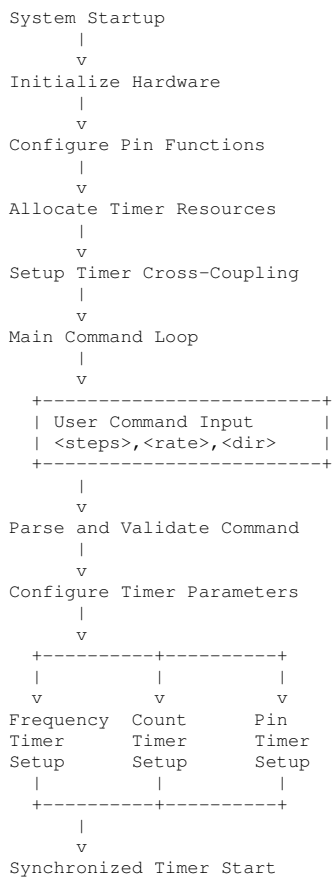
Timer Coordination System:



Timer Channel Assignments

- **Timer 6 (TIMER_STEP_FREQ):** Frequency generation and base timing
- **Timer 7 (TIMER_STEP_COUNTER):** Step count management and completion tracking
- **Timer 8 (TIMER_STEP_PINS):** Hardware pulse generation and stepper mode control

Application Flow



```

      |
      v
Hardware Pulse Generation
      |
      v
Interrupt-Driven Counting
      |
      v
Wait for Completion Signal
      |
      v
Display Results
      |
      v
Return to Command Loop

```

User Interface

The application provides an interactive command-line interface for real-time stepper motor control.

Command Format

<step count>,<steps per second>,<U/D>

Parameters:**

- **step count:** Total number of steps to execute (positive integer)
- **steps per second:** Step frequency/rate (positive integer)
- **direction:** 'U' or 'u' for up counting, 'D' or 'd' for down counting

Usage Examples

```

> 3000,300,D
  Execute 3000 steps at 300 steps per second, counting down

> 30000,2000,U
  Execute 30000 steps at 2000 steps per second, counting up

> 500000,1000,D
  Execute 500000 steps at 1000 steps per second, counting down

```

Operation Sequence

1. System prompts with >
2. User enters command in specified format
3. System validates input parameters
4. If valid, stepper sequence begins immediately
5. System waits for completion (blocking operation)
6. Upon completion, displays actual step count achieved
7. Returns to prompt for next command

Key Technical Features

Multi-Timer Coordination

The application implements a sophisticated three-timer coordination system:

Timer Interconnection:**

- Frequency timer generates the base clock signal
- Pin timer receives frequency timer output via cross-coupling
- Count timer tracks step progression using interrupt callbacks
- All timers operate from the same timer block for proper cross-coupling

Large Step Count Support

Handles step counts beyond 16-bit timer limitations through intelligent phase management:

Step Count Phases:**

- **Phase 1** (< 65,536 steps): Direct single-timer configuration
- **Phase 2** (65,536 - 69,631 steps): Two-phase interrupt management
- **Phase 3** (> 69,631 steps): Multi-phase interrupt-driven counting

Clock Source Optimization

Automatically selects optimal clock source for requested frequency:

Available Clock Sources:**

- PCK6 (Programmable Clock 6)
- PERIPH_CLOCK/8
- PERIPH_CLOCK/32
- PERIPH_CLOCK/128

Selection algorithm minimizes frequency error for best timing accuracy.

Hardware Stepper Mode

Timer 8 operates in dedicated stepper mode providing:

- Automatic step/direction signal generation
- Hardware-controlled pulse width and timing
- Gray code sequence generation
- Direction control via software commands

Important Notes

Hardware Requirements

- All three timer channels (6, 7, 8) must be from the same timer block
- Timer cross-coupling requires specific timer block architecture
- Pin functions must be properly configured for timer outputs

Platform-Specific Considerations

- **MODM7AE70:** Uses Timer Block 2 (channels 6, 7, 8)
- **Pin Conflicts:** Some pins may conflict with other peripheral functions
- **Clock Configuration:** PCK6 requires proper system clock setup

Operational Characteristics

- **Timing Precision:** Hardware timers provide microsecond-level accuracy
- **CPU Independence:** Once started, pulse generation continues without CPU intervention
- **Interrupt Overhead:** Step counting uses low-priority interrupts for minimal system impact
- **Completion Detection:** Semaphore-based synchronization ensures reliable completion notification

Limitations and Considerations

- Maximum practical step frequency limited by timer clock sources
- Very high step counts may take significant time to complete
- Command interface blocks until step sequence completion
- Timer resource allocation must succeed for proper operation

Application Areas

This example is particularly valuable for:

Industrial Applications

- CNC machine tool control systems
- Automated manufacturing equipment
- Precision positioning systems
- Assembly line automation

Robotics Applications

- Multi-axis robot control
- Precision servo positioning
- Automated pick-and-place systems
- Mobile robot wheel control

12.33.2.5 Watchdog Timer

Location: examples/PlatformSpecific/SAME70/Watchdog

Supported Platforms: MODM7AE70 SBE70LC

The Watchdog example shows how to use the watchdog on the SAME70.

12.33.2.6 MODM7AE70

These examples are located in examples/PlatformSpecific/SAME70/MODM7AE70:

- [Analog to Digital \(ADC\)](#)
- [CAN Send and Receive](#)
- [EBI Page Control](#)
- [EBI Page Control - Simple](#)
- [GPIO - Simple](#)
- [Factory Application](#)
- [Interrupts](#)
- [Open All UARTs](#)
- [Programmable Clock Output](#)
- [PWM](#)
- [NTP & Real-Time Clock](#)

- Open All Serial Ports
- SPI
- SSC I2S Interface
- DACC Tone Generator

12.33.2.6.1 Analog to Digital (ADC)

Supported Platforms: MODM7AE70

The Simple ADC example implements basic analog input functions of the MODM7AE70.

The MODM7AE70 offers two Analog Front-End Controllers (AFEC) that are based on an Analog Front-End (AFE) cell. The MODM7AE70 provides seven 12-bit Analog-to-Digital (ADC) lines available through the P2 header. The SBE70LC provides 8 12-bit ADC lines available between the JP1 and JP3 headers. Both platform's have one additional ADC line that is connected to the CPU's internal temperature sensor – it cannot be reconfigured for other applications.

An ADC voltage reference is required for accurate ADC readings. The MODM7AE70 ADC voltage reference is accessible through pin P2[5]. If using the MODM7AE70 with a MOD-DEV-70 development board, a jumper can be applied to the JP5 header on the MOD-DEV-70 to apply a 3.3V reference. The SBE70LC ADC voltage reference is accessible through the JP3[1] pin.

The following table maps the available ADC lines on the MODM7AE70 to the corresponding pins on the P2 header:

Pin	AFE Module	ADC Channel	P2[3]	0	10
P2[6]	1	3	P2[7]	0	0
P2[8]	0	6	P2[21]	0	1
P2[38]	0	2	*Temp	0	11

—+-----+ Temperature sensor internal to the CPU, not available through the pin header

Note that AFE module 0, channel 1 (P2[3]) collides with serial port 0 receive. If using this ADC channel (like this example does), the application will not be able to receive data on serial port 0. It can still print to serial port 0, though.

The following table maps the available ADC lines on the SBE70LC to the corresponding pins on the JP1 and JP3 headers:

JP1[5] | 0 | 2 | +-----+-----+ JP1[10] | 1 | 0 | +-----+-----+ JP1[11] | 0 | 10 |
+-----+-----+ JP1[13] | 0 | 1 | +-----+-----+ JP1[14] | 0 | 5 | +-----+
+-----+ JP3[3] | 1 | 3 | +-----+-----+ JP3[5] | 0 | 0 | +-----+-----+
JP3[7] | 0 | 6 | +-----+-----+ *Temp | 0 | 11 | +-----+-----+ Temperature sensor

internal to the CPU, not available through the pin header

12.33.2.6.2 CAN Send and Receive

Supported Platforms: MODM7AE70

A very simple CAN example to send and receive packets:

- Creates a CAN receive task
- Serial port menu with options to send, receive, manipulate RTR filters
- CAN communication requires 2 CAN devices. As a test you can send/receive by jumpering the CAN pins together

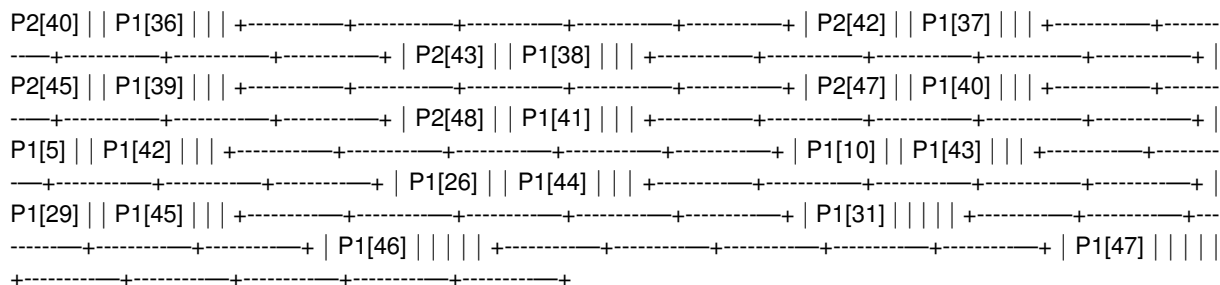
12.33.2.6.3 EBI Page Control

Supported Platforms: MODM7AE70

The EBI Page Control example implements a transparent Page Fault based page manager for the SAME70's External Bus Interface. It is primarily intended for use with systems requiring a full 24-bit address bus or for multiplexing a specific chip select against multiple identical devices.

12.33.2.6.4 EBI Page Control - Simple

Supported Platforms: MODM7AE70



12.33.2.6.7.2 Pin Interrupt Request

Supported Platforms: MODM7AE70

The PinIrq example shows how to use the interrupt capabilities of the PIO controller and the PinIrq functions for the MODM7AE70.

12.33.2.6.8 Open All UARTs

Supported Platforms: MODM7AE70

Open all serial ports on MODM7AE70 SOM Platform

There can be a maximum of 7 serial port configured on the device. Note that each signal pin on the microprocessor can be configured for up to 5 different functions, so to achieve the maximum number of serial ports some functions/peripherals will be unavailable.

The SAME70 processor has two peripherals that can be used as a UART function: Three USARTs, representing Serial Port numbers 0 and 1 (USART numbers 0 - 1) Four UARTs, representing Serial Port numbers 3, 4, 5 and 6 (UART numbers 0 - 4) This mapping is described in the MODM7AE70 data sheet.

Serial Port P1 Pin P2 Pin Periph Mode USART/UART

0 RX 3 C USART 0 RX 0 TX 4 C USART 0 TX 0 CTS 29 C USART 0 CTS 0 RTS 38 C USART 0 RTS Note: Serial port 0 is stdio by default

1 RX 21 A USART 1 RX 1 TX 22 D USART 1 TX 1 RTS 32 A USART 1 RTS 1 CTS 33 A USART 1 CTS 1 DSR 16 A USART 1 DSR 1 DCD 17 A USART 1 DCD 1 DTR 18 A USART 1 DTR 1 RI 20 A USART 1 RI

2 RX 34 A UART 0 RX NOTE: tied to LED on mod-dev-70 2 TX 35 A UART 0 TX

3 RX 12 C UART 1 RX, PA5* 3 TX 31 C UART 1 TX 3 TX 42 C UART 1 TX (Alternate) 3 TX 44 D UART 1 TX (Alternate)

4 RX 41 C UART 2 RX 4 TX 44 C UART 2 TX

5 RX 23 A UART 3 RX 5 TX 7 A UART 3 TX 5 TX 24 B UART 3 TX (Alternate)

6 RX 10 C UART 4 RX 6 TX 7 C UART 4 TX CAUTION: P1.7 Cannot be used if the external address/data bus interface (EBI) is enabled.

12.33.2.6.9 Programmable Clock Output

Supported Platforms: MODM7AE70

The Power Management Controller (PMC) optimizes power consumption by controlling all system and user peripheral clocks. The PMC enables/disables the clock inputs to many of the peripherals and the Cortex-M7 processor. The clocking system in the SAME70 processor has a multitude of configuration options. The consequence of that is that it also complex. For detailed information, especially with respect to PCK registers, refer to chapter 31. Power Management Controller (PMC) of the SAME70 data sheet in your "\nburn\docs\Microchip" folder.

The PMC controller handles clocking for the entire processor. This examples if focused on the PCK outputs PCK0 and PCK1 on MODM7AE70:

- PCK0 on pin P1.31 and P2.11
- PCK1 on pin P2.8, P2.21, P2.24, P2.38, P2.39 and P2.40

For reference, PCK System Assignments:

- PCK3: ETM
- PCK4: UARTx/USARTx
- PCK5: MCANx

- PCK6: TCx
- PCK7: TC0

PMC_PCKx [x=0..7]: PMC Programmable Clock Register, Read/write Programmable Clock Prescaler and Programmable Clock Source. Bits 11:4 - PRES[7:0] Programmable Clock Prescaler Value Description 0 - 255 Selected clock is divided by PRES+1.

Bits 2:0 - CSS[2:0] Programmable Clock Source Selection Value Name Description 0 SLOW_CLK SLCK is selected 1 MAIN_CLK MAINCK is selected 2 PLLA_CLK PLLACK is selected 3 UPLL_CLK UPLLCKDIV is selected 4 MCK MCK is selected 5 AUDIO_CLK AUDIOPLLCLK is selected

Slow Clock: Slow R/C oscillator input to processor, not used. Main Clock: Oscillator input to processor (MAINCK), 12MHz. PLLA Clock: Generated from MAINCK by the PLLA and a predivider, 300MHz. UTMI PLL Clock: USB transmitter clock (UPLL_CLK). Master Clock: Output from Master Clock Controller, 150MHz. Audio Clock: External audio clock input.

Note

Processor electrical specifications limit the clock output frequency based on the pin group. In example testing on P2.11, the maximum output was 150MHz, but other pins may be lower. Refer to the pin group specification for the selected pin in your application.

PMC_SCER: PMC System Clock Enable Register, Enables the corresponding Programmable Clock output. PMC_↔_SCDR: PMC System Clock Disable Register, Disables the corresponding Programmable Clock output. PMC_↔_SCSR: PMC System Clock Status Register, Reads enable/disable state of Programmable Clock output.

12.33.2.6.10 PWM

Supported Platforms: MODM7AE70

Demonstrates the use of the PWM peripheral.

12.33.2.6.11 NTP & Real-Time Clock

Supported Platforms: MODM7AE70

NTP-SYSTEM-RTC DEMO APPLICATION

HARDWARE SETUP: Tested with MODM7AE70 on a MOD-DEV-70 hardware rev 1.93.

This example demonstrates the following features:

- Updates the system time with the current UTC time pulled from an available NTP server (pool.ntp.org). If your module is set to static IP address settings, then you will need to manually assign a DNS server address for this feature to work.
- Take the current system time and save it to the real-time clock (RTC). The RTC will then be running in sync with the system time.
- Take the current time in the RTC and set the system time information. The system time will then be running in sync with the RTC.
- When the device or module is reset or powered on again after a recent power loss, the system time will be reset, but the RTC can continue running until the super-capacitor is discharged. The software-reset option is available to demonstrate the RTC's ability to retain saved time information once it is properly set.
- Set the local time zone information with a call to `tzsetchar()`. For more more information on this function, refer to "Chapter 14 - NBTTime Library" of the NetBurner Runtime Libraries PDF document. In this example, Pacific time zone information is used as a demonstration.
- Display the current system and RTC times, and local time zone information if applicable. It is preferred that the RTC store the UTC time, not local time.
- In order to take advantage of setting and getting time information from the real-time clock (RTC), a device that contains an RTC component or a module that is mounted on a development board with an RTC component must be used. Current RTC components supported in this example at the time of this writing are the Intersil X1226 and NXP PCF8563.

12.33.2.6.12 Open All Serial Ports

Location: examples/PlatformSpecific/MODM7AE70/Serial-OpenAll

Supported Platforms: MODM7AE70

Demonstrates how to open all available serial ports on the MODM7AE70. There are a total of 7 named SerialPort 0 - 6, which correspond to USARTs 0 and 1, and UARTs 0 - 4. Note that the difference between a USART and a UART is that the UARTs can only be used for regular serial with TX and RX signals. The USARTs also include hardware handshaking, can be used for RS-422/485, and support other serial modes such as SPI.

12.33.2.6.13 SPI

MODM7AE70 SPI Platform Examples

- [Serial-to-SPI QuadSPI](#)
- [Serial-to-SPI USART](#)

12.33.2.6.13.1 Serial-to-SPI QuadSPI

Location: examples/PlatformSpecific/MODM7AE70/SPI/QuadSPI_Quadrant

Supported Platforms: MODM7AE70

The Serial-to-SPI example demonstrates the basic configuration of the SPI driver and sending/receiving data over an SPI interface. This example demonstrates how to use the driver for the QuadSPI peripheral in single-bit, master mode SPI.

Supported Platforms: MODM7AE70

The Serial-to-SPI example demonstrates the basic configuration of the SPI driver and sending/receiving data over an SPI interface. This example demonstrates how to use the driver for the QuadSPI peripheral in single-bit, master mode SPI.

12.33.2.6.13.2 Serial-to-SPI USART

Supported Platforms: MODM7AE70

The Serial-to-SPI example demonstrates the basic configuration of the SPI driver and sending/receiving data over an SPI interface. This example demonstrates how to use the driver for the USART peripheral in single-bit, master mode SPI.

12.33.2.6.14 SSC I2S Interface

Supported Platforms: MODM7AE70

SSC_I2S implements an I2S interface using the Synchronous Serial Controller and uses it to communicate with a [WM8904](#) Audio Codec.

12.33.2.6.15 DACC Tone Generator

Supported Platforms: MODM7AE70

Digital to Analog converter (referred to as "DACC" in the SAME70 manuals) example implementing basic DACC interface and timer triggered conversions.

12.33.2.7 SBE70LC

SBE70LC Platform Examples

- [GPIO - Simple](#)
- [PWM](#)

12.33.2.7.1 GPIO - Simple

Location: examples/PlatformSpecific/SAME70/SBE70LC/GPIOSimple

Supported Platforms: SBE70LC

Very simple GPIO example that toggles the GPIO on J1[3] of the SBE70LC development board. There is a more advanced multi-platform example in the `\nburn\examples` folder.

12.33.2.7.2 PWM

Location: examples/PlatformSpecific/SAME70/SBE70LC/PWM

Supported Platforms: SBE70LC

Demonstrates the use of the PWM peripheral. The PWM module is used to generating a 5kHz square wave with a 50% duty cycle on pin J1[7] and a 10 Hz square wave with a 60% duty cycle on pin J1[15].

12.33.3 SOMRT1061

SOMRT1061 Platform Examples

- [CAN](#)
- [Clock Module Example](#)
- [EFS Multiple Partitions](#)
- [External Real Time Clock](#)
- [Periodic Interrupt Timer \(PIT\)](#)
- [Open All UARTs](#)
- [SOMRT1061 Pin IRQ Example](#)
- [PWM](#)
- [SEMC Simple](#)
- [SEMC to SRAM, FPGA, LCD](#)
- [Analog to Digital \(ADC\)](#)
- [SPI PickIt Demo Board](#)
- [On-Chip Temperature Sensor \(TEMPMON\)](#)
- [Dev Board Temperature Sensor and ADC](#)

12.33.3.1 CAN

SOMRT1061 CAN Examples

- [FlexCan Interrupt Transfer](#)
- [FlexCan Loopback](#)

12.33.3.1.1 FlexCan Interrupt Transfer

Location: examples/PlatformSpecific/SOMRT1061/CAN

12.33.3.1.1.1 Overview

The flexcan_interrupt example shows how to use FlexCAN driver in none-blocking interrupt way. It has been ported from the NXP Freescale Software Library (FSL).

In this example, 2 boards are connected through CAN bus. Endpoint A(board A) send a CAN Message to Endpoint B(board B) when user press space key in terminal. Endpoint B receive the message, print the message content to terminal and echo back the message. Endpoint A will increase the received message and waiting for the next transmission of the user initiated.

For self wake up from STOP mode, since steps which MCU enters STOP mode differs on different MCUs, take flexcan_interrupt_transfer of twrke18f for example, user should do like this:

```
#include "fsl_smc.h"
...
...
flexcanConfig.enableSelfWakeup = true;
FLEXCAN_Init();
...
```

```

...
SMC_SetPowerModeStop(SMC, kSMC_PartialStop1);
if (wakenUp)
{
    PRINTF("B has been waken up!\r\n\r\n");
}
...
...

```

12.33.3.1.1.2 NXP SDK version

- Version: 2.16.000

12.33.3.1.1.3 Hardware requirements

- One DEV-SOMRT1061 Development board and one other CAN device, or two DEV-SOMRT1061 boards
- Personal Computer
- Serial terminal program connected to the DEV-SOMRT1061 RS-232 port

Warning

Two CAN devices are required for CAN to operate (except for loopback example)

12.33.3.1.1.4 Board settings

If using 2 DEV-SOMRT1061 boards, connect the two pin CAN header pins of each

12.33.3.1.1.5 Running the demo

After connecting the two boards, these instructions display on each terminal window. One board must be chosen as node A and the other board as node B. (Note: Node B should start first) Data is sent continuously between the node A and the node B.

Consider special errata 005829 feature, the 1st valid MB should be used as reserved one. The TX MB number will change from 8 to 9, while RX MB number will change from 9 to 10. This message displays on the node A terminal:

```

FLEXCAN Interrupt EXAMPLE *****

Message format: Standard (11 bit id)

Message buffer 10 used for Rx.

Message buffer 9 used for Tx.

Interrupt Mode: Enabled

Operation Mode: TX and RX --> Normal

```

Please select local node as A or B:

Note: Node B should start first.

Node:a

Press any key to trigger one-shot transmission

Rx MB ID: 0x123, Rx MB data: 0x0, Time stamp: 8877

Press any key to trigger the next transmission!

Rx MB ID: 0x123, Rx MB data: 0x1, Time stamp: 32459

Press any key to trigger the next transmission!

This message displays on the node B terminal:

```

FLEXCAN Interrupt EXAMPLE *****

Message format: Standard (11 bit id)

Message buffer 10 used for Rx.

Message buffer 9 used for Tx.

Interrupt Mode: Enabled

Operation Mode: TX and RX --> Normal

Please select local node as A or B:

Note: Node B should start first.

Node:b

Start to Wait data from Node A

Rx MB ID: 0x321, Rx MB data: 0x0, Time stamp: 5759

Wait Node A to trigger the next transmission!

Rx MB ID: 0x321, Rx MB data: 0x1, Time stamp: 57276

Wait Node A to trigger the next transmission!

```

12.33.3.1.2 FlexCan Loopback

Location: examples/PlatformSpecific/SOMRT1061/FlexcanLoopback

12.33.3.1.2.1 Overview

This example has been ported from the NXP Freescale Software Library (FSL).

The flexcan_loopback_functional example shows how to use the loopback test mode to debug your CAN Bus design:

To demonstrate this example, only one board is needed. The example will config one FlexCAN Message Buffer to Rx Message Buffer and the other FlexCAN Message Buffer to Tx Message Buffer with same ID. After that, the example will send a CAN Message from the Tx Message Buffer to the Rx Message Buffer through internal loopback interconnect and print out the Message payload to terminal.

12.33.3.1.2.2 NXP SDK version

- Version: 2.16.000

12.33.3.1.2.3 Hardware requirements

- DEV-SOMRT1061
- Personal Computer
- Serial terminal program connected to the DEV-SOMRT1061 RS-232 port

12.33.3.1.2.4 Board settings

No special settings are required.

12.33.3.1.2.5 Running the demo

Consider special errata 005829 feature, the 1st valid MB should be used as reserved one. The TX MB number will change from 8 to 9, while RX MB number will change from 9 to 10. When the example runs successfully, following information can be seen on the OpenSDA terminal:


```
==FlexCAN loopback functional example -- Start.==
```

```
Send message from MB9 to MB10
```

```
tx word0 = 0x11223344
```

```
tx word1 = 0x55667788
```

```
Received message from MB10
```

```
rx word0 = 0x11223344
```

```
rx word1 = 0x55667788
```

```
==FlexCAN loopback functional example -- Finish.==
```

12.33.3.2 Clock Module Example

12.33.3.2.1 Demonstrates the Clock Control Module (CCM) for the SOMRT1061 platform

Location: examples/PlatformSpecific/SOMRT1061/ClockModule

Supported Platforms

- SOMRT1061

12.33.3.2.2 Overview

The Clock Control Module (CCM) generates and controls clocks to the various modules in the design and manages low power modes. This example enables, sets, and displays the available clock sources for the SOMRT1061 platform.

The CCM uses available clock sources to generate clock roots and includes separate dividers to control generation of core and bus root clocks (AXI, AHB, IPG).

12.33.3.2.3 Key Features

Feature	Description
Clock Configuration	Sets external oscillator and RTC crystal
Clock Speed Display	Reports all major clock frequencies
Clock Root Enumeration	Lists all peripheral clock root frequencies
Visual Feedback	LED blink at 2Hz to indicate operation

12.33.3.2.4 Clock Architecture

12.33.3.2.4.1 AMBA Background

AMBA (Arm Advanced Microcontroller Bus Architecture) is an open-standard, on-chip interconnect specification for the connection and management of functional blocks in system-on-a-chip (SoC) designs. AMBA was introduced by Arm in 1996. The first AMBA buses were:

- Advanced System Bus (ASB)
- Advanced Peripheral Bus (APB)

12.33.3.2.4.2 Clock Types

IPG Clock (IPBus Global Clock)

The IPG clock drives the AHB to IP Bridge (AIPSTZ); the IPBus drives chip-specific peripherals

The AIPS bridge serves as a protocol translator between the AHB system bus and the IPbus. The AHB to IP bridge is the interface between the AHB and on-chip IPS peripherals, which are sub-blocks containing readable/writable control and status registers.

The AHB master reads and writes these registers through the AIPSTZ. The bridge generates:

- Block enables
- Block address
- Transfer attributes
- Byte enables
- Write data as inputs to the IPS peripherals

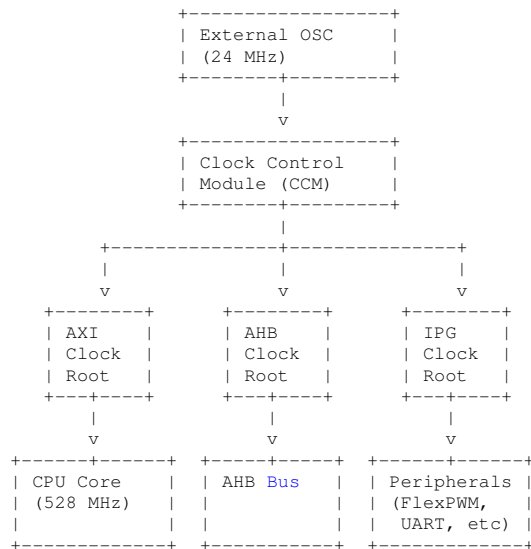
The bridge captures read data from the IPS interface and drives it on the AHB.

Each bridge that connects to the IPS (or peripherals) is referred to as AIPS. The chip has several separate AIPS modules, and peripherals are grouped and assigned under each AIPS block.

AHB Clock (AMBA High-performance Bus)

The AHB clock drives the high-performance system bus used for high-speed communication between the CPU and high-bandwidth peripherals.

12.33.3.2.4.3 Clock Hierarchy



12.33.3.2.5 Code Structure

12.33.3.2.5.1 Main Components

The example consists of three main functions:

displayClockSpeeds()

Reports all major clock frequencies and clock root frequencies:

Clock Type	Description
Ext OSC	External oscillator frequency (24 MHz)
CPU	CPU core clock frequency
AHB	AHB bus clock frequency
IPG	IPG bus clock and peripheral clock
PER	Peripheral clock frequency
SEMC	SDRAM controller clock frequency

Clock roots enumerated:

- kCLOCK_Usdhc1ClkRoot / kCLOCK_Usdhc2ClkRoot

- kCLOCK_FlexspiClkRoot / kCLOCK_Flexspi2ClkRoot
- kCLOCK_CsiClkRoot
- kCLOCK_LpspiClkRoot
- kCLOCK_TraceClkRoot
- kCLOCK_Sai1ClkRoot / kCLOCK_Sai2ClkRoot / kCLOCK_Sai3ClkRoot
- kCLOCK_Lpi2cClkRoot
- kCLOCK_CanClkRoot
- kCLOCK_UartClkRoot
- kCLOCK_LcdifClkRoot
- kCLOCK_SpdifClkRoot
- kCLOCK_Flexio1ClkRoot / kCLOCK_Flexio2ClkRoot

ledBlink()

Toggles LED0 (PIN_78_GPIO4_IO30) at 2Hz to provide visual feedback that the application is running.

UserMain()

Main entry point that:

1. Initializes the NetBurner system
2. Configures crystal frequencies (24 MHz OSC, 32 KHz RTC)
3. Displays all clock speeds to serial console
4. Enters main loop blinking LED at 2Hz

12.33.3.2.5.2 Clock Configuration

```
CLOCK_SetXtalFreq(24); // OSC is 24MHz. 88x multiplier provides 528MHz System Clock
CLOCK_SetRtcXtalFreq(32); // RTC crystal is 32KHz
```

The 24 MHz external oscillator is multiplied by 88 to generate the 528 MHz system clock.

12.33.3.2.6 Hardware Requirements

12.33.3.2.6.1 Development Board

- SOMRT1061 Development Kit

12.33.3.2.6.2 LED Connections

LED	Pin	GPIO
LED0	PIN_78_GPIO4_IO30	Used for blink
LED1	PIN_21_GPIO4_IO06	Not used in example
LED2	PIN_75_GPIO3_IO19	Not used in example
LED3	PIN_74_GPIO3_IO20	Not used in example

12.33.3.2.6.3 Serial Connection

Connect a serial terminal to the development board:

- Baud rate: 115200
- Data bits: 8

- Parity: None
- Stop bits: 1

Warning

The `fsl_` files use the `assert()` debug function to test values. If an assert fails, an error message is displayed on the serial port and the system is halted. Be sure to connect a serial terminal program to watch for any error messages.

12.33.3.2.7 Expected Output

Upon startup, the serial console will display:

```
Running SOMRT1061 Clock Module
```

```
Ext OSC: 24 MHz
```

```
CPU: 528 MHz
```

```
AHB: 132 MHz
```

```
IPG: 66 MHz, Peripheral Clock: 66 MHz
```

```
PER: 24 MHz
```

```
SEMC: 132 MHz
```

```
Running SOMRT1061 Clock Module
```

```
Ext OSC: 24 MHz
```

```
CPU: 528 MHz
```

```
AHB: 528 MHz
```

```
IPG: 132 MHz, Peripheral Clock: 528 MHz
```

```
PER: 24 MHz
```

```
SEMC: 176 MHz
```

```
CLOCK ROOT FREQ
```

```
kCLOCK_Usdhc1ClkRoot 198 MHz
```

```
kCLOCK_Usdhc2ClkRoot 198 MHz
```

```
kCLOCK_FlexspiClkRoot 315 MHz
```

```
kCLOCK_Flexspi2ClkRoot 45 MHz
```

```
kCLOCK_CsiClkRoot 12 MHz
```

```
kCLOCK_LpspiClkRoot 132 MHz
```

```
kCLOCK_TraceClkRoot 99 MHz
```

```
kCLOCK_Sai1ClkRoot 40 MHz
```

```
kCLOCK_Sai2ClkRoot 40 MHz
```

```
kCLOCK_Sai3ClkRoot 40 MHz
```

```
kCLOCK_Lpi2cClkRoot 60 MHz
```

```
kCLOCK_LcdifClkRoot 29 MHz
```

```
kCLOCK_SpdifClkRoot 30 MHz
```

```
kCLOCK_Flexio1ClkRoot 30 MHz
```

```
kCLOCK_Flexio2ClkRoot 30 MHz
```

LED0 will blink at 2Hz (500ms on, 500ms off).

12.33.3.2.8 API Reference

12.33.3.2.8.1 Clock Functions Used

The example uses the following functions from [fsl_clock.h](#):

Function	Description
<code>CLOCK_SetXtalFreq()</code>	Set external crystal oscillator frequency
<code>CLOCK_SetRtcXtalFreq()</code>	Set RTC crystal oscillator frequency
<code>CLOCK_GetOscFreq()</code>	Get external oscillator frequency
<code>CLOCK_GetCpuClkFreq()</code>	Get CPU clock frequency
<code>CLOCK_GetAhbFreq()</code>	Get AHB bus frequency
<code>CLOCK_GetIpgFreq()</code>	Get IPG bus frequency
<code>CLOCK_GetPeriphClkFreq()</code>	Get peripheral clock frequency
<code>CLOCK_GetPerClkFreq()</code>	Get PER clock frequency
<code>CLOCK_GetSemcFreq()</code>	Get SEMC (SDRAM controller) frequency
<code>CLOCK_GetClockRootFreq()</code>	Get specific clock root frequency

12.33.3.2.8.2 Required Headers

```
#include <predef.h>
#include <stdio.h>
```

```
#include <init.h>
#include <sim.h>
#include <nbrtos.h>
#include <iosys.h>
#include <pins.h>
#include <cpu_pins.h>
#include <string.h>
#include "fsl_clock.h"
```

Note

All fsl_MCUXpresso files are located in: \nburn\arch\cortex-m7\cpu\MIMXRT10xx\include

12.33.3.2.9 Usage Notes

12.33.3.2.9.1 PWM Clock Source

For applications using FlexPWM, the IPG clock is the source:

```
#define PWM_SRC_CLK_FREQ CLOCK_GetFreq(kCLOCK_IpgClk)
```

12.33.3.2.9.2 Timing Considerations

- The example uses a 500ms delay (TICKS_PER_SECOND / 2) for LED blinking
- Clock speeds are displayed once at startup
- The main loop runs continuously

12.33.3.2.9.3 Extending the Example

To modify clock speeds or add additional clock monitoring:

1. Reference the i.MX RT1061 Reference Manual for clock tree details
2. Use [fsl_clock.h](#) functions to read or modify clock configurations
3. Be aware that changing core clocks may require adjusting wait states and voltage regulators

12.33.3.2.10 Troubleshooting

Issue	Solution
LED not blinking	Check LED0 connection on PIN_78_GPIO4_IO30
No serial output	Verify serial terminal settings (115200 8N1)
Assert message	Review serial output for specific error
Incorrect frequencies	Ensure crystal oscillators are correct values

12.33.3.2.11 References

- i.MX RT1061 Reference Manual - Clock Control Module (CCM) chapter
- NXP MCUXpresso SDK Documentation
- AMBA Bus Specification (ARM)
- NetBurner [fsl_clock.h](#) API documentation

12.33.3.2.12 Related Examples

- GPIO examples for additional LED control
- Timer examples for precision timing
- FlexPWM examples for PWM clock usage

12.33.3.3 EFFS Multiple Partitions

This example program demonstrates how to create a second partition in the Application Flash, so that a guaranteed amount can be reserved for NNDK System usage, while still allowing general Application usage as well.

12.33.3.4 External Real Time Clock

Please refer to [I2C](#) Wire example: [RV3032 I2C Real-Time Clock](#)

12.33.3.5 Periodic Interrupt Timer (PIT)

Location: examples/PlatformSpecific/SOMRT1061/PIT

Warning

The NetBurner IntervalTimer is the preferred and safe method to use for Periodic Interrupts, as it will allocate and free timers without interfering with system operation. Although the FSL PIT timer example works, if an application uses it on a PIT timer used by the system, the device may halt or crash.

12.33.3.5.1 Overview

The PIT project is a simple demonstration program of the SDK PIT driver. It sets up the PIT hardware block to trigger a periodic interrupt after every 1 second. When the PIT interrupt is triggered a message is printed on the UART terminal and an LED is toggled on the board.

This has been ported from the NXP Freescale Software Library (FSL).

12.33.3.5.2 SDK version

- Version: 2.16.000

12.33.3.5.3 Toolchain supported

- MCUXpresso 11.10.0

12.33.3.5.4 Hardware requirements

- NetBurner DEV-SOMRT1061 Development board
- Personal Computer
- Serial terminal program with RS-232 connection

12.33.3.5.5 Board settings

No special settings are required.

12.33.3.5.6 Prepare the Demo

1. Connect a USB cable between the PC host and the USB port on the board.
2. Open a serial terminal with these settings:
 - 115200 baud rate
 - 8 data bits
 - No parity
 - One stop bit
 - No flow control
3. Download the program to the target board.
4. Example will automatically start

12.33.3.5.7 Running the demo

These instructions are displayed/shown on the terminal window:

```
Starting channel No.0 ...
Channel No.0 interrupt is occurred !
Channel No.0 interrupt is occurred !
Channel No.0 interrupt is occurred !
.....
.....
```

And you will find the user LED is taking turns to shine.

12.33.3.6 Open All UARTs

Location: examples/PlatformSpecific/SOMRT1061/OpenAllUARTs

Supported Platforms: SOMRT1061

Open all serial ports on SOMRT1061 Platform

There can be a maximum of 7 serial port configured on the device. Note that each signal pin on the microprocessor can be configured for up to 5 different functions, so to achieve the maximum number of serial ports some peripheral functions will be unavailable.

Serial Port	Pin Number	Peripheral Mode	LPUART	Comments for use with DEV-SOMRT1061-100CR
0 TX	8	2	LPUART 3 TX	Note: Serial port 0 is stdio by default
0 RX	7	2	LPUART 3 RX	
0 CTS	6	2	LPUART 3 CTS	
0 RTS	4	2	LPUART 3 RTS	
1 TX	42	2	LPUART 2 TX	Connected to GND through 0.1uF cap. High bus speed not recommended.
1 RX	3	2	LPUART 2 RX	
1 CTS	40	2	LPUART 2 CTS	
1 RTS	41	2	LPUART 2 RTS	
2 TX	88	2	LPUART 4 TX	Shares trace with RTC CLK OUT. Cannot be used if RTC clock output is enabled (not enabled by default).
2 RX	87	2	LPUART 4 RX	
2 CTS	3	2	LPUART 4 CTS	
2 RTS	2	2	LPUART 4 RTS	
3 TX	86	2	LPUART 5 TX	Shares trace with NBWIFI enable pin. Cannot be used simultaneously with NBWIFI.
3 RX	85	2	LPUART 5 RX	
3 CTS	81	2	LPUART 5 CTS	
3 RTS	82	2	LPUART 5 RTS	
4 TX	84	2	LPUART 6 TX	Only supports RS485.
4 RX	83	2	LPUART 6 RX	
4 CTS	78	2	LPUART 6 CTS	
4 RTS	80	2	LPUART 6 RTS	
5 TX	77	2	LPUART 7 TX	Shares trace with RGB red LED. HIGH bus speed not recommended.

Serial Port	Pin Number	Peripheral Mode	LPUART	Comments for use with DEV-SOMRT1061-100CR
5 RX	76	2	LPUART 7 RX	Shares trace with Enet 2. Ethernet driver source modifications can be made for testing while not mounted in DEV-SOMRT1061-100CR. Warning LPUART7 RX cannot be used while mounted in the DEV-SOMRT1061-100CR
6 TX	29	2	LPUART 8 TX	Shares trace with SDHC (PIN_29_↔USDHC1_DATA2). Trace pulled high with 47k resistor. High bus speed not recommended. Cannot be used with SD card is mounted.
6 RX	28	2	LPUART 8 RX	Shares trace with SDHC (PIN_28_↔USDHC1_DATA3). Trace pulled high with 47k resistor. High bus speed not recommended. Cannot be used with SD card is mounted.
6 CTS	25	2	LPUART 8 CTS	Shares trace with SDHC (PIN_25_↔USDHC1_DATA0). Trace pulled high with 47k resistor. High bus speed not recommended. Cannot be used with SD card is mounted.
6 RTS	24	2	LPUART 8 RTS	Shares trace with SDHC (PIN_25_↔USDHC1_DATA1). Trace pulled high with 47k resistor. High bus speed not recommended. Cannot be used with SD card is mounted. Warning LPUART 8 cannot be used if an SD card is mounted in the SD card socket.

12.33.3.7 SOMRT1061 Pin IRQ Example

12.33.3.7.1 SOMRT1061 PinIrq Application

12.33.3.7.1.1 Overview

This application demonstrates how to use pin interrupt functionality on the SOMRT1061 platform. It implements a real-time edge detection system that monitors pin state changes and provides visual feedback through an LED indicator.

12.33.3.7.1.2 Location

examples/PlatformSpecific/SOMRT1061/PinIrq

12.33.3.7.1.3 Supported Platforms

- SOMRT1061

12.33.3.7.1.4 Application Description

The PinIrq application showcases interrupt-driven GPIO monitoring using NetBurner's RTOS capabilities. The program:

- Monitors pin 30 for edge transitions (both rising and falling edges)
- Uses a semaphore-based notification system to handle interrupts safely
- Provides real-time feedback through pin 78 (LED output)
- Tracks and reports the total number of detected transitions
- Demonstrates proper interrupt handling in an RTOS environment

12.33.3.7.1.5 Key Features

Hardware Configuration

- **Input Pin:** Pin 30 (configured as input with interrupt capability)
- **Output Pin:** Pin 78 (LED indicator)
- **Edge Detection:** Both rising and falling edges

Software Architecture

- **Interrupt Service Routine:** Minimal ISR that posts to a semaphore
- **Task-based Processing:** Dedicated task handles the actual pin change logic
- **Thread-safe Communication:** Uses OS semaphores for inter-task communication
- **Real-time Monitoring:** Continuous status reporting every second

12.33.3.7.1.6 How It Works

1. **Initialization:** The application configures pin 30 as input and pin 78 as output
2. **Interrupt Setup:** Registers an interrupt handler for pin 30 that triggers on both edges
3. **Event Handling:** When a pin transition occurs:
 - The interrupt handler (`NotifyFunc`) posts to a semaphore
 - The `PinChangeTask` wakes up and processes the event
 - The transition counter is incremented
 - The LED state is updated to follow the input pin state
4. **Status Reporting:** Main loop continuously prints the current transition count and pin state

12.33.3.7.1.7 Code Structure

Main Components

- **NotifyFunc():** Interrupt service routine that signals pin changes
- **PinChangeTask():** Task that handles pin change processing
- **UserMain():** Main application loop with status reporting
- **Semaphore (NotifySem):** Synchronization primitive for interrupt handling

Global Variables

- **Transitions:** Counter tracking total number of edge detections
- **irq_pin_num:** Pin number for interrupt monitoring (pin 30)

12.33.3.7.1.8 Usage

1. Connect a switch or signal source to pin 30
2. Connect an LED to pin 78 to observe the output
3. Compile and upload the application to the SOMRT1061
4. Monitor the serial output to see transition counts and pin states
5. Toggle the input on pin 30 to observe the LED response and counter updates

12.33.3.7.1.9 Expected Output

The application will continuously output status information in the format:

```
Running SOMRT1061 PinIrq
{We have detected X edges pin is currently: Y
```

Where X is the total transition count and Y is the current pin state (0 or 1).

12.33.3.7.1.10 Technical Notes

- The interrupt handler is kept minimal to avoid issues in interrupt context
- All significant processing is performed in task context for safety
- The LED output inverts the input state, providing clear visual feedback
- System diagnostics are enabled for debugging purposes

12.33.3.7.1.11 Dependencies

- NetBurner RTOS
- Pin interrupt functionality
- Standard NetBurner GPIO libraries

12.33.3.8 PWM

SOMRT1061 PWM Examples

- [FlexPWM 3-Channel](#)
- [QTMR PWM](#)

12.33.3.8.1 FlexPWM 3-Channel

12.33.3.8.1.1 Overview

This example has been ported from the NXP Freescale Software Library (FSL).

The PWM project is a simple demonstration program of the SDK PWM driver. The pulse width modulator (PWM) module contains PWM submodules, each of which is set up to control a single half-bridge power stage. Fault channel support is provided. This PWM module can generate various switching patterns, including highly sophisticated waveforms. It can be used to control all known Switched Mode Power Supplies (SMPS) topologies.

The project uses eFlexPWM to generate A-phase pwm, B-phase pwm and C-phase pwm through submodule 0, submodule 1, submodule 2.

Submodule 0 can generate one complementary PWM: PWM A and PWM B. For PWM A, PWM A(1kHz 50% duty cycle) is setup, PWM A works based on SignedCenterAligned. PWM B operates in PWM A complementary mode, so the dutycycle field of PWM B does not matter. To prevent short circuits, the dead time count is set, about 650ns. Submodule 1 and submodule 2 are set to MasterSync. This means that submodule 1 and submodule 2 are initialized synchronously from submodule 0. In MasterSync mode, submodule 0 can be used as the main module to generate control signals to control other modules.

This program is set to center-aligned PWM mode, and the duty cycle and position of the PWM can be changed by modifying the value of pwmVal in this mode.

In this project, the duty cycle of submodule 0 is pwmVal, the duty cycle of submodule 1 is pwmVal divided by 2 and the duty cycle of submodule 2 is pwmVal divided by 4.

PWM_SetPwmLdok() can issue the load command to multiple submodules at the same time. The values are loaded immediately if kPWM_ReloadImmediate option was chosen during config. Else the values are loaded at the next PWM reload point.

The log below shows example output of the PWM driver demo in the oscilloscope :

12.33.3.8.2 QTMR PWM

Location: examples/PlatformSpecific/SOMRT1061/PWM

Supported Platforms: SOMRT1061

The QTMR project is a demonstration program of the SDK Quad Timer (QTMR) driver's input capture and output pwm feature.

This has been ported from the NXP Freescale Software Library (FSL).

The example sets up a QTMR channel for input capture. Once the input signal is received, the example will print the capture frequency value while the check is interrupted and waiting for the capture frequency to stabilize.

The example also sets up one QTMR channel to output pwm. The user should probe a 50Khz PWM signal output with 50% dutycycle with a oscilloscope.

12.33.3.8.2.1 SDK version

- Version: 2.16.000

12.33.3.8.2.2 Hardware requirements

- DEV-SOMRT1061
- Personal Computer
- Serial terminal program connected to the DEV-SOMRT1061 RS-232 port

For reference, Xpresso github locations are:

- Drivers: <https://github.com/nxp-mcuxpresso/mcux-sdk/tree/main/drivers>
- Examples: https://github.com/nxp-mcuxpresso/mcux-sdk-examples/blob/main/evkcmimxrt10_examples/
- API Reference: https://mcuxpresso.nxp.com/api_doc/dev/210/modules.html

Main page: <https://github.com/nxp-mcuxpresso>. If you wish to pull the git repo: <https://github.com/NXPmicro/mcux-sdk>.

12.33.3.9 SEMC Simple

Location: examples/PlatformSpecific/SOMRT1061/SEMC

12.33.3.9.1 SOMRT1061 Smart External Memory Controller (SEMC) Simple Example

12.33.3.9.1.1 Overview

This example demonstrates the basic usage of the Smart External Memory Controller (SEMC) on the SOMRT1061 platform. The SEMC provides an interface to external memory devices with configurable timing and bus width settings.

12.33.3.9.1.2 Supported Platforms

- SOMRT1061

12.33.3.9.1.3 Application Description

The main application (main.cpp) provides an interactive demonstration of SEMC functionality with the following features:

Key Components

- **SEMC Configuration:** Configures the external memory controller with 8-bit bus width, synchronous clock mode, and burst operation
- **Clock Management:** Sets up appropriate clock multiplexers and dividers for SEMC operation
- **Interactive Testing:** Command-line interface for testing memory operations

Configuration Parameters

The SEMC is configured with the following settings:

```
Bus Width: 8-bit
Clock Mode: Synchronous
Burst Mode: 16-word burst
Column Address Width: 4-bit
Address Multiplexing: Enabled
```

Timing Configuration

- Setup times and hold times are configured for reliable memory access
- Clock dividers are set to achieve proper SEMC frequency
- USB PLL bypass is utilized during memory operations

Test Constants

- **Test Address Offset:** 0xA5
- **Test Data (16-bit):** 0x5A5A
- **Test Data (8-bit):** 0x5A

12.33.3.9.1.4 Interactive Commands

The application provides a command-line interface with the following options:

Command	Function	Description
R	Read	Reads data from the test address offset and displays the result
W	Write	Writes test data to the test address offset
T	Test Variable	Writes 0xDEADBEEF to a test variable in external memory
F	Fast Read	Performs 4096 consecutive read operations for performance testing
C	Continuous Write	Performs 4096 consecutive write operations with incremental data

12.33.3.9.1.5 Memory Layout

The application uses a special memory section `DATABUS_VAR_SECT` for placing variables in external memory. The `TestVar` variable is specifically allocated to this section to demonstrate external memory usage.

12.33.3.9.1.6 Clock Configuration

The application configures several clock domains:

- **SEMC Clock:** Derived from USB1 PLL with appropriate dividers
- **CPU Clock:** System CPU frequency
- **UART Clock:** For serial communication
- **USB PLL:** Used as clock source with bypass capability

12.33.3.9.1.7 Usage

1. **Compile and Flash:** Build the application for SOMRT1061 and flash to the device
2. **Serial Connection:** Connect to the device via UART to see output and provide input
3. **Interactive Testing:** Use the command interface to test different memory operations
4. **Monitor Output:** Observe memory access results and frequency information

12.33.3.9.1.8 Expected Output

Upon startup, the application displays:

- Application name and version
- SEMC frequency information
- UART frequency information
- Interactive command prompt

12.33.3.9.1.9 Technical Notes

Memory Access Patterns

- Single byte operations for basic read/write testing
- Burst operations for performance evaluation
- Variable placement in external memory sections

Clock Management

- USB PLL bypass is enabled during memory operations to ensure stable timing
- Multiple data synchronization barriers (`dsb`) are used to ensure proper instruction ordering
- Clock frequencies are displayed for debugging and verification

Error Handling

The application includes basic system diagnostics and proper initialization sequences to ensure reliable operation.

12.33.3.9.1.10 Dependencies

- NetBurner RTOS libraries
- FSL (Freescale) clock management libraries
- SEMC hardware abstraction layer
- Standard C libraries for I/O operations

12.33.3.9.1.11 Development Notes

- Remove `EnableSystemDiagnostics()` call for production builds
- Adjust timing parameters based on specific external memory device requirements
- Consider memory access patterns for optimal performance
- Ensure proper pin configuration for SEMC signals

12.33.3.10 SEMC to SRAM, FPGA, LCD

SEMC SRAM Example

Example application demonstrating the use of the Smart External Memory Controller (SEMC) on the NetBurner SOMRT1061-200IR module for interfacing with external SRAM devices.

12.33.3.10.0.1 Overview

This example shows how to:

- Configure the SEMC controller for 8-bit or 16-bit external SRAM
- Calculate proper timing parameters based on device datasheets
- Perform comprehensive memory testing
- Access external memory using different data widths (byte, word, dword)
- Measure memory performance

12.33.3.10.0.2 Additional Documentation

- [Document Index](#)
- [Quick Reference](#)
- [Hardware Connections](#)

12.33.3.10.0.3 Hardware Requirements

NetBurner Module

- **SOMRT1061-200IR** (NXP i.MX RT1061 microcontroller)

External Memory Device

- Asynchronous SRAM (8-bit or 16-bit data bus)
- Typical devices:
 - IS61WV51216BLL (512K x 16-bit SRAM)
 - IS61C5128AL (512K x 8-bit SRAM)
 - CY62157EV30LL (512K x 16-bit SRAM)
 - Similar devices with compatible timing

Required Connections

The SEMC interface uses dedicated pins on the RT1061. Connect your external SRAM to the following pins:

Data Bus Connections

8-bit Configuration:**

- D[7:0] -> EMC_00 to EMC_07

16-bit Configuration:**

- D[7:0] -> EMC_00 to EMC_07 (lower byte)
- D[15:8] -> EMC_30 to EMC_37 (upper byte)

Address Bus Connections

Number of address pins needed depends on SRAM size:

- A[0] -> EMC_09
- A[1] -> EMC_10
- A[2] -> EMC_11
- A[3] -> EMC_12
- A[4] -> EMC_13
- A[5] -> EMC_14
- A[6] -> EMC_15
- A[7] -> EMC_16
- A[8] -> EMC_41
- A[9] -> B0_00
- A[10] -> B0_01
- A[11] -> B0_02

Control Signal Connections

- **nWE** (Write Enable) -> EMC_19
- **nOE** (Output Enable) -> EMC_20
- **ADV** (Address Valid) -> EMC_22
- **CS** (Chip Select) -> EMC_41 (or B0_00, B0_01, B0_02 for CS1-CS3)

Optional Connections

- **CLK** -> SD_B0_02 or SD_B0_03 (for synchronous mode only)

Power Connections

- Connect SRAM VCC to appropriate voltage (typically 3.3V)
- Connect SRAM GND to ground
- Add 0.1uF decoupling capacitors near SRAM VCC pins

12.33.3.10.0.4 Memory Address Mapping

The SEMC controller maps external memory into the processor's address space:

Memory Map:

+-----+-----+	0x9FFFFFFF
Available SEMC	
Address Space	
(512 MB total)	
+-----+-----+	SEMC_Base <- SEMC_Base (Default configuration)
Internal Memory	
+-----+-----+	0x00000000

Default Configuration:**

- Base Address: SEMC_Base
- Size: 512KB (0x80000)

- Addressable range: SEMC_Base to 0x8007FFFF

You can modify these values in `main.cpp` to match your hardware:

```
#define SRAM_BASE_ADDR SEMC_Base
#define SRAM_SIZE      0x80000 // 512KB
```

12.33.3.10.0.5 Software Configuration

Selecting Bus Width

Edit `main.cpp` to select 8-bit or 16-bit bus width:

```
#define USE_16BIT_BUS true // Change to false for 8-bit SRAM
```

Timing Configuration

The example calculates timing parameters based on:

1. SEMC clock frequency (default: 100 MHz)
2. External SRAM datasheet specifications

Default timing specifications used:**

- Address setup: 15-20 ns
- Address hold: 10 ns
- Read enable: 40-50 ns
- Write enable: 40 ns
- Chip select setup: 20 ns

To adjust timing for your specific SRAM device:**

1. Consult your SRAM datasheet for timing specifications
2. Modify the `ConfigureSEMC_8bit()` or `ConfigureSEMC_16bit()` function
3. Use the provided macros for easy calculation:

```
// Example: For a SRAM requiring 25ns address setup time
cfg.addr_setup = CYCLES_TO_N_PLUS_1(NS_TO_CYCLES(25));
```

The macros handle the conversion:

- `NS_TO_CYCLES(ns)` - Converts nanoseconds to clock cycles
- `CYCLES_TO_N_PLUS_1(cycles)` - Adjusts for (n+1) timing parameters

Pin Assignment

Modify pin assignments if using different chip select or clock pins:

```
// Pin assignments - modify based on your hardware design
#define SEMC_CLK_PIN 67 // Clock output pin
#define SEMC_CS_PIN 68 // Chip select pin (CS0)
```

12.33.3.10.0.6 Application Features

1. Automatic SEMC Configuration

The application automatically configures the SEMC controller based on the selected bus width and calculates appropriate timing parameters.

2. Comprehensive Memory Testing

Runs multiple test patterns to verify memory functionality:

Walking 1's Test

- Writes and verifies a walking 1's pattern
- Detects stuck-at faults on data lines
- Catches address line shorts

Address Uniqueness Test

- Verifies each memory location is unique
- Detects address line aliasing
- Catches decoding issues

Random Pattern Test

- Multiple iterations with random data
- Comprehensive data integrity verification
- Stress testing capability

Byte Access Test

- Tests byte-level read/write operations
- Verifies byte-enable signals
- Critical for 8-bit operations

3. Access Method Demonstrations

The example demonstrates three access methods:

```
// Byte access (8-bit)
sram8[0] = 0x42;
uint8_t value = sram8[0];

// Word access (16-bit)
sram16[0] = 0x1234;
uint16_t word = sram16[0];

// Dword access (32-bit)
sram32[0] = 0xDEADBEEF;
uint32_t dword = sram32[0];
```

4. Performance Measurement

Measures and reports:

- Sequential byte read/write throughput
- Sequential word read/write throughput
- Actual MB/s performance for different access patterns

12.33.3.10.0.7 Usage**1. Connect Hardware**

Connect your external SRAM device to the SEMC pins as described in the Hardware Requirements section.

2. Configure Software

Edit `main.cpp` to match your hardware:

- Set `USE_16BIT_BUS` to match your SRAM data bus width
- Adjust `SRAM_SIZE` to match your SRAM capacity
- Modify timing parameters if needed for your specific device

3. Build and Load

Build the application and load it to your SOMRT1061 module:

```
make clean
make PLATFORM=SOMRT1061
make load DEVIP=<your_device_ip>
```

4. Monitor Output

Connect to the serial console to see the application output:

```
=====
SEMC SRAM Example
=====
Platform: SOMRT1061-200IR
MCU: NXP i.MX RT1061
=====

Selected configuration: 16-bit data bus

Configuring SEMC for 16-bit SRAM...
SEMC Clock: 100 MHz
Clock period: 10 ns
Configuration complete
...

=====
COMPREHENSIVE MEMORY TEST SUITE
=====
Base Address: SEMC_Base
Size: 524288 bytes (512 KB)
=====

Test 1: Walking 1's Pattern
Walking 1's test: SEMC_Base (524288 bytes)...
Walking 1's test PASSED

Test 2: Address Uniqueness
...

=== Performance Demonstration ===
Test size: 8192 bytes

Performance Results:
Byte write: 1234 ticks (XX.XX MB/s)
Byte read: 1156 ticks (XX.XX MB/s)
...
```

12.33.3.10.0.8 Troubleshooting

Memory Tests Fail

Problem:** Walking 1's test or address uniqueness test fails

Solutions:**

1. Verify all pin connections between SOMRT1061 and SRAM
2. Check power supply to SRAM (should be stable 3.3V)
3. Add or verify decoupling capacitors near SRAM
4. Check SRAM datasheet and increase timing margins
5. Verify ground connections

Example timing adjustment:**

```
// Increase read timing by 20ns
cfg.rd_en_hi = CYCLES_TO_N_PLUS_1(NS_TO_CYCLES(70)); // Was 50ns
```

Intermittent Errors

Problem:** Tests sometimes pass, sometimes fail

Solutions:**

1. Add timing margin (increase all timing parameters by 20-30%)
2. Check signal integrity with oscilloscope
3. Reduce SEMC clock frequency if possible
4. Check for noise on power supply
5. Verify PCB layout follows good design practices

Build Errors

Problem:** Compilation errors

Solutions:**

1. Verify NNDK_ROOT is set correctly
2. Ensure PLATFORM=SOMRT1061 is set
3. Check that [semc.h](#) is available in your SDK
4. Verify you're using a compatible NetBurner SDK version

No Output on Console

Problem:** Application loads but no serial output

Solutions:**

1. Verify serial connection settings (115200 baud, 8N1)
2. Check that [init\(\)](#) is called in `UserMain()`
3. Try a simple `printf` test before SEMC configuration

12.33.3.10.0.9 Advanced Usage

Multiple Chip Selects

To use multiple SRAM devices with different chip selects:

```
// First SRAM - CS0
SEMC_cfg_t cfg0 = ConfigureSEMC_16bit();
cfg0.csPin = Pin_EMC_41; // CS0
ConfigureSEMC(SEMC_Base, 0x80000, cfg0);

// Second SRAM - CS1 (non-overlapping address)
SEMC_cfg_t cfg1 = ConfigureSEMC_16bit();
cfg1.csPin = Pin_B0_00; // CS1
ConfigureSEMC(0x80080000, 0x80000, cfg1);

// Access first SRAM
uint8_t *sram_cs0 = (uint8_t *)SEMC_Base;
sram_cs0[0] = 0x42;

// Access second SRAM
uint8_t *sram_cs1 = (uint8_t *)0x80080000;
sram_cs1[0] = 0xAB;
```

Synchronous Mode

For synchronous PSRAM devices with burst capability:

```
SEMC_cfg_t syncConfig = {};

// Synchronous configuration
syncConfig.busWidth = SEMC_BusWidth_t::Width_16;
syncConfig.clkMode = SEMC_ClkMode_t::Sync;
syncConfig.burstLen = SEMC_Burst_t::Burst_8;
```

```

syncConfig.colAddrWidth = SEMC_ColAddrWidth_t::Width_8;

// Synchronous timing
syncConfig.cs_setup = 2;
syncConfig.cs_hold_min = 2;
syncConfig.addr_setup = 1;
syncConfig.addr_hold = 1;
syncConfig.wr_dat_setup = 1;
syncConfig.wr_dat_hold = 0;
syncConfig.latency = 3;           // Read latency cycles
syncConfig.rd_cycle = 5;         // Total read cycle
syncConfig.cs_min_intv = 1;
syncConfig.rd_hold = 1;

// Pin assignments
syncConfig.clkPin = Pin_SD_B0_02; // Clock is required for sync mode
syncConfig.csPin = Pin_EMC_41;

ConfigureSEMC(0x98000000, 0x400000, syncConfig); // 4MB PSRAM

```

Custom Timing Calculations

For precise timing based on your device datasheet:

```

// Example: IS61WV51216BLL SRAM Datasheet
// tAA (Address Access Time): 10ns
// tOHA (Output Hold Time): 3ns
// tWP (Write Pulse Width): 8ns
// tAS (Address Setup Time): 0ns
// tAH (Address Hold Time): 3ns

SEMC_cfg_t customTiming = {};

// Add 20% margin to all timing
const float TIMING_MARGIN = 1.2f;

// Calculate timing with margin
uint32_t tAA_cycles = NS_TO_CYCLES(10 * TIMING_MARGIN);
uint32_t tWP_cycles = NS_TO_CYCLES(8 * TIMING_MARGIN);
uint32_t tAS_cycles = NS_TO_CYCLES(0 * TIMING_MARGIN);
uint32_t tAH_cycles = NS_TO_CYCLES(3 * TIMING_MARGIN);

customTiming.busWidth = SEMC_BusWidth_t::Width_16;
customTiming.clkMode = SEMC_ClkMode_t::Async;
customTiming.muxMode = SEMC_AddrMux_t::Mux;

// Apply calculated timing
customTiming.addr_setup = CYCLES_TO_N_PLUS_1(tAS_cycles);
customTiming.addr_hold = CYCLES_TO_N_PLUS_1(tAH_cycles);
customTiming.rd_en_hi = CYCLES_TO_N_PLUS_1(tAA_cycles);
customTiming.wr_en_lo = CYCLES_TO_N_PLUS_1(tWP_cycles);
customTiming.wr_en_hi = CYCLES_TO_N_PLUS_1(tWP_cycles);

customTiming.clkPin = Pin_SD_B0_02;
customTiming.csPin = Pin_EMC_41;

ConfigureSEMC(SEMC_Base, 0x100000, customTiming);

```

12.33.3.10.0.10 Timing Parameter Reference

Asynchronous Mode Parameters (used in this example)

Parameter	Description	Units	Notes
cs_setup	Chip select setup before address valid	n+1 cycles	Time CS asserted before address
cs_hold_min	Minimum chip select hold after cycle	n+1 cycles	Time CS held after cycle end
addr_setup	Address setup before control signal	n+1 cycles	Address stable before nOE/nWE
addr_hold	Address hold after control signal	n+1 cycles	Address held after nOE/nWE
wr_en_lo	Write enable low pulse width	n+1 cycles	nWE low time
wr_en_hi	Write enable high period	n+1 cycles	nWE high time
rd_en_lo	Read enable low pulse width	n+1 cycles	nOE low time
rd_en_hi	Read enable high period	n+1 cycles	nOE high time, includes data valid
addr_wr_hold	Address to write data hold	n+1 cycles	Address held after write data
turnaround	Bus turnaround time	n cycles	Time between read/write cycles
cs_min_intv	Minimum CS interval	n+1 cycles	Minimum time between CS assertions

Synchronous Mode Parameters

Parameter	Description	Units	Notes
cs_setup	Chip select setup before clock	n+1 cycles	CS setup to clock edge
cs_hold_min	Chip select hold after clock	n+1 cycles	CS held after clock edge
addr_setup	Address setup to clock	n+1 cycles	Address stable before clock
addr_hold	Address hold from clock	n cycles	Address held after clock
wr_dat_setup	Write data setup	n+1 cycles	Data setup before clock
wr_dat_hold	Write data hold	n cycles	Data held after clock
latency	Read data latency	n cycles	Clock cycles before first data
rd_cycle	Total read cycle length	n+1 cycles	Complete read cycle time
rd_hold	Read data hold	n cycles	Data hold after clock
cs_min_intv	Minimum CS interval	n+1 cycles	Minimum time between operations

12.33.3.10.0.11 Memory Test Reference

Walking 1's Test

Tests each bit position independently by walking a single '1' bit through all positions.

What it detects:**

- Stuck-at-0 faults (bit always reads 0)
- Stuck-at-1 faults (bit always reads 1)
- Data line shorts to ground/VCC
- Basic address line functionality

Pattern:**

```
Address 0: 0x00000001
Address 1: 0x00000002
Address 2: 0x00000004
Address 3: 0x00000008
...
Address 31: SEMC_Base
Address 32: 0x00000001 (repeats)
```

Address Uniqueness Test

Writes the address value to each location, then verifies each location contains its unique address.

What it detects:**

- Address line aliasing (multiple addresses map to same location)
- Missing address line connections
- Address decoder problems
- Memory size mismatches

Pattern:**

```
Write to SEMC_Base: value SEMC_Base
Write to 0x80000004: value 0x80000004
Write to 0x80000008: value 0x80000008
...
Then verify each location
```

Random Pattern Test

Writes and verifies random data patterns with multiple iterations.

What it detects:**

- Data-dependent errors
- Timing-sensitive issues

- Crosstalk between data lines
- Pattern-sensitive failures

```
Usage:**
// Quick verification
TestRandomPattern(baseAddr, size, 1);

// Stress test
TestRandomPattern(baseAddr, size, 1000);
```

Byte Access Test

Specifically tests byte-level operations (8-bit access).

What it detects:**

- Byte enable signal problems
- Word alignment issues
- Byte-level data corruption

```
Pattern:**
Address 0: 0x00
Address 1: 0x01
Address 2: 0x02
...
Address 255: 0xFF
Address 256: 0x00 (repeats)
```

12.33.3.10.0.12 Performance Considerations

Bus Width Impact

8-bit bus:**

- Lower pin count (8 data pins)
- Suitable for byte-oriented data
- 32-bit access requires 4 bus cycles
- Typical: 10-30 MB/s throughput

16-bit bus:**

- Medium pin count (16 data pins)
- 2x throughput vs 8-bit
- 32-bit access requires 2 bus cycles
- Typical: 20-60 MB/s throughput

Access Pattern Optimization

Sequential access:**

```
// Efficient: Sequential access
for (uint32_t i = 0; i < size; i++) {
    sram8[i] = data[i];
}
```

Random access:**

```
// Less efficient: Random access
for (uint32_t i = 0; i < size; i++) {
    sram8[rand() % size] = data[i];
}
```

Data Type Selection

Choose the access width that matches your data:

```
// Byte data: Use uint8_t*
uint8_t *buffer = (uint8_t *)SRAM_BASE_ADDR;
buffer[i] = byte_value;

// Word data: Use uint16_t* (best for 16-bit bus)
uint16_t *buffer = (uint16_t *)SRAM_BASE_ADDR;
buffer[i] = word_value;

// Dword data: Use uint32_t*
uint32_t *buffer = (uint32_t *)SRAM_BASE_ADDR;
buffer[i] = dword_value;
```

12.33.3.10.0.13 API Reference

Configuration Functions

ConfigureSEMC_8bit()

Creates SEMC configuration for 8-bit asynchronous SRAM.

- **Returns:** [SEMC_cfg_t](#) structure
- **Use:** For 8-bit data bus SRAM devices

ConfigureSEMC_16bit()

Creates SEMC configuration for 16-bit asynchronous SRAM.

- **Returns:** [SEMC_cfg_t](#) structure
- **Use:** For 16-bit data bus SRAM devices

Memory Test Functions

TestWalking1s(baseAddr, sizeBytes)

Performs walking 1's test on specified memory region.

- **Parameters:**
 - baseAddr: Starting address
 - sizeBytes: Number of bytes to test
- **Returns:** true if passed, false if failed

TestAddressUniqueness(baseAddr, sizeBytes)

Verifies each address location is unique.

- **Parameters:**
 - baseAddr: Starting address
 - sizeBytes: Number of bytes to test
- **Returns:** true if passed, false if failed

TestRandomPattern(baseAddr, sizeBytes, iterations)

Tests with random data patterns.

- **Parameters:**
 - baseAddr: Starting address
 - sizeBytes: Number of bytes to test
 - iterations: Number of test iterations
- **Returns:** true if passed, false if failed

TestByteAccess (baseAddr, sizeBytes)

Tests byte-level access patterns.

- **Parameters:**
 - baseAddr: Starting address
 - sizeBytes: Number of bytes to test
- **Returns:** true if passed, false if failed

RunMemoryTestSuite (baseAddr, sizeBytes)

Runs all memory tests in sequence.

- **Parameters:**
 - baseAddr: Starting address
 - sizeBytes: Number of bytes to test
- **Returns:** true if all tests passed

Demonstration Functions**DemoByteAccess ()**

Demonstrates 8-bit byte access operations.

DemoWordAccess ()

Demonstrates 16-bit word access operations.

DemoDwordAccess ()

Demonstrates 32-bit dword access operations.

DemoPerformance ()

Measures and displays memory performance metrics.

12.33.3.10.0.14 References**NetBurner Documentation**

- SEMC API: C:\gitroot\ncburn3\arch\cortex-m7\cpu\MIMXRT10xx\include\semc.h
- SEMC Implementation: C:\gitroot\ncburn3\arch\cortex-m7\cpu\MIMXRT10xx\source\semc.c ←
cpp

NXP Documentation

- i.MX RT1061 Reference Manual
- i.MX RT1061 Datasheet
- SEMC Chapter in Reference Manual

SRAM Device Examples

- IS61WV51216BLL - 512K x 16 SRAM (ISSI)
- CY62157EV30LL - 512K x 16 SRAM (Cypress)
- IS61C5128AL - 512K x 8 SRAM (ISSI)

12.33.3.10.0.15 Notes

1. Always add 10-20% timing margin beyond device datasheet minimums
 2. Test thoroughly with your specific hardware before deployment
 3. Monitor temperature effects on timing if operating in extreme conditions
 4. Use oscilloscope to verify signal integrity for high-speed operation
 5. Consider EMI/EMC requirements for your PCB design
-

12.33.3.10.1 Document Index**12.33.3.10.1.1 SEMC SRAM Example - Complete Project Index****Project Location**

C:\NetBurner\ClaudeWork\SEMC_SRAM\

File Overview**Source Code Files****src/main.cpp**

Purpose:** Complete SEMC SRAM example application

Contains:**

- UserMain() implementation
- SEMC configuration functions for 8-bit and 16-bit
- Comprehensive inline documentation

src/main_test.cpp**

- Memory test functions (4 different test patterns)
- Performance measurement functions
- Access method demonstrations

Key Functions

- ConfigureSEMC_8bit() - Create 8-bit configuration
 - ConfigureSEMC_16bit() - Create 16-bit configuration
 - TestWalking1s() - Walking 1's test pattern
 - TestAddressUniqueness() - Address uniqueness verification
 - TestRandomPattern() - Random data pattern test
 - TestByteAccess() - Byte-level access test
 - RunMemoryTestSuite() - Complete test suite
 - DemoByteAccess() - Byte access demonstration
 - DemoWordAccess() - Word access demonstration
 - DemoDwordAccess() - Dword access demonstration
 - DemoPerformance() - Performance measurement
-

src/semc_utils.h

Purpose:** Utility functions and helper macros

Contains:**

- Timing calculation macros
- Configuration validation functions
- SRAM device timing presets
- Helper functions for configuration
- Debug print functions

Key Macros:**

- NS_TO_CYCLES () - Convert nanoseconds to cycles
- CYCLES_TO_N_PLUS_1 () - Calculate register values
- IS_ALIGNED () - Check address alignment
- IS_POWER_OF_2 () - Validate memory size

Key Functions:**

- CreateSEMCConfigFromSpec () - Auto-generate configuration
- PrintSEMCConfig () - Display configuration details
- ValidateSEMCConfig () - Validate configuration

Timing Presets:**

- SRAM_10NS - Ultra-fast SRAM (10ns access)
- SRAM_55NS - Standard SRAM (55ns access)
- SRAM_70NS - Slower SRAM (70ns access)

makefile (12 lines)

Purpose:** NetBurner build configuration

Settings:**

- Project name: SEMC_SRAM
- Target platform: SOMRT1061
- Source files: src/main.cpp

Documentation Files**README.txt**

Purpose:** Comprehensive guide to the project

Sections:**

1. Overview and features
2. Hardware requirements
3. Pin connections (detailed tables)
4. Memory address mapping
5. Software configuration

6. Timing configuration
7. Build instructions
8. Application features (detailed)
9. Usage instructions (step-by-step)
10. Troubleshooting (extensive)
11. Advanced usage (multiple CS, sync mode)
12. Timing parameter reference
13. Memory test reference
14. Performance considerations
15. API reference
16. File structure
17. References and support

Best for:**

- First-time users
- Complete understanding
- Reference during development
- Troubleshooting issues

QuickReference.dox FAST START

Purpose:** Quick start guide and reference

Sections:**

1. Essential configuration steps
2. Timing calculation formulas
3. Common SRAM speeds
4. Pin connections quick reference
5. Memory map
6. Common memory sizes
7. Troubleshooting quick fixes
8. Timing parameters reference
9. Configuration templates
10. Memory test functions
11. Multiple chip selects
12. Performance tips
13. Build commands
14. Common defines
15. Validation checklist

16. Useful macros

Best for:**

- Experienced developers
- Quick lookups
- Copy/paste templates
- Fast troubleshooting

HardwareConnections.dox HARDWARE GUIDE

Purpose:** Hardware connection and PCB design guide

Sections:**

1. Complete pin mapping tables
2. Example SRAM connections (3 detailed examples)
3. PCB design guidelines
4. Signal integrity recommendations
5. Power supply considerations
6. Layout best practices
7. Series termination guidance
8. Timing considerations
9. Common hardware issues and solutions
10. Test equipment recommendations
11. Address line requirements
12. Schematic symbol examples
13. Bill of materials
14. Hardware bring-up checklist

Best for:**

- Hardware designers
- PCB layout
- Hardware debugging
- Signal integrity analysis

Which File Should I Read First?**For Developers**New to SEMC?** -> Start with **README.txt** (complete guide)Experienced developer?** -> Jump to **QuickReference.dox** (fast start)Need specific code?** -> Look in **src/main_tests.cpp** (all examples)**For Hardware Engineers**Designing PCB?** -> Read **HardwareConnections.dox** firstTroubleshooting hardware?** -> Check **HardwareConnections.dox** issues sectionNeed pin mapping?** -> **QuickReference.dox** or **HardwareConnections.dox**

For Project Management

Evaluating feasibility? ** -> Check **README.txt** requirements section

Quick Start Path

Option 1: Minimal (10 minutes)

1. Read **QuickReference.dox** - Essential configuration
2. Modify **src/main.cpp** - Set USE_16BIT_BUS
3. Build and load - `make clean && make PLATFORM=SOMRT1061`
4. Monitor serial output - See test results

Option 2: Thorough (30 minutes)

1. Read **README.txt** - Complete understanding
2. Read **HardwareConnections.dox** - Verify connections
3. Modify **src/main.cpp** - Configure for your hardware
4. Build and load
5. Run tests and verify results

Option 3: Comprehensive (2 hours)

1. Read all documentation
 2. Study **src/main.cpp** - Understand all functions
 3. Review [src/semc_utils.h](#) - Learn utilities
 4. Modify for your specific SRAM device
 5. Build, load, and test thoroughly
 6. Verify with oscilloscope
-

Finding Specific Information

Configuration

- **8-bit setup:** README.txt -> Software Configuration
- **16-bit setup:** README.txt -> Software Configuration
- **Timing calc:** QuickReference.dox -> Timing Calculation
- **Pin assignments:** All docs have pin tables

Hardware

- **Pin connections:** HardwareConnections.dox -> Pin Mapping
- **PCB design:** HardwareConnections.dox -> PCB Guidelines
- **Example circuits:** HardwareConnections.dox -> Examples
- **BOM:** HardwareConnections.dox -> Bill of Materials

Testing

- **Test functions:** src/main_tests.cpp -> Memory Test Functions
- **Running tests:** README.txt -> Usage
- **Troubleshooting:** README.txt -> Troubleshooting

Performance

- **Optimization:** README.txt -> Performance Considerations
- **Measurement:** src/main.cpp -> DemoPerformance()

Advanced Topics

- **Multiple CS:** README.txt -> Advanced Usage
- **Sync mode:** README.txt -> Synchronous Mode
- **Custom timing:** QuickReference.dox -> Custom Calculations

Common Tasks - Where to Look

Task	Primary File	Section
First time setup	README.txt	Usage -> Steps 1-4
Change bus width	src/main.cpp	Line ~108: USE_16BIT_BUS
Adjust timing	src/main.cpp	ConfigureSEMC_8bit/16bit
Add new SRAM	src/semc_utils.h	Add timing preset
Debug hardware	HardwareConnections.dox	Common Issues
Optimize performance	README.txt	Performance Considerations
Multiple devices	README.txt	Advanced Usage -> Multiple CS
Build errors	README.txt	Troubleshooting -> Build Errors
Test failures	README.txt	Troubleshooting -> Memory Tests
Pin mapping	QuickReference.dox	Pin Connections

Hardware Datasheets

- **RT1061:** NXP i.MX RT1061 Reference Manual
- **SRAM:** Check your specific device datasheet

12.33.3.10.2 Quick Reference

SEMC Quick Reference Guide

Essential Configuration Steps

1. Basic 8-bit SRAM Setup

```
#include <semc.h>

// Create configuration
SEMC_cfg_t cfg = {};
cfg.busWidth = SEMC_BusWidth_t::Width_8;
cfg.clkMode = SEMC_ClkMode_t::Async;
cfg.burstLen = SEMC_Burst_t::Burst_1;
cfg.muxMode = SEMC_AddrMux_t::Mux;
cfg.colAddrWidth = SEMC_ColAddrWidth_t::Width_0;

// Timing (for 100 MHz SEMC clock)
cfg.cs_setup = 1;           // 2 cycles
cfg.cs_hold_min = 1;        // 2 cycles
cfg.addr_setup = 1;         // 2 cycles
```

```

cfg.addr_hold = 1;           // 2 cycles
cfg.wr_en_lo = 3;           // 4 cycles
cfg.wr_en_hi = 3;           // 4 cycles
cfg.rd_en_lo = 3;           // 4 cycles
cfg.rd_en_hi = 4;           // 5 cycles
cfg.turnaround = 2;         // 2 cycles

// Pins
cfg.clkPin = Pin_SD_B0_02;
cfg.csPin = Pin_EMCC_41;

// Apply configuration
ConfigureSEMC(SEMC_Base, 0x80000, cfg); // 512KB at SEMC_Base

```

2. Basic 16-bit SRAM Setup

```

// Same as 8-bit, but change bus width:
cfg.busWidth = SEMC_BusWidth_t::Width_16;

```

3. Memory Access

```

// Get pointers
uint8_t *sram8 = (uint8_t *)SEMC_Base;
uint16_t *sram16 = (uint16_t *)SEMC_Base;
uint32_t *sram32 = (uint32_t *)SEMC_Base;

// Write/read bytes
sram8[0] = 0x42;
uint8_t value = sram8[0];

// Write/read words
sram16[0] = 0x1234;
uint16_t word = sram16[0];

// Write/read dwords
sram32[0] = 0xDEADBEEF;
uint32_t dword = sram32[0];

```

Timing Calculation

Formula for Async Mode

```

Cycles needed = ceil(time_ns / clock_period_ns)
Register value = cycles - 1 (for n+1 parameters)

```

Example for 100 MHz clock (10 ns period):

- 50 ns requirement: $\text{ceil}(50/10) = 5$ cycles, register = 4
- 25 ns requirement: $\text{ceil}(25/10) = 3$ cycles, register = 2

Common SRAM Speeds

SRAM Speed	Address Access	Min Read Cycle	Min Write Cycle
10 ns	10 ns	10 ns	10 ns
55 ns	55 ns	55 ns	55 ns
70 ns	70 ns	70 ns	70 ns

Pin Connections Quick Reference

Data Bus

8-bit: EMC_00 to EMC_07
 16-bit: EMC_00 to EMC_07 (lower byte)
 EMC_30 to EMC_37 (upper byte)

Address Bus (use as many as needed)

A[0-7]: EMC_09 to EMC_16
 A[8]: EMC_41
 A[9]: B0_00
 A[10]: B0_01
 A[11]: B0_02

Control Signals

nWE: EMC_19 (Write Enable)
 nOE: EMC_20 (Output Enable)
 ADV: EMC_22 (Address Valid)

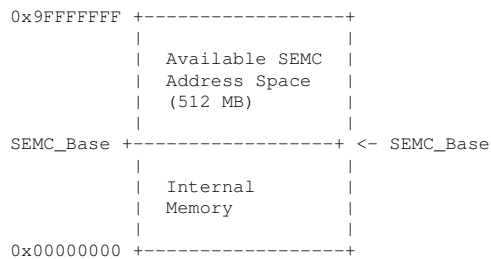
Chip Select Options

CS0: Pin_EMC_41 (default, function 0)
 CS1: Pin_B0_00 (function 6)
 CS2: Pin_B0_01 (function 6)
 CS3: Pin_B0_02 (function 6)

Clock (Sync mode only)

CLK0: Pin_SD_B0_02 or Pin_EMC_40
 CLK1: Pin_SD_B0_03 or Pin_AD_B0_11

Memory Map



Common Memory Sizes

Size	Hex Value	Address Lines Needed
64 KB	0x10000	16 (A0-A15)
128 KB	0x20000	17 (A0-A16)
256 KB	0x40000	18 (A0-A17)
512 KB	0x80000	19 (A0-A18)
1 MB	0x100000	20 (A0-A19)
2 MB	0x200000	21 (A0-A20)
4 MB	0x400000	22 (A0-A21)

Troubleshooting Quick Fixes

Memory Test Fails

1. Add 20% timing margin to all parameters
2. Verify power supply is stable 3.3V
3. Check all pin connections
4. Add decoupling capacitors (0.1uF near SRAM VCC)

Intermittent Errors

1. Increase all timing by 2-3 cycles
2. Check signal integrity with oscilloscope
3. Add series termination resistors (33ohm) on data lines

Timing Parameters Reference

Asynchronous Mode (n+1 = actual cycles is value + 1)

Parameter	Description	Type	Typical
cs_setup	CS setup before address	n+1	1-2
cs_hold_min	CS hold after cycle	n+1	1-2
addr_setup	Address setup before control	n+1	1-2
addr_hold	Address hold after control	n+1	1-2
wr_en_lo	Write enable low time	n+1	3-5
wr_en_hi	Write enable high time	n+1	3-5
rd_en_lo	Read enable low time	n+1	3-5
rd_en_hi	Read enable high time	n+1	4-6
addr_wr_hold	Address to write hold	n+1	1-2
turnaround	Bus turnaround time	n	2
cs_min_intv	Minimum CS interval	n+1	1-2

Configuration Templates

Ultra-Fast SRAM (10ns)

```
cfg.cs_setup = 0;
cfg.rd_en_lo = 1;
cfg.rd_en_hi = 2;
cfg.wr_en_lo = 1;
cfg.wr_en_hi = 2;
```

Standard SRAM (55ns)

```
cfg.cs_setup = 1;
cfg.rd_en_lo = 3;
cfg.rd_en_hi = 4;
cfg.wr_en_lo = 3;
cfg.wr_en_hi = 3;
```

Slow SRAM (70ns or slower)

```
cfg.cs_setup = 2;
cfg.rd_en_lo = 4;
cfg.rd_en_hi = 6;
cfg.wr_en_lo = 4;
cfg.wr_en_hi = 5;
```

Memory Test Functions

```
// Quick test (first 4KB)
TestWalking1s(SEMC_Base, 4096);

// Full test
TestWalking1s(SEMC_Base, SRAM_SIZE);
TestAddressUniqueness(SEMC_Base, SRAM_SIZE);
TestRandomPattern(SEMC_Base, SRAM_SIZE, 3);
TestByteAccess(SEMC_Base, SRAM_SIZE);

// Complete test suite
RunMemoryTestSuite(SEMC_Base, SRAM_SIZE);
```

Multiple Chip Selects

```
// CS0: 512KB at SEMC_Base
SEMC_cfg_t cfg0 = ConfigureSEMC_16bit();
cfg0.csPin = Pin_EMC_41;
ConfigureSEMC(SEMC_Base, 0x80000, cfg0);

// CS1: 512KB at 0x80080000 (non-overlapping)
SEMC_cfg_t cfg1 = ConfigureSEMC_16bit();
cfg1.csPin = Pin_B0_00;
ConfigureSEMC(SEMC_Base + 0x80000, 0x80000, cfg1);
```

Performance Tips

Maximize Throughput

1. Use 16-bit bus width when possible

2. Access data with matching width (use uint16_t* for 16-bit bus)
3. Keep data aligned to natural boundaries
4. Use sequential access patterns
5. Minimize random access

Typical Performance (100 MHz SEMC)

- 8-bit bus: 10-30 MB/s
- 16-bit bus: 20-60 MB/s
- Burst mode (sync): 80-100 MB/s

Common Defines

```
// Configuration
#define USE_16BIT_BUS true           // or false for 8-bit
#define SRAM_BASE_ADDR SEMC_Base
#define SRAM_SIZE 0x80000           // 512KB
#define SEMC_CLOCK_MHZ 100

// Pins
#define SEMC_CLK_PIN 67
#define SEMC_CS_PIN 68
```

Validation Checklist

- ☐ Base address is SEMC_Base or higher
- ☐ Size is power of 2
- ☐ Base address aligned to size
- ☐ No overlap with other CS regions
- ☐ Timing parameters meet datasheet requirements
- ☐ All required pins connected
- ☐ Power supply stable 3.3V
- ☐ Decoupling capacitors present
- ☐ Memory tests pass

Debug Output

Enable detailed timing output:

```
printf("SEMC Clock: %u MHz\n", SEMC_CLOCK_MHZ);
printf("Clock period: %u ns\n", 1000 / SEMC_CLOCK_MHZ);
printf("Read cycle: %u cycles = %u ns\n",
      (rd_en_lo+1) + (rd_en_hi+1),
      ((rd_en_lo+1) + (rd_en_hi+1)) * (1000/SEMC_CLOCK_MHZ));
```

Useful Macros

```
// From semc_utils.h
#define NS_TO_CYCLES(ns) (((ns) * SEMC_CLOCK_MHZ + 999) / 1000)
#define CYCLES_TO_NS_PLUS_1(cycles) ((cycles) > 0 ? (cycles) - 1 : 0)
#define IS_ALIGNED(addr, size) (((addr) & ((size) - 1)) == 0)
#define IS_POWER_OF_2(size) (((size) != 0) && (((size) & ((size) - 1)) == 0))
```

12.33.3.10.3 Hardware Connections

12.33.3.10.3.1 SEMC Hardware Connection Guide

Overview

This document provides detailed hardware connection information for interfacing external SRAM to the NetBurner SOMRT1061-200IR module using the SEMC (Smart External Memory Controller) interface.

WARNING** THIS GUIDE IS ONLY FOR REFERENCE PURPOSES. IT IS THE RESPONSIBILITY OF THE DESIGNER TO CHECK EACH PIN, SIGNAL AND FUNCTION FOR THEIR PARTICULAR DESIGN.

SOMRT1061-200IR Pin Mapping

Complete SEMC Pin List

The following pins on the SOMRT1061-200IR are dedicated to SEMC functionality:

Data Bus Pins

Lower Byte (D[7:0]) - Required for all configurations:**

SEMC Signal	Pin Name	Pin Function	Description
DATA[0]	EMC_00	GPIO_EMC_00	Data bit 0 (LSB)
DATA[1]	EMC_01	GPIO_EMC_01	Data bit 1
DATA[2]	EMC_02	GPIO_EMC_02	Data bit 2
DATA[3]	EMC_03	GPIO_EMC_03	Data bit 3
DATA[4]	EMC_04	GPIO_EMC_04	Data bit 4
DATA[5]	EMC_05	GPIO_EMC_05	Data bit 5
DATA[6]	EMC_06	GPIO_EMC_06	Data bit 6
DATA[7]	EMC_07	GPIO_EMC_07	Data bit 7

Upper Byte (D[15:8]) - Required for 16-bit configurations only:**

SEMC Signal	Pin Name	Pin Function	Description
DATA[8]	EMC_30	GPIO_EMC_30	Data bit 8
DATA[9]	EMC_31	GPIO_EMC_31	Data bit 9
DATA[10]	EMC_32	GPIO_EMC_32	Data bit 10
DATA[11]	EMC_33	GPIO_EMC_33	Data bit 11
DATA[12]	EMC_34	GPIO_EMC_34	Data bit 12
DATA[13]	EMC_35	GPIO_EMC_35	Data bit 13
DATA[14]	EMC_36	GPIO_EMC_36	Data bit 14
DATA[15]	EMC_37	GPIO_EMC_37	Data bit 15 (MSB)

Address Bus Pins

Number of address pins needed depends on memory size:**

SEMC Signal	Pin Name	Pin Function	Max Memory with this Address
ADDR[0]	EMC_09	GPIO_EMC_09	Required for all sizes
ADDR[1]	EMC_10	GPIO_EMC_10	Required for all sizes
ADDR[2]	EMC_11	GPIO_EMC_11	Required for all sizes
ADDR[3]	EMC_12	GPIO_EMC_12	Required for all sizes
ADDR[4]	EMC_13	GPIO_EMC_13	Required for all sizes
ADDR[5]	EMC_14	GPIO_EMC_14	Required for all sizes
ADDR[6]	EMC_15	GPIO_EMC_15	Required for all sizes

SEMC Signal	Pin Name	Pin Function	Max Memory with this Address
ADDR[7]	EMC_16	GPIO_EMC_16	Required for all sizes
ADDR[8]	EMC_41	GPIO_EMC_41	> 128KB
ADDR[9]	B0_00	GPIO_B0_00	> 256KB
ADDR[10]	B0_01	GPIO_B0_01	> 512KB
ADDR[11]	B0_02	GPIO_B0_02	> 1MB

Control Signal Pins

Required for all configurations:**

SEMC Signal	Pin Name	Pin Function	Description
WE_n	EMC_19	GPIO_EMC_19	Write Enable (active low)
OE_n	EMC_20	GPIO_EMC_20	Output Enable (active low)
ADV	EMC_22	GPIO_EMC_22	Address Valid (for multiplexed mode)

Chip Select Pins

Choose one CS pin for each memory device:**

SEMC Signal	Pin Name	Pin Function	Notes
CS0_n	EMC_41	GPIO_EMC_41	Default, most common
CS1_n	B0_00	GPIO_B0_00	For multiple devices
CS2_n	B0_01	GPIO_B0_01	For multiple devices
CS3_n	B0_02	GPIO_B0_02	For multiple devices

Byte Enable Pins (16-bit only)

Required for 16-bit configurations:**

SEMC Signal	Pin Name	Pin Function	Description
DQM0	EMC_08	GPIO_EMC_08	Lower Byte Mask
DQM1	EMC_38	GPIO_EMC_38	Upper Byte Mask

Clock Pin (Synchronous mode only)

Optional - only needed for synchronous/burst memory:**

SEMC Signal	Pin Name	Pin Function	Notes
CLK0	SD_B0_02	GPIO_SD_B0_02	Primary clock option
CLK0_ALT	EMC_40	GPIO_EMC_40	Alternative clock
CLK1	SD_B0_03	GPIO_SD_B0_03	Secondary clock option
CLK1_ALT	AD_B0_11	GPIO_AD_B0_11	Alternative clock

Example SRAM Connections

Example 1: 8-bit SRAM (512KB) - IS61C5128AL

Memory Device:** ISSI IS61C5128AL (512K x 8-bit SRAM)

Connection Table

SRAM Pin	Function	Connect To	SOMRT1061 Pin
1-8	DQ0-DQ7	Data	EMC_00 to EMC_07

SRAM Pin	Function	Connect To	SOMRT1061 Pin
9	VCC	Power	3.3V
10	A0	Address	EMC_09
11	A1	Address	EMC_10
12	A2	Address	EMC_11
13	A3	Address	EMC_12
14	A4	Address	EMC_13
15	A5	Address	EMC_14
16	A6	Address	EMC_15
17	A7	Address	EMC_16
18	CE_n	Chip Sel	EMC_41 (CS0)
19	WE_n	Write En	EMC_19
20	OE_n	Output En	EMC_20
21	A8	Address	EMC_41
22	A9	Address	B0_00
23	A10	Address	B0_01
24	A11	Address	B0_02
25-28	A12-A15	Address	Not used (512K only uses A0-A11)
29	A16	Address	Not used
30	A17	Address	Not used
31	A18	Address	Not used
32	GND	Ground	GND

Notes:**

- Add 0.1uF decoupling capacitor between VCC (pin 9) and GND (pin 32)
- For 512KB (512K x 8), you need A0-A18 (19 address lines)
- This device is 5V tolerant but operates at 3.3V

Example 2: 16-bit SRAM (512KB) - IS61WV51216BLL

Memory Device:** ISSI IS61WV51216BLL (512K x 16-bit SRAM)

Connection Table

SRAM Pin	Function	Connect To	SOMRT1061 Pin
1	A4	Address	EMC_13
2	A3	Address	EMC_12
3	A2	Address	EMC_11
4	A1	Address	EMC_10
5	A0	Address	EMC_09
6	CE_n	Chip Sel	EMC_41 (CS0)
7	DQ0	Data	EMC_00
8	DQ1	Data	EMC_01
9	DQ2	Data	EMC_02
10	DQ3	Data	EMC_03
11	VCC	Power	3.3V
12	GND	Ground	GND
13	DQ4	Data	EMC_04
14	DQ5	Data	EMC_05
15	DQ6	Data	EMC_06
16	DQ7	Data	EMC_07

SRAM Pin	Function	Connect To	SOMRT1061 Pin
17	WE_n	Write En	EMC_19
18	A17	Address	Not used (tie to GND)
19	A16	Address	Not used (tie to GND)
20	A15	Address	Not used (tie to GND)
21	A14	Address	Not used (tie to GND)
22	A13	Address	Not used (tie to GND)
23	A12	Address	Not used (tie to GND)
24	A11	Address	B0_02
25	A10	Address	B0_01
26	A9	Address	B0_00
27	A8	Address	EMC_41
28	A18	Address	Not used (tie to GND)
29	DQ15	Data	EMC_37
30	DQ14	Data	EMC_36
31	DQ13	Data	EMC_35
32	DQ12	Data	EMC_34
33	VCC	Power	3.3V
34	GND	Ground	GND
35	DQ11	Data	EMC_33
36	DQ10	Data	EMC_32
37	DQ9	Data	EMC_31
38	DQ8	Data	EMC_30
39	OE_n	Output En	EMC_20
40	LB_n	Low Byte	EMC_08 (DQM0)
41	UB_n	Upper Byte	EMC_38 (DQM1)
42	A5	Address	EMC_14
43	A6	Address	EMC_15
44	A7	Address	EMC_16

Notes:**

- Add 0.1uF decoupling capacitors between VCC (pins 11, 33) and GND (pins 12, 34)
- For 512KB (512K x 16), you need A0-A17 (18 address lines)
- Unused address lines A12-A18 should be tied to GND
- LB_n and UB_n are byte enable signals (active low)

Example 3: Multiple SRAM Devices

Configuration:** Two 16-bit SRAM devices for 1MB total

First Device (CS0)

Memory: IS61WV51216BLL #1
 Base Address: 0x80000000
 Size: 512KB
 Chip Select: EMC_41 (CS0)

Connections:

- Data: EMC_00-07 (lower), EMC_30-37 (upper)
- Address: EMC_09-16, EMC_41, B0_00-02 (A0-A10, A8 uses EMC_41 in address mode)
- Control: EMC_19 (WE), EMC_20 (OE)
- Chip Select: Configure EMC_41 as CS0

Second Device (CS1)

Memory: IS61WV51216BLL #2

Base Address: 0x80080000
Size: 512KB
Chip Select: B0_00 (CS1)

Connections:

- Data: EMC_00-07 (lower), EMC_30-37 (upper) [shared with CS0]
- Address: EMC_09-16, EMC_41, B0_01-02 [shared with CS0]
- Control: EMC_19 (WE), EMC_20 (OE) [shared with CS0]
- Chip Select: Configure B0_00 as CS1

Important:** When using multiple devices, data and address buses are shared. Only the chip select signal differs.

PCB Design Guidelines

Signal Integrity

1. Trace Length

- Keep all SEMC traces as short as possible
- Match trace lengths for data bus signals (within 20mm)
- Match trace lengths for address bus signals (within 30mm)

2. Trace Width

- Minimum trace width: 6 mil (0.15mm)
- Recommended: 8-10 mil (0.2-0.25mm) for better signal integrity

3. Trace Spacing

- Minimum spacing: 6 mil (0.15mm)
- Recommended: 8 mil (0.2mm) to reduce crosstalk

4. Via Usage

- Minimize vias on high-speed signals
- Use via-in-pad for BGA packages when necessary

Power Supply Considerations

1. SRAM Power Supply

- Provide clean, stable 3.3V supply
- Use dedicated LDO if possible for low noise
- Power supply should handle peak current during write operations

2. Decoupling Capacitors

- Place 0.1uF ceramic capacitor at each SRAM VCC pin
- Keep capacitor within 5mm of VCC pin
- Use short, wide traces to minimize inductance
- Add 10uF bulk capacitor for each SRAM device
- Use X7R or X5R dielectric for stability

3. Ground Connection

- Solid ground plane under SEMC signals
- Connect all SRAM GND pins directly to ground plane
- Multiple vias at each GND connection point

Layout Best Practices

1. Component Placement

- Place SRAM as close as possible to SOMRT1061
- Orient SRAM to minimize trace crossings
- Keep decoupling capacitors immediately adjacent to VCC pins

2. Signal Routing

- Route data bus on same layer if possible
- Route address bus on same layer if possible
- Avoid running signals parallel for long distances
- Use ground planes between signal layers

3. Layer Stack-Up (4-layer recommended)

Layer 1: Signal (data, address, control)
Layer 2: Ground plane
Layer 3: Power plane (3.3V)
Layer 4: Signal (return paths, additional routing)

Series Termination (if needed for high-speed operation)

For SEMC clock > 100 MHz or trace lengths > 3 inches:

1. Data Lines

- Add 33ohm series resistor close to driver (SOMRT1061 side)
- Place resistor within 10mm of source

2. Address Lines

- Add 33ohm series resistor close to driver
- Can use resistor arrays for space savings

3. Control Lines

- Add 33ohm series resistor on WE, OE, CS signals

Timing Considerations

Signal Quality Checks

Use oscilloscope to verify:

1. Setup Time

- Measure address valid to WE/OE assertion
- Should meet or exceed datasheet specification + 20% margin

2. Hold Time

- Measure address/data stable after WE/OE de-assertion
- Should meet or exceed datasheet specification + 20% margin

3. Signal Levels

- Logic high: > 2.0V (for 3.3V CMOS)
- Logic low: < 0.8V (for 3.3V CMOS)
- Check for overshoot/undershoot (should be < 10%)

4. Edge Rates

- Rise time: 1-3 ns typical
- Fall time: 1-3 ns typical
- Too fast can cause ringing, too slow reduces performance

Measurement Points

Key signals to probe:

- Data bus: DQ0, DQ7 (for 8-bit), DQ15 (for 16-bit)
- Address bus: A0, A10 (or highest address used)
- Control signals: WE_n, OE_n, CS_n
- Power supply: VCC at SRAM pin

Common Hardware Issues and Solutions**Issue 1: Memory Tests Fail Completely**

Symptoms:** All memory tests fail, reads return 0x00 or 0xFF

Check:**

1. Verify power supply is present at SRAM (3.3V)
2. Check all GND connections
3. Verify CS signal is connected and toggling
4. Check OE signal is connected (active low for reads)
5. Verify data bus connections D0-D7 (and D8-D15 for 16-bit)

Measurement:** Use oscilloscope to verify signals are toggling

Issue 2: Intermittent Failures

Symptoms:** Sometimes passes, sometimes fails

Possible Causes:**

1. Insufficient decoupling capacitors
2. Timing too aggressive (not enough margin)
3. Poor signal integrity (ringing, crosstalk)
4. Borderline power supply
5. Temperature effects

Solutions:**

1. Add more decoupling capacitors (0.1uF near SRAM)
2. Increase timing parameters by 20-30%
3. Add series termination resistors
4. Use dedicated LDO for SRAM power
5. Test at temperature extremes

Issue 3: Some Bits Always Wrong

Symptoms:** Specific data bits always read incorrectly

Possible Causes:**

1. Broken trace or cold solder joint on specific data line
2. Short to VCC or GND on specific pin
3. Damaged SRAM device

Solutions:**

1. Check continuity of specific data line with multimeter
2. Check for shorts to power/ground
3. Replace SRAM device
4. Inspect solder joints under microscope

Issue 4: Address Aliasing

Symptoms:** Different addresses return same data

Possible Causes:**

1. Missing address line connection
2. Address line shorted to another signal
3. Incorrect address pin assignment in software

Solutions:**

1. Verify all address line connections with multimeter
2. Check for shorts between address lines
3. Use address uniqueness test to identify which address bit is missing
4. Verify software configuration matches hardware

Issue 5: Works on Bench, Fails in Production

Symptoms:** Prototype works, production units fail

Possible Causes:**

1. Manufacturing process issues (cold joints, etc.)
2. Component tolerance variations
3. PCB impedance variations
4. Temperature range differences

Solutions:**

1. Add timing margin (30% instead of 20%)
2. Implement thorough production test
3. Use tighter component tolerances
4. Design for worst-case conditions

Test Equipment Recommendations

Essential Tools

1. Multimeter

- Verify power supply voltage
- Check continuity of signals
- Measure resistance for shorts

2. Oscilloscope

- Minimum 100 MHz bandwidth for SEMC signals
- 4 channels recommended (address, data, control)
- Use to verify timing relationships

3. Logic Analyzer

- Capture multiple signals simultaneously
- Verify bus protocol sequences
- Debug complex timing issues

Recommended Probe Points

Add test points on PCB for:

- VCC (power supply)
- GND (ground reference)
- Key data lines (DQ0, DQ7, DQ15)
- Key address lines (A0, A10)
- Control signals (WE, OE, CS)

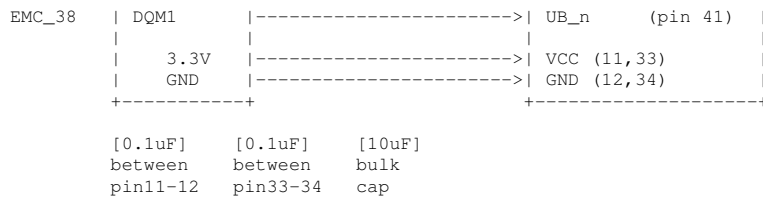
Address Line Requirements by Memory Size

Memory Size	Total Address Lines	Byte Address Range	Word Address Range (16-bit)
32 KB	15 (A0-A14)	0x0000 - 0x7FFF	0x0000 - 0x3FFF
64 KB	16 (A0-A15)	0x0000 - 0xFFFF	0x0000 - 0x7FFF
128 KB	17 (A0-A16)	0x00000 - 0x1FFFF	0x00000 - 0x0FFFF
256 KB	18 (A0-A17)	0x00000 - 0x3FFFF	0x00000 - 0x1FFFF
512 KB	19 (A0-A18)	0x00000 - 0x7FFFF	0x00000 - 0x3FFFF
1 MB	20 (A0-A19)	0x00000 - 0xFFFFF	0x00000 - 0x7FFFF

Note:** For x16 devices, divide byte count by 2 to get word count.

Schematic Symbol Example

SOMRT1061			SRAM (IS61WV51216BLL)		
+-----+			+-----+		
EMC_00	DATA[0]	----->	DQ0	(pin 7)	
EMC_01	DATA[1]	----->	DQ1	(pin 8)	
EMC_02	DATA[2]	----->	DQ2	(pin 9)	
EMC_03	DATA[3]	----->	DQ3	(pin 10)	
EMC_04	DATA[4]	----->	DQ4	(pin 13)	
EMC_05	DATA[5]	----->	DQ5	(pin 14)	
EMC_06	DATA[6]	----->	DQ6	(pin 15)	
EMC_07	DATA[7]	----->	DQ7	(pin 16)	
EMC_30	DATA[8]	----->	DQ8	(pin 38)	
EMC_31	DATA[9]	----->	DQ9	(pin 37)	
EMC_32	DATA[10]	----->	DQ10	(pin 36)	
EMC_33	DATA[11]	----->	DQ11	(pin 35)	
EMC_34	DATA[12]	----->	DQ12	(pin 32)	
EMC_35	DATA[13]	----->	DQ13	(pin 31)	
EMC_36	DATA[14]	----->	DQ14	(pin 30)	
EMC_37	DATA[15]	----->	DQ15	(pin 29)	
EMC_09	ADDR[0]	----->	A0	(pin 5)	
EMC_10	ADDR[1]	----->	A1	(pin 4)	
EMC_11	ADDR[2]	----->	A2	(pin 3)	
EMC_12	ADDR[3]	----->	A3	(pin 2)	
EMC_13	ADDR[4]	----->	A4	(pin 1)	
EMC_14	ADDR[5]	----->	A5	(pin 42)	
EMC_15	ADDR[6]	----->	A6	(pin 43)	
EMC_16	ADDR[7]	----->	A7	(pin 44)	
EMC_41	ADDR[8]	----->	A8	(pin 27)	
BO_00	ADDR[9]	----->	A9	(pin 26)	
BO_01	ADDR[10]	----->	A10	(pin 25)	
BO_02	ADDR[11]	----->	A11	(pin 24)	
EMC_19	WE_n	----->	WE_n	(pin 17)	
EMC_20	OE_n	----->	OE_n	(pin 39)	
EMC_41	CS0_n	----->	CE_n	(pin 6)	
EMC_08	DQM0	----->	LB_n	(pin 40)	



Bill of Materials (BOM) Example

For typical 16-bit SRAM interface:

Qty	Reference	Part Number	Description	Notes
1	U1	SOMRT1061-200IR	NetBurner module	Main processor
1	U2	IS61WV51216BLL	512Kx16 SRAM	External memory
2	C1, C2	0.1uF/50V/X7R	Decoupling capacitor 0603	At SRAM VCC pins
1	C3	10uF/16V/X7R	Bulk capacitor 0805	Near SRAM
16	R1-R16	33ohm 5% 0603	Series termination (optional)	Data lines only if needed

Checklist for Hardware Bring-Up

- ☐ Verify 3.3V present at SRAM VCC pins
- ☐ Verify GND connections with continuity test
- ☐ Check all data line connections (D0-D7, D8-D15 for 16-bit)
- ☐ Check all address line connections
- ☐ Verify WE_n connection
- ☐ Verify OE_n connection
- ☐ Verify CS_n connection
- ☐ Check ADV connection (for multiplexed mode)
- ☐ Verify byte enable connections (for 16-bit)
- ☐ Confirm no shorts between adjacent pins
- ☐ Measure power consumption (should be < 100mA typically)
- ☐ Scope signals to verify toggling
- ☐ Run software memory tests
- ☐ Verify timing with oscilloscope
- ☐ Test at temperature extremes if required

Additional Resources

- NXP i.MX RT1061 Reference Manual - Chapter on SEMC
- SRAM Device Datasheet - for specific timing requirements
- IPC-2221 - PCB design standards
- High-Speed Digital Design: A Handbook of Black Magic - by Howard Johnson

12.33.3.11 Analog to Digital (ADC)

Location: examples/PlatformSpecific/SOMRT1061/ADC_Simple

Supported Platforms: SOMRT1061

Simple ADC example implementing basic analog input functions of the SOMRT1061

12.33.3.12 SPI PicKit Demo Board

Location: examples/PlatformSpecific/SOMRT1061/PicKitSerialSPIDemoBd

Supported Platforms: MODM7AE70, SOMRT1061

Example demonstrating the 7 SPI peripherals on the Microchip PicKit Serial SPI Demo Board.

Demonstrates the use of 7 Microchip SPI Peripherals:

- 25LC020A 2K SPI Bus Serial EEPROM
- TC77-5.0 Thermal Sensor with SPI Interface
- MCP3201 2.7V 12-Bit A/D Converter with SPI Serial Interface
- MCP4822 12-Bit DAC with Internal VREF and SPI Interface
- MCP41010 Single/Dual Digital Potentiometer with SPI Interface
- MCP6S92 Single-Ended, Rail-to-Rail I/O, Low-Gain PGA
- MCP23S08 8-Bit I/O Expander with Serial Interface

12.33.3.12.1 PicKit SPI Demo Board Application

12.33.3.12.1.1 Overview

This application demonstrates the use of 7 Microchip SPI peripherals on the PicKit Serial SPI Demo Board. It provides a comprehensive example of interfacing with various SPI devices including sensors, storage, analog components, and I/O expanders.

12.33.3.12.1.2 Supported Platforms

- MODM7AE70
- SOMRT1061

12.33.3.12.1.3 SPI Peripherals Demonstrated

The application interfaces with the following 7 Microchip SPI peripherals:

1. **25LC020A** - 2K SPI Bus Serial EEPROM
2. **TC77-5.0** - Thermal Sensor with SPI Interface
3. **MCP3201** - 2.7V 12-Bit A/D Converter with SPI Serial Interface
4. **MCP4822** - 12-Bit DAC with Internal VREF and SPI Interface
5. **MCP41010** - Single/Dual Digital Potentiometer with SPI Interface
6. **MCP6S92** - Single-Ended, Rail-to-Rail I/O, Low-Gain PGA
7. **MCP23S08** - 8-Bit I/O Expander with Serial Interface

12.33.3.12.1.4 Key Features

EEPROM Operations

- Read/Write operations on 25LC020A 2K EEPROM
- Page-based writing (16-byte pages)
- Full memory erase functionality
- Incrementing sequence test patterns

Analog Processing

- **ADC:** 12-bit analog-to-digital conversion with voltage scaling
- **DAC:** Dual-channel 12-bit digital-to-analog conversion with programmable output
- **PGA:** Programmable gain amplifier with configurable gain settings (1x to 32x)

Temperature Sensing

- Real-time temperature measurement using TC77-5.0 sensor
- Temperature display in Celsius, Fahrenheit, and Kelvin
- Continuous conversion mode support

Digital Control

- **Potentiometer:** 256-position digital potentiometer control
- **GPIO Expander:** LED control via 8-bit I/O expander
- **User Interface:** Interactive serial menu system

12.33.3.12.1.5 Interactive Menu System

The application provides a user-friendly menu interface with the following options:

EEPROM Operations

- 1 - Erase EEPROM (fill with 0xFF)
- 2 - Read EEPROM (display hex dump)
- 3 - Write incrementing sequence to EEPROM

Peripheral Testing

- A - Read ADC value and display in volts
- D - DAC test (continuous output ramping)
- G - Programmable Gain Amplifier test
- T - Read temperature sensor
- P - Potentiometer test (user-configurable resistance)
- L - Toggle LEDs on GPIO expander

12.33.3.12.1.6 Technical Implementation

SPI Configuration

The application uses the DSPI (Deserial Serial Peripheral Interface) module with different configurations for each peripheral:

- **Clock frequencies:** 700 kHz to 5 MHz depending on device requirements
- **Clock polarity and phase:** Configured per device specifications
- **Chip select management:** Automatic and manual control modes

Memory Management

- Uses NetBurner RTOS semaphores for SPI bus synchronization
- Implements proper timing delays for device-specific requirements
- Handles endianness for multi-byte data transfers

Platform-Specific Features

- Conditional compilation for different hardware platforms
- Pin function configuration for various board layouts
- Platform-specific chip select and timing configurations

12.33.3.12.1.7 Pin Assignments

MODM7AE70

- SPI0_NPCS2: P2[30]
- SPI0_SPCK: P2[25]
- SPI0_MISO: P2[27]
- SPI0_MOSI: P2[28]

SOMRT1061

- LPSPI1_SCK: Pin 27
- LPSPI1_PCS0: Pin 26
- LPSPI1_SDO: Pin 25
- LPSPI1_SDI: Pin 24

12.33.3.12.1.8 Build Configuration

The application includes preprocessor directives for:

- Platform-specific pin assignments
- SPI module selection
- Chip select configuration
- Device-specific timing parameters

12.33.3.12.1.9 Usage Notes

1. **Temperature Sensor:** Requires tri-state configuration of MOSI line for proper operation
2. **EEPROM:** Implements write-in-progress polling for reliable data storage
3. **DAC:** Supports dual-channel operation with independent voltage control
4. **Safety:** All SPI operations include proper semaphore handling and error checking

12.33.3.12.1.10 Error Handling

The application includes comprehensive error handling:

- SPI transaction completion verification
- Device presence detection (TC77 ID check)
- Parameter validation for user inputs
- Timeout handling for write operations

12.33.3.13 On-Chip Temperature Sensor (TEMPMON)

Location: examples/PlatformSpecific/SOMRT1061/TempSensor1061

Supported Platforms: SOMRT1061

Initialize and read the i.MX RT1061 on-chip temperature sensor, referred to as Temperature Monitor (TEMPMON) in the processor documentation.

Note

Also see the SOMRT1061 Development Board temperature sensor example

12.33.3.14 Dev Board Temperature Sensor and ADC

Location: examples/PlatformSpecific/SOMRT1061/TempSensorDevBoard

Supported Platforms: SOMRT1061

Initialize and read the temperature sensor on the NetBurner DEV-SOMRT1061 Development Board. The sensor is connected to ADC channel 1.

Also reads other ADC devices on the Development Board, like the potentiometer on ADC channel 0, and anything connected to channels 2+.

Note

Also see the Temperature Monitor example for the i.MX RT1061 processor on-chip temperature sensor (TEMPMON)

12.34 PPP

12.34.1 PPP Server Application

12.34.1.1 Overview

This NetBurner embedded application demonstrates how to create and manage Point-to-Point Protocol (PPP) server instances. The PPP protocol enables network connectivity over serial connections, supporting both direct serial connections and modem-based connections for TCP/IP networking.

12.34.1.2 Features

- **Dual PPP Server Support:** Creates two independent PPP server instances (PPP1 and PPP2)
- **Flexible Connection Types:** Supports both direct serial and modem-based PPP connections
- **Authentication System:** Includes customizable username/password authentication
- **Network Integration:** Full TCP/IP stack integration with HTTP server support
- **Interactive Command Interface:** Runtime control and monitoring via serial console
- **Network Diagnostics:** Built-in ping functionality and connection status monitoring
- **IPv4/IPv6 Support:** Comprehensive IP address management and display utilities

12.34.1.3 Hardware Requirements

- NetBurner embedded device with serial port support
- Serial cable or modem connection
- Host computer with PPP client capabilities

12.34.1.4 Connection Types

12.34.1.4.1 Direct Connection

- Uses only a serial cable connection between PC and NetBurner device
- Immediately begins PPP negotiations with the host/client
- No modem initialization required
- Ideal for direct PC-to-NetBurner communication

12.34.1.4.2 Modem Connection

- Uses a modem between the PC and NetBurner device
- Sends AT commands for modem initialization before PPP negotiation
- Supports dial-up and answer modes
- Handles carrier detection and modem state management

12.34.1.5 Configuration

12.34.1.5.1 Default Settings

- **Serial Port:** Port 1 (configurable via `SERIALPORT_TO_USE`)
- **Baud Rate:** 19,200 bps (configurable via `BAUD_TO_USE`)
- **NetBurner IP:** 192.1.1.1
- **Gateway IP:** 192.1.1.2
- **DNS Server:** 8.8.8.8

12.34.1.5.2 Network Configuration Example

```
pppServerObj.ip4.addr = AsciiToIp4("192.1.1.1");
pppServerObj.ip4.gate = AsciiToIp4("192.1.1.2");
pppServerObj.OfferedDNSAddress = AsciiToIp4("8.8.8.8");
```

12.34.1.6 Authentication

The application includes a sample authentication function `test_Auth()` that currently accepts any user-name/password combination. **For production use, replace this with proper credential validation:**

```
int test_Auth(const char *name, const char *pass)
{
    printf("Auth %s:%s\r\n", name, pass);
    return 1; // Always returns success - modify for real authentication
}
```

12.34.1.7 Interactive Commands

Once the PPP connection is established, the application provides an interactive menu accessible via serial console:

- **C** - Show network counters and statistics
- **H** - Force hangup/disconnect the PPP session
- **P** - Ping the remote host to test connectivity
- **S** - Show detailed PPP connection status
- **X** - Exit the application
- **?** - Display help menu
- **F** - Show interface information (debug builds)
- **L** - Set log level (debug builds)

12.34.1.8 PPP State Machine

The application monitors and displays various PPP connection states:

State	Description
eClosed	Connection closed
eInitializingModem	Initializing modem (modem connections only)
eDialing	Dialing remote number
eWait4Ring	Waiting for incoming call
eAnswering	Answering incoming call
eWaitForTrain	Waiting for carrier signal
eLCPNegotiate	Link Control Protocol negotiation
ePAPAuthenticate	Password Authentication Protocol
eNCPNegotiate	Network Control Protocol negotiation
eOpen	Connection established and active
eClosing	Connection closing

12.34.1.9 Error Handling

The application includes error handling for common PPP issues:

Error Code	Description
ERR_PPP_ALREADY_OPEN	PPP interface already active
ERR_PPP_NO_DIALTONE	No dial tone detected
ERR_PPP_NO_ANSWER	No answer from remote
ERR_PPP_BUSY	Remote line busy
ERR_PPP_FAIL	General PPP failure
ERR_PPP_PASSFAIL	Authentication failure
ERR_PPP_LOSTCARRIER	Lost carrier signal
ERR_PPP_NO_MODEM	Modem not responding
ERR_PPP_LCP_FAILED	LCP negotiation failed

12.34.1.10 IP Address Utilities

The application includes IP address management utilities:

12.34.1.10.1 IPv4 Features

- Display IP, subnet mask, gateway, and DNS server information
- Interface MAC address display
- Link status monitoring (speed and duplex)
- Web-based interface display with clickable links

12.34.1.10.2 IPv6 Features (when enabled)

- Multiple address assignment support (Link Local, Router, [DHCP](#), Static)
- [DHCP](#) lease time information with renewal/rebind timers
- DNS server information display
- Prefix length and source identification

12.34.1.11 Web Interface

The application includes a built-in HTTP server that provides:

- Web-based display of all network interface information
- Clickable links to test IPv4 and IPv6 connectivity
- Real-time interface status and configuration display
- **DHCP** lease time information for IPv6 addresses

12.34.1.12 File Structure

```
project/  
main.cpp           # Main application logic and PPP server management  
ip_util.cpp        # IP address utility functions  
ip_util.h          # IP utility function declarations
```

12.34.1.13 Build Requirements

- NetBurner development environment
- Serial communication support libraries
- TCP/IP stack libraries
- HTTP server support

12.34.1.14 Usage Instructions

1. Initial Setup

- Connect NetBurner device to host via serial cable or modem
- Configure serial port settings (default: Port 1, 19200 baud)
- Build and deploy application to NetBurner device

2. Starting PPP Connection

- Application automatically initializes network stack
- Starts HTTP server on port 80
- Waits for network initialization
- Opens serial port and starts PPP server

3. Host Configuration

- Configure PPP client on host computer
- Set up serial connection with matching baud rate
- Initiate PPP connection from host

4. Testing Connection

- Use interactive menu commands to monitor connection
- Test connectivity with built-in ping function
- Access web interface at configured IP address

12.34.1.15 Testing PPP Connections

Recommended testing approach:

1. Create a PPP interface on a host computer
2. Connect via serial cable to the NetBurner device
3. Configure host PPP client with appropriate settings
4. Establish direct PPP connection
5. Use interactive commands to monitor connection status
6. Verify network connectivity using built-in ping function
7. Test web interface accessibility

12.34.1.16 Development Notes

12.34.1.16.1 Production Considerations

- Replace sample authentication function with secure credential checking
- Disable system diagnostics in production builds
- Configure appropriate IP addresses for your network topology
- Implement connection retry logic for unreliable connections
- Monitor and log PPP events for troubleshooting

12.34.1.16.2 Debug Features

- System diagnostics enabled for debugging
- Comprehensive logging and error reporting
- Interactive command interface for real-time monitoring

12.34.1.16.3 Network Integration

- Full TCP/IP stack support
- HTTP server integration
- Multi-interface support with status monitoring

12.34.1.17 Troubleshooting

12.34.1.17.1 Common Issues

1. **Connection Fails:** Check serial cable connections and baud rate settings
2. **Authentication Errors:** Verify username/password if custom authentication is implemented
3. **No Network Access:** Confirm IP address configuration and routing
4. **Modem Issues:** Check AT command compatibility and modem initialization

12.34.1.17.2 Debug Commands

- Use 'S' command to check PPP state
- Use 'C' command to view network counters
- Use 'P' command to test connectivity
- Check web interface for detailed network information

12.35 Profiler

12.35.1 Profiler Example Application

12.35.1.1 Overview

This is a simple profiler program designed for the NBRTOS (NetBurner Real-Time Operating System) that demonstrates task profiling capabilities. The application provides a menu-driven interface to monitor task execution times and system performance metrics.

12.35.1.2 Features

- **Task Time Monitoring:** Track execution time for individual tasks
- **System Statistics:** Display total system tick counts and timing information
- **Real-time Counter:** Shows a continuously updating counter display
- **Web Server Integration:** Includes HTTP server functionality
- **Network Configuration:** Automatic [DHCP](#) setup and network interface management

12.35.1.3 Profiling Functionality

The NBRTOS profiling system works by recording tick counts and tick fractions during task switching events. This allows the system to track how much CPU time each task consumes.

12.35.1.3.1 Key Profiling Functions

- `GetCurrentTaskTime(uint32_t *TotalTicks)` - Returns the number of ticks the current task has run
- `ShowTaskTimes()` - Prints a list of all tasks with their execution times and percentages
- `ClearTaskTimes()` - Resets all task timing counters to zero

12.35.1.4 Menu Commands

The application provides an interactive menu system with the following commands:

- **C** - Get statistics on the current task (execution time and percentage)
- **S** - Show OS seconds counter and tick information
- **T** - Display statistics for all tasks in the system
- **?** - Display the main menu

12.35.1.5 Setup and Configuration

12.35.1.5.1 Enabling Profiling

Profiling functionality requires the system to be built with `NBRTOS_TIME` defined. There are two ways to enable this:

12.35.1.5.1.1 Method 1: Using Overload Folder (Recommended)

Copy the provided overload folder into your application directory. This method is preferred as it only affects your specific application.

12.35.1.5.1.2 Method 2: System-wide Configuration

1. Edit `\nburn\nbrtos\include\predef.h`
2. Uncomment the line: `#define NBRTOS_TIME(1)`
3. Rebuild the system libraries

Note**: Method 2 affects all applications and adds overhead to the entire system.

12.35.1.5.2 Important Considerations

- Profiling adds computational overhead and should typically only be used during development
- The profiling feature is disabled by default for performance reasons
- When profiling is disabled, attempting to use profiling commands will display a warning message

12.35.1.6 Application Flow

1. **Initialization:** The application initializes the network stack and starts the HTTP server
2. **Network Setup:** Waits for `DHCP` completion and displays IP address information
3. **Main Loop:** Continuously checks for user input while displaying an updating counter
4. **Command Processing:** Handles user commands through a simple dispatcher system

12.35.1.7 Network Features

- **DHCP Support:** Automatic IP address configuration
- **Web Server:** Built-in HTTP server on port 80
- **Interface Monitoring:** Displays network interface information
- **Connection Status:** Shows current IP addresses and network configuration

12.35.1.8 Development Notes

- The application includes system diagnostics that should be removed for production code
- The counter display uses backspace characters to create an updating effect
- Error handling is included for cases where profiling features are not available
- The application waits up to 5 seconds for network activation

12.35.1.9 Output Examples

When profiling is enabled, you can expect output like:

```
Current Task Ticks: 12345, Total System Ticks: 100000, Percent: 12.35
```

```
Tick Count = 0x1A2B3C = 1715004 (171 seconds)
```

12.35.1.10 Compilation Requirements

- NBRTOS development environment
- Network libraries (ethervars, ipshow)
- Standard I/O system libraries (iosys)
- Initialization libraries (init)

12.35.1.11 Usage

1. Compile and flash the application to your NetBurner device
2. Connect via serial terminal
3. Wait for network initialization to complete
4. Use the menu commands to monitor system performance
5. Press any key after each command display to return to the menu

12.36 RTOS

NetBurner RTOS (NBRTOS) Examples:

- [MaxTasks](#)
- [OSCrit - Critical Section](#)
- [OSFifo - FIFO](#)
- [OSFlags](#)
- [OSMailbox](#)
- [exampleRTOS-OSMultiTaskUserInput](#)
- [OSQueue](#)
- [OSSemaphore](#)
- [OSTaskCreate](#)
- [Round Robin Task Sheduler](#)

12.36.1 MaxTasks

12.36.1.1 MaxTasks RTOS Example

12.36.1.1.1 Overview

This example demonstrates how to increase the maximum number of system tasks beyond the default limits in the NBRTOS (NetBurner Real-Time Operating System). The application creates multiple tasks that execute in a sequential chain pattern, showcasing the system's ability to handle an increased task count.

12.36.1.1.2 Application Description

The MaxTasks example creates 64 additional tasks beyond the standard system tasks. These tasks operate in a coordinated sequence where each task:

1. Waits for its turn using a semaphore
2. Introduces a short delay for readable output
3. Prints its task information (name, priority, system tick)
4. Signals the next task in the sequence
5. Returns to sleep

The tasks execute in reverse priority order, creating a continuous cycle that demonstrates the system's task scheduling capabilities.

12.36.1.1.3 Key Features

- **Task Creation:** Creates 64 extra tasks with unique names and priorities
- **Sequential Execution:** Tasks execute in a coordinated chain pattern
- **Status Reporting:** Each task reports its name, priority, and current system tick
- **Error Handling:** Provides feedback on task creation success/failure
- **Network Integration:** Includes network stack initialization and [DHCP](#) support

12.36.1.1.4 Configuration

12.36.1.1.4.1 System Limits

The application requires modification of default RTOS limits through the `constants-overload.h` file:

```
#define OS_MAX_TASKS (64)    // Max number of system tasks
#define OS_MAX_PRIOS (128)  // Maximum number of system priorities
```

Note

The application project must have the constants file located in the project's `overflow` folder with the same path as the where the constants file is normally found, `\nbrtos\include\constants-overload.h`.

The entire project path for constants is then: `\MaxTasks\overflow\nbrtos\include\constants-overload.h`.

12.36.1.1.4.2 Task Configuration

- **Extra Task Count:** 64 tasks (configurable via `EXTRA_TASK_COUNT`)
- **Base Priority:** Starts at `MAIN_PRIO + 1` (configurable via `EXTRA_TASK_BASE_PRIO`)
- **Stack Size:** Each task uses `USER_TASK_STK_SIZE` stack space
- **Task Names:** Auto-generated as "ExtraTask_X" where X is the task index

12.36.1.1.5 Performance Considerations

12.36.1.1.5.1 Hard Limits

- Maximum of 256 task priorities in NBRRTOS
- Priority values stored in `uint8_t` format
- Limited by `OSTaskID()` return type

12.36.1.1.5.2 Performance Impact

Increasing maximum task priorities affects system performance in several ways:

1. **Memory Usage:** All OS control objects (`OS_CRIT`, `OS_SEM`, `OS_FIFO`, `OS_MBOX`) must grow to accommodate additional priority levels
2. **Processing Overhead:**
 - Additional scanning required for priority bit checking
 - Increased memory bandwidth requirements
 - Greater cache space utilization
3. **Memory Alignment:** Default 16-byte OS control objects lose power-of-two alignment, potentially spanning multiple cache lines

12.36.1.1.5.3 Recommendation

A caution regarding system limits and performance. There is a hard limit of 256 task priorities in the NBRTOS. This is a function of storing the task priority in a `uint8_t` in the `OS_TCB` and the return type of `OSTaskID()`.

As for the impact on system performance, when you increase the maximum task priorities, the OS must be able to handle the additional priority values. To do so, all task priority lists used in various OS related objects (i.e. `OS_CRIT`) have to grow in size. This not only means more memory usage, but, more fundamentally, more processing overhead. The processing overhead is twofold:

1. Scanning additional storage for bits being set.
2. The memory bandwidth and cache space associated with having to read more memory.

By default, most OS control objects (`OS_CRIT`, `OS_SEM`, `OS_FIFO`, `OS_MBOX`) are 16 bytes. This fits nicely from a memory alignment perspective as it's a power of two. By increasing the max priority value, it increases the size and breaks that symmetry, increasing the odds of the structure spanning multiple cache lines. Given this, the number of priorities should not be increased unless absolutely necessary.

12.36.1.1.6 File Structure

```
main.cpp           # Main application code
constants-overload.h # System limit overrides
```

12.36.1.1.7 Code Structure

12.36.1.1.7.1 Main Components

1. **Task Arrays:** Pre-allocated storage for stacks, names, and semaphores
2. **ExtraTask():** The common task function executed by all extra tasks
3. **UserMain():** Application entry point that creates and manages tasks
4. **Validation:** Compile-time checks to ensure configuration consistency

12.36.1.1.7.2 Task Synchronization

The application uses an array of semaphores (`ExtraTaskSem[]`) to coordinate task execution. Each task:

- Waits on its own semaphore
- Posts to the previous task's semaphore (creating reverse-order execution)
- The cycle is primed by posting to the first task's semaphore

12.36.1.1.8 Build Requirements

- NetBurner NNDK (NetBurner Network Development Kit)
- NBRTOS support
- Network stack libraries

12.36.1.1.9 Usage

1. Configure the desired number of extra tasks by modifying `EXTRA_TASK_COUNT`
2. Ensure `constants-overload.h` has appropriate `OS_MAX_TASKS` and `OS_MAX_PRIOS` values
3. Compile and deploy to NetBurner hardware
4. Monitor serial output to observe task execution sequence

12.36.1.1.10 Output Format

Each task prints information in the format:

```
[TaskName][Priority]: [SystemTick]
```

Example output:

```
ExtraTask_0[ 21]: 1234
ExtraTask_63[ 84]: 1235
ExtraTask_62[ 83]: 1236
...
```

12.36.1.1.11 Error Handling

The application includes comprehensive error checking:

- Compile-time validation of configuration limits
- Runtime reporting of task creation success/failure
- Network initialization timeout handling

12.36.1.1.12 Customization

To modify the application:

- Change `EXTRA_TASK_COUNT` to create more/fewer tasks
- Adjust `EXTRA_TASK_BASE_PRIO` to change starting priority
- Modify timing delays in `ExtraTask()` and `UserMain()`
- Update task naming convention in the creation loop

12.36.1.1.13 Warning

This example is primarily for demonstration and testing purposes. In production systems, carefully consider the performance implications of increasing task limits and only do so when the additional tasks are essential for your application's functionality.

12.36.2 OSCrit - Critical Section

12.36.2.1 OSCrit - Critical Section Example

12.36.2.1.1 Overview

This example demonstrates how to implement critical sections in a real-time operating system (RTOS) environment using two different approaches: traditional C-style function calls and modern C++ object-oriented design.

12.36.2.1.2 Purpose

Critical sections are essential for protecting shared resources in multi-tasking environments. They ensure that only one task can access a shared resource at a time, preventing race conditions and data corruption that could occur when multiple tasks attempt to modify the same data simultaneously.

12.36.2.1.3 Application Description

The OSCrit example (`main.cpp`) showcases both methods of implementing critical sections:

12.36.2.1.3.1 Method 1: Traditional C-Style Critical Sections

Uses explicit function calls:

- `OSCritEnter()` - Enters the critical section
- `OSCritLeave()` - Exits the critical section

```
OS_CRIT cs;
cs.Enter();
GlobalVariable = 1;
cs.Leave();
```

12.36.2.1.3.2 Method 2: C++ Object-Oriented Critical Sections

Uses RAIL (Resource Acquisition Is Initialization) pattern:

- Constructor automatically calls `OSCritEnter()`
- Destructor automatically calls `OSCritLeave()` when object goes out of scope

```
OS_CRIT cs2;
{
    OSCriticalSectionObj UserMainCriticalSection(cs2);
    GlobalVariable = 2;
    // Automatic cleanup when scope ends
}
```

12.36.2.1.4 Key Features

- **Network Initialization:** Waits for [DHCP](#) address assignment (5-second timeout)
- **System Diagnostics:** Enables system diagnostic features for debugging
- **Shared Resource Protection:** Demonstrates protection of a global variable
- **Timeout Handling:** Shows how to configure timeout values for critical section acquisition
- **Automatic Resource Management:** C++ approach eliminates the risk of mismatched enter/leave calls

12.36.2.1.5 Advantages of Each Approach

12.36.2.1.5.1 Traditional C-Style

- **Explicit Control:** Clear visibility of when critical sections are entered and exited
- **Timeout Configuration:** Allows custom timeout values for critical section acquisition

12.36.2.1.5.2 C++ Object-Oriented

- **Automatic Management:** No risk of forgetting to call `OSCritLeave()`
- **Exception Safety:** Guaranteed cleanup even if exceptions occur
- **Scope-Based:** Critical section is automatically released when leaving the scope

12.36.2.1.6 Technical Details

- **Application Name:** "OSCrit Example"
- **Shared Resource:** `GlobalVariable` (integer)
- **Network Timeout:** 5 seconds for [DHCP](#) acquisition
- **Main Loop:** 5-second delay iterations
- **Timeout Behavior:** Value of 0 means wait forever for critical section access

12.36.2.1.7 Use Cases

While this example uses a simple global variable for demonstration, critical sections are typically used for:

- **Linked Lists:** Protecting list modifications during insertions/deletions
- **Shared Buffers:** Ensuring atomic access to circular buffers or queues
- **Hardware Resources:** Coordinating access to peripherals or memory-mapped registers
- **Data Structures:** Protecting complex objects that require multiple operations to remain consistent

12.36.2.1.8 Best Practices Demonstrated

1. **Proper Initialization:** System and network setup before critical operations
2. **Resource Protection:** Using critical sections for shared data access
3. **Modern C++ Techniques:** Leveraging RAI for automatic resource management
4. **Comparison Approach:** Showing both traditional and modern implementation methods

12.36.2.1.9 Implementation Notes

- The second parameter in critical section calls represents timeout in system ticks
- A timeout value of 0 means the task will wait indefinitely
- The C++ approach always uses infinite timeout (wait forever)
- System diagnostics should be disabled in production code for optimal performance

12.36.3 OSFifo - FIFO

12.36.3.1 RTOS FIFO Example Application

12.36.3.1.1 Overview

This application demonstrates the use of a First-In-First-Out (FIFO) queue in a Real-Time Operating System (RTOS) environment. The program creates two tasks that communicate with each other through a FIFO buffer, showcasing inter-task communication and timeout handling.

12.36.3.1.2 Application Description

The OSFifo example creates a producer-consumer pattern where:

- One task (FifoPostTask) posts messages to a FIFO queue
- Another task (UserMain) reads messages from the FIFO queue
- A timeout mechanism is implemented to demonstrate FIFO timeout features

12.36.3.1.2.1 Key Components

Data Structure

```
typedef struct {
    void *pUsedByFifo;    // Internal FIFO linked list pointer (do not modify)
    int x;                // Data payload
    BOOL free;            // Availability flag
} MyStructure;
```

FIFO Management

- **FIFO Object:** `OS_FIFO MyFIFO` - The main FIFO queue
- **Structure Array:** `MyStructure StructArray[NUM_POSTS]` - Pre-allocated message pool
- **Memory Management:** Uses static allocation instead of dynamic memory

12.36.3.1.3 Program Flow

12.36.3.1.3.1 Initialization Phase

1. Initialize the FIFO object
2. Initialize all structures in the array as "free"
3. Create the FIFO posting task
4. Enable system diagnostics

12.36.3.1.3.2 Runtime Operation

Producer Task (FifoPostTask)

- Runs continuously in a loop
- Posts NUM_POSTS (5) messages per cycle
- For each message:
 - Finds a free structure in the array
 - Sets the data value (x) to the current count
 - Marks the structure as "in use"
 - Posts the structure pointer to the FIFO
- Waits 9 seconds between posting cycles

Consumer Task (UserMain)

- Continuously pends on the FIFO with a 5-second timeout
- When a message is received:
 - Prints the received data value
 - Marks the structure as "free" for reuse
- When timeout occurs:
 - Prints a timeout message
 - Continues waiting for the next message

12.36.3.1.4 Timeout Demonstration

The application is designed to demonstrate FIFO timeout behavior:

- The producer posts 5 messages quickly, then waits 9 seconds
- The consumer has a 5-second timeout on FIFO pend operations
- After consuming all 5 messages, the consumer will timeout once before the next batch arrives

12.36.3.1.5 Key Features

12.36.3.1.5.1 Memory Management

- Uses static memory allocation instead of dynamic allocation
- Implements a simple free-list mechanism for structure reuse
- Prevents memory leaks through proper structure lifecycle management

12.36.3.1.5.2 Error Handling

- Checks for task creation errors
- Handles FIFO timeout conditions gracefully
- Validates structure availability before posting

12.36.3.1.5.3 Synchronization

- Demonstrates safe inter-task communication using FIFO
- Shows proper handling of resource contention
- Illustrates timeout-based resource management

12.36.3.1.6 Configuration

- `NUM_POSTS`: Number of structures to post per cycle (currently 5)
- `USER_TASK_STK_SIZE`: Stack size for the posting task
- `TICKS_PER_SECOND`: System tick rate for timing calculations
- `MAIN_PRIO - 1`: Priority level for the posting task

12.36.3.1.7 Expected Output

The application will display:

1. FIFO posting messages showing structure index and data value
2. FIFO reading messages showing read count and data value
3. Periodic timeout messages during the 9-second delay periods
4. Delay notifications before each new posting cycle

12.36.3.1.8 Usage Notes

- This example is designed for educational purposes to demonstrate FIFO usage
- The timeout feature illustrates how to handle blocking operations with time limits
- The static memory management approach is suitable for resource-constrained embedded systems
- System diagnostics are enabled for debugging (should be disabled in production)

12.36.3.1.9 Technical Requirements

- NetBurner RTOS environment
- Standard C++ compiler support
- RTOS task creation and timing services
- FIFO queue implementation (`OS_FIFO` class)

12.36.4 OSFlags

12.36.4.1 OSFlags RTOS Example

12.36.4.1.1 Overview

This example demonstrates how to use OSFlags in a Real-Time Operating System (RTOS) environment to coordinate multiple tasks through event signaling. The application creates multiple tasks that set flags at different intervals, while a main task waits for these events to occur.

12.36.4.1.2 Application Description

The OSFlags example illustrates inter-task communication using event flags. It creates three worker tasks that operate on different time schedules and set corresponding flags. The main task monitors these flags and responds when any (or optionally all) of the flags are set.

12.36.4.1.2.1 Key Components

- **Global OS Flag Object:** `gOsFlag` - A shared flag object used for inter-task communication
- **Three Worker Tasks:** Each task sets a specific flag after a time delay
- **Main Monitoring Task:** Waits for flag events and processes them

12.36.4.1.3 Task Structure

12.36.4.1.3.1 Worker Tasks

1. OsFlagTask1

- Priority: MAIN_PRIO + 1
- Delay: 2 seconds
- Sets: TASK_FLAG_1 (bit 0x00000001)

2. OsFlagTask2

- Priority: MAIN_PRIO + 2
- Delay: 4 seconds
- Sets: TASK_FLAG_2 (bit 0x00000002)

3. OsFlagTask3

- Priority: MAIN_PRIO + 3
- Delay: 6 seconds
- Sets: TASK_FLAG_3 (bit 0x00000004)

12.36.4.1.3.2 Main Task (UserMain)

The main task performs the following operations:

1. Initializes the system and network
2. Creates the three worker tasks
3. Enters an infinite loop that:
 - Waits for any flag to be set using `PendAny()`
 - Identifies which flags are currently set
 - Prints the status of active flags
 - Clears the processed flags

12.36.4.1.4 Flag Operations

12.36.4.1.4.1 Pending Modes

The application supports two different pending modes:

- **PendAny()**: Returns when one or more flags are set (default behavior)
- **PendAll()**: Returns only when all specified flags are set (commented out)

12.36.4.1.4.2 Flag Management

- **Setting Flags**: Each worker task sets its assigned flag using `gOsFlag.Set()`
- **Reading Flags**: The main task reads flag state using `gOsFlag.State()`
- **Clearing Flags**: Processed flags are cleared using `gOsFlag.Clear()`

12.36.4.1.5 Expected Behavior

The application will produce output showing which tasks have set their flags:

```
OSFlagPendAny() detected the following flag(s) are set: TASK1
OSFlagPendAny() detected the following flag(s) are set: TASK2
OSFlagPendAny() detected the following flag(s) are set: TASK1 TASK3
```

The timing pattern will be:

- Task1 flag every 2 seconds
- Task2 flag every 4 seconds
- Task3 flag every 6 seconds

12.36.4.1.6 Configuration Options

12.36.4.1.6.1 Modifying Pending Behavior

To change from "any flag" to "all flags" behavior:

1. Comment out the `PendAny()` call
2. Uncomment the `PendAll()` call

This will cause the main task to wait until all three flags are set simultaneously before proceeding.

12.36.4.1.6.2 Timing Adjustments

Task delays can be modified by changing the multiplier values:

- `OSTimeDly((2 * TICKS_PER_SECOND))` - 2 second delay
- `OSTimeDly((4 * TICKS_PER_SECOND))` - 4 second delay
- `OSTimeDly((6 * TICKS_PER_SECOND))` - 6 second delay

12.36.4.1.7 System Requirements

- RTOS environment with OSFlags support
- Network capability ([DHCP](#))
- System diagnostics support
- Multi-tasking support

12.36.4.1.8 Key Learning Points

This example demonstrates:

1. **Inter-task Communication:** Using flags to coordinate between tasks
2. **Event-Driven Programming:** Tasks respond to events rather than polling
3. **Bitwise Operations:** Flag manipulation using bitmasking
4. **Task Priorities:** Different priority levels for worker tasks
5. **Synchronization:** Coordinating multiple asynchronous tasks

12.36.4.1.9 Usage Notes

- The application waits for network initialization ([DHCP](#)) before starting
- System diagnostics are enabled (should be disabled for production)
- Flag clearing is important to prevent false triggers
- Only one pending function (`PendAny` or `PendAll`) can be active at a time

12.36.5 OSMailbox

12.36.5.1 RTOS Mailbox Example

12.36.5.1.1 Overview

This application demonstrates inter-task communication using mailboxes in a real-time operating system (RTOS). The program creates two tasks that communicate through a shared mailbox mechanism, showcasing how tasks can send and receive messages in a multi-threaded environment.

12.36.5.1.2 Application Description

The **Mailbox Example** application consists of two main tasks:

1. **UserMain Task** - The primary task that waits for messages from other tasks
2. **MyTask** - A secondary task that periodically sends messages to the mailbox

12.36.5.1.2.1 Program Flow

1. The application initializes the system and enables network connectivity
2. A mailbox object (`MyMailbox`) is created and initialized
3. Two tasks are created with different priorities:
 - `UserMain` runs at main priority level
 - `MyTask` runs at main priority + 1 (higher priority)
4. The tasks communicate through the mailbox using a producer-consumer pattern

12.36.5.1.2.2 Task Behavior

UserMain Task

- Continuously waits (pends) on the mailbox for incoming messages
- When a message is received, it prints the message value
- Increments the received message value before the next iteration
- Handles error conditions when message reception fails

MyTask

- Runs in an infinite loop with a 3-second delay between iterations
- Posts a message containing a `uint8_t` value to the mailbox
- Prints status information when posting messages

12.36.5.1.3 Key Features

- **Inter-task Communication:** Demonstrates safe message passing between tasks
- **Blocking Operations:** Shows how tasks can block waiting for messages
- **Error Handling:** Includes proper error checking for mailbox operations
- **Debug Output:** Provides serial port output for monitoring program execution
- **Network Integration:** Waits for active network before starting main operations

12.36.5.1.4 Technical Details

12.36.5.1.4.1 Data Structures

- `OS_MBOX MyMailbox` - The mailbox object for inter-task communication
- `uint32_t MyTaskStack[USER_TASK_STK_SIZE]` - Stack allocation for MyTask
- `uint8_t myMsg` - The message data passed between tasks

12.36.5.1.4.2 RTOS Functions Used

- `OSTimeDly()` - Task delay function
- `MyMailbox.Init()` - Mailbox initialization
- `MyMailbox.Post()` - Send message to mailbox
- `MyMailbox.Pend()` - Wait for message from mailbox
- `OSSimpleTaskCreateWithName()` - Task creation with naming

12.36.5.1.4.3 System Functions

- `init()` - System initialization
- `EnableSystemDiagnostics()` - Enable system diagnostic features
- `WaitForActiveNetwork()` - Wait for network connectivity

12.36.5.1.5 Output

The program outputs status messages through the serial debug port, which can be monitored using terminal programs like MTTTY. The output shows:

- When UserMain is waiting for messages
- When MyTask posts messages to the mailbox
- The received message values
- Any error conditions that occur

12.36.5.1.6 Build Requirements

- RTOS system with mailbox support
- Serial debug port capability
- Network stack (optional, used for initialization timing)

12.36.5.1.7 Usage

1. Compile and flash the program to your target device
2. Connect a serial terminal to the debug port
3. Monitor the output to observe the mailbox communication pattern
4. The program runs indefinitely, demonstrating continuous inter-task messaging

12.36.6 OSQueue

Queue Example Program

This program creates two tasks and a Queue. Messages are sent from one task to another using the Queue. A timeout value for the pend function is used to illustrate the timeout feature of the FIFO.

12.36.7 OSSemaphore

This program will create 2 tasks and a semaphore. The UserMain task will block for a semaphore posted from My↔ Task. The output for this example is displayed through the serial debug port, which can be viewed with the MTTTY program.

12.36.8 OSTaskCreate

This example demonstrates how to use two of the task creation functions:

- OSSimpleTaskCreatewName()
- OSTaskCreatewName()

This example demonstrates how to use create tasks and wait for task completion:

- OSTaskCreatewName()
- OSTaskJoin()

12.36.9 Round Robin Task Scheduler

This example demonstrates how to use the Priority Promotion optional behavior to implement a round robin scheduler for a set of worker tasks. This ends up using N+1 tasks to implement a set of N tasks.

The scheduler implemented is not 100% efficient at time allocation, but will guarantee a minimum time slice based on system Ticks, until the running task completes its iteration. In the event that the task being given the timeslice blocks, the tasks will end up falling through and being scheduled as normal, based on their relative priorities.

In the event that no task is ready, the 'next' task will be promoted to the scheduler's priority, at which point it will be blocked, then execution will fall through to whatever tasks may exist at lower priorities, all the way to System Idle.

12.37 Serial

Serial Port Examples:

- [Dual TCP To Serial](#)
- [Serial To Ethernet](#)
- [Serial HTTP Get Request](#)
- [Serial Receive Callback](#)
- [Serial to Serial](#)
- [Serial Application Update](#)
- [TCP to Serial](#)

12.37.1 Dual TCP To Serial

12.37.1.1 DualTCP2Serial Example

A NetBurner embedded application that bridges TCP network connections to serial ports, enabling remote access to serial devices over Ethernet.

12.37.1.1.1 Overview

This application creates TCP-to-serial bridges on two serial ports simultaneously, allowing network clients to communicate with serial devices as if they were directly connected. Each serial port listens on a different TCP port and handles one client connection at a time.

12.37.1.1.2 Features

- **Dual Serial Port Support:** Bridges both serial port 0 and serial port 1
- **Bidirectional Data Transfer:** Data flows seamlessly between TCP clients and serial devices
- **Connection Override:** New clients can override existing connections after a timeout period
- **Configurable Parameters:** Easily adjust baud rates, TCP ports, and timeout values
- **Error Handling:** Robust handling of network and serial port errors

12.37.1.1.3 Configuration

12.37.1.1.3.1 Default Settings

Parameter	Port 0	Port 1
TCP Port	2000	2001
Baud Rate	115200	115200
Data Bits	8	8
Stop Bits	1	1
Parity	None	None
Override Timeout	10 seconds	10 seconds

12.37.1.1.3.2 Customization

Modify these constants in `main.cpp` to change the configuration:

```
#define BAUDRATE_TO_USE (115200) // Baud rate for both serial ports
#define STOP_BITS (1) // Number of stop bits
#define DATA_BITS (8) // Number of data bits
#define TCP_PORT0_TO_USE (2000) // TCP port for serial port 0
#define TCP_PORT1_TO_USE (2001) // TCP port for serial port 1
#define OVERRIDE_TIMEOUT (TICKS_PER_SECOND * 10) // Override timeout
```

12.37.1.1.4 How It Works

12.37.1.1.4.1 Connection Process

1. **Startup:** Application initializes both serial ports and begins listening on configured TCP ports
2. **Client Connection:** When a TCP client connects, a dedicated bridge is established
3. **Data Transfer:** All data received from the network is sent to the serial port, and vice versa
4. **Connection Management:** Only one client per serial port at a time

12.37.1.1.4.2 Override Feature

The override timeout prevents stale connections from blocking new clients:

- When a client connects, the listen socket is closed to prevent additional connections
- After the timeout period expires, a new listen socket opens
- If a new client attempts to connect, the old connection is automatically closed
- This handles cases where clients crash or disconnect improperly

12.37.1.1.5 Usage

12.37.1.1.5.1 Network Client Connection

Connect to the NetBurner device using any TCP client:

```
# Connect to serial port 0
telnet <netburner_ip> 2000

# Connect to serial port 1
telnet <netburner_ip> 2001
```

12.37.1.1.5.2 Example Use Cases

- **Remote Serial Console:** Access device debug output over network
- **Serial Device Control:** Control industrial equipment remotely
- **Data Logging:** Collect serial data from remote sensors
- **Protocol Bridging:** Convert between network and serial protocols

12.37.1.1.6 Architecture

12.37.1.1.6.1 Task Structure

- **UserMain Task:** Handles serial port 0 and TCP port 2000
- **SerialTask2:** Handles serial port 1 and TCP port 2001 (lower priority)

12.37.1.1.6.2 Data Flow

TCP Client <-> Network Stack <-> SerialToEthernetFunction <-> Serial Port <-> Connected Device

12.37.1.1.7 Technical Details

12.37.1.1.7.1 Buffer Management

- Uses 80-byte buffers for data transfer
- Non-blocking I/O with `select()` for efficient multiplexing
- Immediate data forwarding (no buffering delays)

12.37.1.1.7.2 Error Handling

- Automatic socket cleanup on network errors
- Graceful handling of serial port errors
- Connection state monitoring and recovery

12.37.1.1.7.3 Performance Considerations

- SerialTask2 runs at lower priority than UserMain
- 2-second select timeout for responsive operation
- Efficient file descriptor monitoring

12.37.1.1.8 Building and Deployment

1. **Requirements:** NetBurner development environment and hardware
2. **Compilation:** Use standard NetBurner build tools
3. **Deployment:** Flash to NetBurner device
4. **Network Setup:** Ensure device has network connectivity

12.37.1.1.9 Troubleshooting

12.37.1.1.9.1 Common Issues

No Connection**: Verify network connectivity and TCP port accessibility
 Data Loss**: Check baud rate and serial settings match connected device
 Stale Connections**: Override timeout will eventually clear blocked connections
 Performance Issues**: Consider adjusting buffer sizes for high-throughput applications

12.37.1.1.9.2 Debug Output

The application provides console output showing:

- Listen socket creation
- Client connections and disconnections
- Override timeout events
- Socket file descriptor information

12.37.2 Serial To Ethernet

12.37.2.1 Serial to Ethernet Bridge Application

12.37.2.1.1 Overview

This NetBurner NNDK application creates a bidirectional bridge between an RS-232 serial device and an Ethernet network. The program enables real-time data transmission between serial and network clients, effectively allowing serial devices to communicate over TCP/IP networks.

Note

Excellent example of creating a user web page configuration, using the Configuration System (`config_object`), Configuration Tags, and configuration objects in an application.

12.37.2.1.2 Features

- **Bidirectional Data Flow:** Forwards data seamlessly between serial port and TCP connections
- **Web-based Configuration:** HTML interface for configuring settings without recompilation
- **Connection Management:** Handles multiple connection attempts with configurable timeout and override policies
- **Real-time Data Pumping:** Efficient `select()` based I/O multiplexing for simultaneous data transfer
- **Debug Support:** Configurable debug output to separate or shared serial ports
- **Persistent Settings:** Configuration stored in flash memory

12.37.2.1.3 Architecture

12.37.2.1.3.1 Core Components

- **main.cpp:** Application entry point, network setup, and connection handling
- **datapump.cpp:** Core data transfer engine using `select()` for I/O multiplexing
- **nvsettings.h:** Configuration class definitions and default values
- **datapump.h:** Data pump interface and message definitions
- **index.html:** Web-based configuration interface

12.37.2.1.3.2 Data Flow

```
[Serial Device] <----> [Serial Port] <----> [DataPump] <----> [TCP Socket] <----> [Network Client]
```

12.37.2.1.4 Configuration Options

12.37.2.1.4.1 Network Settings

- **Listen Port:** TCP port for incoming connections (default: 23 - Telnet)
- **Client Timeout:** Inactivity timeout in seconds (default: 60)
- **Client Override Timeout:** Time before allowing new connections to override existing ones (default: 20)

12.37.2.1.4.2 Serial Settings

- **Baud Rate:** Configurable via web interface (9600, 19200, 38400, 57600, 115200, 230400)
- **Data Format:** 8 data bits, 1 stop bit, no parity (fixed)
- **Flow Control:** None (configurable in code)

12.37.2.1.4.3 Connection Override Policies

- **Ignore New Connections:** Set override timeout to 0xFFFFFFFF
- **Replace After Timeout:** Set override timeout to desired seconds
- **Always Replace:** Set override timeout to 0

12.37.2.1.5 Build Configuration

12.37.2.1.5.1 Preprocessor Definitions

- `SETTINGS_KEY`: Flash memory configuration key (0x48666050)
- `DATA_SERIAL_PORT`: Serial port number for data (1)
- `DEBUG_SERIAL_PORT`: Serial port for debug output (0)
- `DEBUG`: Enable/disable debug output (1)

12.37.2.1.5.2 Buffer Sizes

- `BUFFER_SIZE`: Data transfer buffer size (1500 bytes)
- `CLIENT_WRITE_BUF_SIZE`: Client write buffer (256 bytes)

12.37.2.1.6 Usage

12.37.2.1.6.1 Initial Setup

1. Connect serial device to the configured data serial port
2. Connect NetBurner device to Ethernet network
3. Power on and wait for network initialization

12.37.2.1.6.2 Web Configuration

1. Navigate to the device's IP address in a web browser
2. Configure TCP and serial port settings as needed
3. Save settings (optionally reboot on save)

12.37.2.1.6.3 Client Connection

1. Connect TCP client to device IP on configured port
2. Data will flow bidirectionally between serial device and TCP client
3. Connection status messages are sent to both serial and network endpoints

12.37.2.1.7 Connection Management

12.37.2.1.7.1 Message Types

- **MTS (Message To Serial):** Status messages sent to serial port
- **MTN (Message To Network):** Status messages sent to network client

12.37.2.1.7.2 Status Messages

- Connection opened/closed notifications
- Timeout warnings
- Connection override notifications

12.37.2.1.8 Data Pump Algorithm

The DataPump function implements efficient I/O multiplexing:

1. **Setup:** Initialize file descriptor sets for read, write, and error monitoring
2. **Select:** Wait for I/O activity with configurable timeout
3. **Process:** Handle data transfer in both directions simultaneously
4. **Buffer Management:** Manage separate buffers for each direction
5. **Error Handling:** Detect and handle connection errors gracefully

12.37.2.1.9 Error Handling

- **Connection Errors:** Automatic cleanup and restart
- **Timeout Management:** Configurable inactivity timeouts
- **Buffer Overflow:** Fixed-size buffers with proper bounds checking
- **Network Errors:** Graceful socket closure and restart

12.37.2.1.10 Performance Characteristics

- **Throughput:** Limited by slowest link (typically serial port)
- **Latency:** Low latency due to `select()` based I/O
- **Memory Usage:** Fixed buffer allocation for predictable memory footprint
- **CPU Usage:** Event-driven architecture minimizes CPU overhead

12.37.2.1.11 Development Notes

12.37.2.1.11.1 Adding Features

- Modify `classSerialSettings` in `nvsettings.h` for new configuration options
- Update `index.html` for web interface changes
- Extend `DataPump` function for additional protocols

12.37.2.1.11.2 Debug Configuration

- Set `DEBUG_SERIAL_PORT` to -1 to share debug and data ports
- Set `DEBUG` to 0 to disable debug output entirely
- Use separate debug port for isolation during development

12.37.2.1.12 Hardware Requirements

- NetBurner embedded module with NNDK support
- RS-232 serial interface
- Ethernet connection

12.37.2.1.13 Software Dependencies

- NetBurner NNDK (NetBurner Network Development Kit)
- Config server system for web-based configuration
- Standard POSIX socket and file descriptor APIs

12.37.3 Serial HTTP Get Request

Example of sending a HTTP GET request through serial port when running a serial to Ethernet application.

Hardware Configuration:

Serial Device <--serial--> NetBurner Serial to Ethernet Device <--ethernet--> Internet

In this example we will be using a SB800EX as the "Serial Device", but this would typically be a microprocessor or FPGA with a UART that needs to communicate over a network. The source code in this file is an example of what you would need to implement in your own microprocessor to send a GET request through a NetBurner device to some other device on the network/Internet running a web server.

The code in the main.cpp source file was tested with the SB800EX, but should work with any of the NetBurner standard devices. A serial cable was connected from port 1 of the SB800EX to port 1 of the SBL2e development board. Since the SBL2e development board DB9 connector is cross-wired for RX and TX, all you need is a straight through serial cable.

SB800EX <--serial--> SBL2e <--ethernet--> Internet

The web page contacted is myip.dnssdynamic.com, which return the IP address of the HTTP GET request.

Software Configuration for SBL2e, serial port 1:

Running the SBL2e factory serial to Ethernet application. This was done through the web page, but you can also do it through serial AT commands.

1. Incoming TCP Settings: Uncheck Listen for Incoming Network Connections
2. Outgoing TCP Settings: Make outgoing connections: If serial data received
3. Outgoing TCP Settings: Connect on network port: 80
4. Outgoing TCP Settings: Connect to this address: myip.dnssdynamic.com

12.37.4 Serial Receive Callback

12.37.4.1 Serial Receive Callback Example

12.37.4.1.1 Overview

This NetBurner embedded application demonstrates how to register a custom callback function to intercept and handle serial port receive processing. The example shows how to bypass the default system serial receive handling and implement custom processing using a circular buffer.

12.37.4.1.2 Application Description

The application creates a serial receive callback system that:

- Opens serial port 1 at 115200 baud rate
- Registers a custom callback function to handle incoming serial data
- Uses a circular buffer to store received data
- Continuously processes and displays received characters

12.37.4.1.3 Key Features

12.37.4.1.3.1 Serial Receive Callback

- **Function:** `SerialReceiveCallback(int uartNum, unsigned char dataByte)`
- **Purpose:** Intercepts serial data at the interrupt level
- **Restrictions:** Called from interrupt context - no I/O functions allowed, only RTOS post functions

12.37.4.1.3.2 Circular Buffer Implementation

- **Size:** 256 bytes (RX_BUF_SIZE)
- **Variables:**
 - `indexGet`: Read pointer for the circular buffer
 - `indexPut`: Write pointer for the circular buffer
 - `CircBufferArray`: The actual buffer storage
- **Buffer Management:** Automatically wraps around when reaching buffer limits

12.37.4.1.3.3 Main Processing Loop

- Opens serial port 1 at 115200 baud
- Assigns the custom callback to UART1 receive processing
- Continuously reads from the circular buffer and displays characters
- Uses `OSTimeDly(1)` for task scheduling

12.37.4.1.4 Hardware Requirements

- NetBurner embedded device with UART1 capability
- Serial connection to UART1 at 115200 baud rate

12.37.4.1.5 Code Structure

```
main.cpp
  Headers and includes
  Global variables (circular buffer)
  SerialReceiveCallback() - Interrupt-level callback
  UserMain() - Main application loop
```

12.37.4.1.6 Important Notes

12.37.4.1.6.1 Interrupt Context Limitations

The `SerialReceiveCallback` function operates in interrupt context, which means:

- No standard I/O functions (`printf`, `iprintf`, etc.) can be called
- Only RTOS post functions are safe to use
- Keep processing minimal and fast

12.37.4.1.6.2 Buffer Overflow Handling

The circular buffer implements a simple overflow strategy:

- When the buffer is full, the oldest data is overwritten
- Both read and write pointers automatically wrap around

12.37.4.1.6.3 System Diagnostics

The application enables system diagnostics with `EnableSystemDiagnostics()`. This should be removed for production code.

12.37.4.1.7 Usage

1. Compile and flash the application to a NetBurner device
2. Connect a serial terminal to UART1 at 115200 baud
3. Send data to the serial port
4. Observe the echoed characters in the application output

12.37.4.1.8 Build Information

The application displays build timestamp information on startup, showing when the code was compiled.

12.37.5 Serial to Serial

12.37.5.1 Serial to Serial Example

Note

Refer to the OpenAllUARTS example for your particular platform to see all possible serial configurations and pin locations

12.37.5.1.1 Overview

This application demonstrates a simple serial port bridge that forwards data bidirectionally between two serial ports on a NetBurner embedded system. The program uses the `select()` function to efficiently monitor both serial ports simultaneously and forward data from one port to the other in real-time.

12.37.5.1.2 Description

The Serial to Serial example opens two serial ports and creates a bridge between them, allowing data sent to one port to be automatically forwarded to the other port. This is useful for applications such as:

- Serial port debugging and monitoring
- Creating a serial bridge between different devices
- Testing serial communication systems
- Bridging different serial interfaces

12.37.5.1.3 Key Features

- **Bidirectional Communication:** Data flows in both directions between the two serial ports
- **Interrupt-Driven Mode:** Uses buffered interrupt-driven serial communication for efficient operation
- **Non-blocking I/O:** Utilizes the `select()` function to monitor multiple file descriptors simultaneously
- **Configurable Timeout:** Uses a 5-second timeout for the select operation
- **LED Indicators:** On MOD5441X platforms, LEDs indicate when data is being transferred

12.37.5.1.4 Technical Implementation

12.37.5.1.4.1 Serial Port Configuration

Both serial ports are configured with identical settings:

- **Baud Rate:** 115,200 bps
- **Stop Bits:** 1
- **Data Bits:** 8
- **Parity:** None

12.37.5.1.4.2 Operation Flow

1. **Initialization:**

- System initialization and network startup
- Serial ports are closed (they start in polled mode from monitor)
- Serial ports are reopened in interrupt-driven mode

2. **Initial Test:**

- Sends "Test1" to serial port 1
- Sends "Test0" to serial port 0

3. **Main Loop:**

- Sets up file descriptor set for both serial ports
- Uses `select ()` to wait for data on either port (5-second timeout)
- When data arrives on port 1, forwards it to port 0
- When data arrives on port 0, forwards it to port 1
- On MOD5441X platforms, toggles LEDs to indicate activity

12.37.5.1.4.3 Buffer Management

- Uses 40-byte buffers for data transfer
- Reads available data and immediately forwards it to the opposite port
- No data modification or processing - pure pass-through operation

12.37.5.1.5 Hardware Requirements

- NetBurner embedded system with at least two serial ports
- Compatible with MOD5441X platforms (includes LED indicators)
- Serial devices or cables connected to both serial ports for testing

12.37.5.1.6 Usage

1. Connect serial devices or use serial cables to connect to both serial ports
2. Compile and deploy the application to your NetBurner device
3. Power on the device - it will automatically start bridging the serial ports
4. Send data to either serial port and observe it being forwarded to the other port
5. On MOD5441X platforms, watch for LED activity indicating data transfer

12.37.5.1.7 Code Structure

- **Initialization:** Sets up the system and network
- **Serial Port Setup:** Configures both serial ports for interrupt-driven operation
- **Main Loop:** Implements the select-based data forwarding logic
- **Platform-Specific Features:** Conditional LED control for supported platforms

12.37.5.1.8 Notes

- The system diagnostics are enabled in this example but should be removed for production code
- The application waits up to 5 seconds for network activation at startup
- The select timeout of 5 seconds means the loop will cycle even if no data is received
- This is a blocking application that runs indefinitely

12.37.6 Serial Application Update

12.37.6.1 Overview

Updating a device application can be done many ways. The most common is over a network connection, but there are times when it is preferable or necessary to update an applicatoin over a serial connection.

The Serial Update Example demonstrates two approaches: #- Serial config server. This can be though of as being the same as manually entering the serial config server with an A to abort command and using the fla command to flash the application. #- Direct. This method assumed 100% control over the update and does not use the serial configuration server.

12.37.6.2 Methods

12.37.6.2.1 Method 1:

The `void ResumeSerialConfig(int fd_cfg, bool persist)` system function is used to trigger the serial configuraiton server to use the specified file descriptor to process the serial stream. Nothing more is required. This is the most common method.

12.37.6.2.2 Method 2:

The direct method puts the update responsibility on the application. The example demonstrates two scenarios: The serial port is being used for both applications data as well as a serial application update. In this case a mechanism is required to identify an update vs serial data. To accomodate this situation, the example provides an approach to using a configurable preamble and escape mechanisms. It is up to the application to decide what the best mechanisam is to identify an update vs data; the example provides one such method. A preamble and/or escape mechanism may or may not be necessary in your specific use case.

12.37.7 TCP to Serial

12.37.7.1 TCP to Serial Bridge

A NetBurner embedded application that creates a bidirectional bridge between TCP network connections and serial communication ports.

12.37.7.1.1 Overview

This application enables remote access to serial devices over a network by creating a TCP server that forwards data between network clients and a serial port. It's commonly used for remote device management, debugging, and data acquisition from serial-based equipment.

Note

You can test with on a single PC by using a telnet application and serial terminal like MTTY

12.37.7.1.2 Features

- **Bidirectional Communication:** Data flows seamlessly between TCP clients and serial devices
- **Telnet Compatible:** Listens on port 23 (standard Telnet port) for easy connection
- **Connection Management:** Handles multiple connection attempts with configurable override behavior
- **Timeout Protection:** Automatically closes idle connections after 60 seconds

- **Multi-Platform Support:** Supports various NetBurner platforms (SOMRT1061, MODRT1171, SB800EX)
- **Status Monitoring:** Provides connection status messages to both serial and network interfaces

12.37.7.1.3 Configuration

12.37.7.1.3.1 Serial Port Settings

- **Baud Rate:** 115,200 bps
- **Data Bits:** 8
- **Stop Bits:** 1
- **Parity:** None

12.37.7.1.3.2 Network Settings

- **TCP Listen Port:** 23 (Telnet)
- **Connection Timeout:** 60 seconds of inactivity
- **Override Timeout:** 20 seconds (configurable)

12.37.7.1.3.3 Connection Override Behavior

The application supports three modes for handling new connections when one is already active:

1. **Ignore New Connections** (`OVERIDE_TIMEOUT = 0xFFFFFFFF`)
 - Maintains current connection, rejects new attempts
2. **Timed Override** (`OVERIDE_TIMEOUT = 20 seconds`)
 - Allows new connection to replace existing one after 20 seconds of current connection activity
3. **Always Override** (`OVERIDE_TIMEOUT = 0`)
 - Immediately replaces existing connection with new one

12.37.7.1.4 Platform Support

The application automatically configures pins for different NetBurner platforms where necessary:

- **SOMRT1061:** Uses LPUART4 on pins 87 (RX), 88 (TX), with flow control on pins 2 (RTS), 3 (CTS)
- **MODRT1171:** Uses LPUART2 on pins 21 (RX), 22 (TX)
- **SB800EX:** Uses UART2

Notes for SOMRT1061

- The console port for status messages is J13 on the development board, which is routed to LPUART3 on the processor.
- The data port is J14 on the development board, which is routed to LPUART4 on the processor.
- RX, TX, RTS, CTS pin selection is always required before opening the serial port.

Note

Refer to the OpenAllUARTS example for your particular platform to see all possible pin configurations.

12.37.7.1.5 Usage

1. **Compile and Deploy:** Build the application for your NetBurner platform and deploy to the device
2. **Connect Serial Device:** Attach your serial device to the configured UART pins
3. **Network Connection:** Connect via Telnet client to the device's IP address on port 23
`telnet <device-ip-address> 23`
4. **Data Exchange:** Data typed in the Telnet session is sent to the serial device, and serial responses appear in the Telnet session

12.37.7.1.6 Status Messages

The application provides informative status messages:

To Serial Port:**

- "Waiting for connection"
- "New Connection Opened"
- "Connection Closed by Network"
- "Connection Timed out and Closed"
- "This Connection is being Overridden"
To Network Client:**
- "Connection Opened"
- "Your Connection was just Overridden"
- "Your Connection Timed out and will be Closed"

12.37.7.1.7 Technical Details

- Uses NetBurner's RTOS with `select()` for efficient I/O multiplexing
- Implements proper socket management with error handling
- Supports hardware flow control (RTS/CTS) where available
- Buffer size: 40 bytes for data transfers
- Non-blocking I/O operations for responsive performance

12.37.7.1.8 Use Cases

- Remote serial device configuration and monitoring
- Data logging from serial sensors over network
- Remote debugging of embedded systems
- Industrial automation equipment access
- Legacy serial device integration into modern networks

12.37.7.1.9 Requirements

- NetBurner development kit
- NetBurner development environment
- Network connectivity (Ethernet)
- Serial device or equipment to bridge

12.37.7.1.10 Notes

- Ensure proper pin configuration for your specific NetBurner platform
- Test connection override behavior matches your application requirements

12.37.8 Serial Webserver

12.37.8.1 Serial Webserver

A NetBurner RTOS application that bridges serial communication with HTTP web services, allowing HTTP requests to be received and processed over a serial port connection.

12.37.8.1.1 Overview

The Serial Webserver demonstrates how to create a bidirectional communication bridge between a serial port and TCP/HTTP connections. This application enables HTTP requests to be sent over a serial connection and receive corresponding web responses back through the same serial interface.

It will receive and reply to HTTP requests over a serial port, using the NetBurner web server implementation. The example will display a Hello World message on the device's index page `index.html`.

12.37.8.1.2 Key Features

- **Serial-to-HTTP Bridge:** Receives HTTP requests via serial port and forwards them to the built-in web server
- **Bidirectional Communication:** Splices data between serial and TCP connections in both directions
- **Automatic Connection Management:** Handles TCP connection creation and cleanup automatically
- **Error Recovery:** Automatically reopens serial connections on errors
- **Built-in Web Server:** Serves static content including HTML, CSS, and images

12.37.8.1.3 Technical Specifications

- **Serial Port:** Port 1 at 115200 baud
- **HTTP Port:** Standard port 80
- **Buffer Size:** Uses Ethernet buffer size for data transfer
- **Network Stack:** NetBurner RTOS with TCP/IP support

12.37.8.1.4 Application Architecture

The application follows a simple event-driven architecture:

1. **Initialization:** Sets up network stack, enables diagnostics, and starts HTTP server
2. **Serial Monitoring:** Waits for incoming serial data
3. **Connection Creation:** Establishes TCP connection to local web server when data arrives
4. **Data Splicing:** Uses `select()` to monitor both serial and TCP file descriptors
5. **Bidirectional Transfer:** Forwards data between serial and TCP connections
6. **Error Handling:** Closes and reopens connections on errors

12.37.8.1.5 HTTP Request Format

The application expects standard HTTP requests over the serial connection. A typical request format:

```
GET /INDEX.HTML HTTP/1.1\r\n\r\n
```

Important**: The web server requires a double carriage return line feed (`\r\n\r\n`) to properly parse HTTP requests.

12.37.8.1.6 Expected Response

A successful request to the index page returns:

```
HTTP/1.0 200 OK
Pragma: no-cache
MIME-version: 1.0
Content-Type: text/html
Content-length: 231
```

```
<html>
<head><link rel="stylesheet" type="text/css" href="style.css"></head>
<body>
<a href="http://www.netburner.com"></a>
<h1>Thank you for NetBurning!</h1>
</body>
</html>
```

12.37.8.1.7 Web Content

The application serves the following static files:

- **index.html**: Main page with NetBurner branding and welcome message
- **style.css**: Stylesheet defining Arial font family for the page
- **logo.jpg**: NetBurner logo image (referenced but not included in source)

12.37.8.1.8 Usage

1. Flash the application to your NetBurner device
2. Connect to the device's serial port (Port 1, 115200 baud)
3. Send HTTP requests through the serial connection
4. Receive HTTP responses back through the same serial connection

12.37.8.1.9 Development Notes

- System diagnostics are enabled for development (consider removing for production)
- The application uses blocking I/O operations
- Memory allocation uses `FAST_USER_VAR` for the splice buffer

12.37.8.1.10 Use Cases

This application is ideal for:

- Testing HTTP communication through serial interfaces
- Bridging legacy serial devices to modern web services
- Remote web server access through serial connections

12.37.8.1.11 Build Requirements

- NetBurner NBRTOS development environment
- Standard NetBurner libraries (init, system, iosys, serial, tcp, netinterface)
- Compatible NetBurner hardware platform

12.38 SHA1 Digest

Calculate the SHA1 128-bit digest on an application. To display the digest when building the application in NBEclipse, go to the project properties and add `-D` to the compcode flags.

- **Formatted display** - Clean HTML tables with organized information
- **Multiple protocol links** - Both IPv4 and IPv6 connectivity testing

12.39.1.3.3 Development and Debugging

- **Serial console output** - Detailed information for development
- **DHCP lease tracking** - Timing information for address renewals
- **Link status reporting** - Physical connection monitoring
- **Optional features** - WiFi and static IPv6 configuration support

12.39.1.4 Implementation Details

12.39.1.4.1 Main Application (main.cpp)

12.39.1.4.1.1 System Initialization and Main Loop

```
void UserMain(void *pd) {
    init(); // Initialize network stack
    EnableSystemDiagnostics(); // Enable debug output
    StartHttp(); // Start HTTP server on port 80
    WaitForActiveNetwork(TICKS_PER_SECOND * 5); // Wait for network connectivity

    printf("Application: %s\nNNDK Revision: %s\n", AppName, GetReleaseTag());

#ifdef USE_WIFI
    InitWifi_SPI(); // Initialize WiFi if enabled
#endif

#ifdef STATIC_IP6_TEST
    // Add static IPv6 address for testing
    IPv6Interface *pIPv6If = IPv6Interface::GetFirst_IP6_Interface();
    IPADDR i6 = IPADDR6::AsciiToIp6("1234:5678:abcd::1234");
    pIPv6If->AddStaticAddress(i6, 64);
#endif

    // Interactive main loop
    while (1) {
        printf("\nType any key to display address information\n");
        printf("Note that IPv6 routable addresses may take a few seconds to be received\n");
        getchar();
        showIpAddressesSerial();
        showLinkStatus();
        printf("\n\n");
    }
}
```

12.39.1.4.1.2 Configuration Options

```
// Optional WiFi support
#define USE_WIFI (1)

// Optional static IPv6 testing
#define STATIC_IP6_TEST (1)

// IPv6 support (typically defined globally)
#ifdef IPV6
```

12.39.1.4.2 Network Information Utilities (ip_util.cpp)

12.39.1.4.2.1 Serial Console Display Function

```
void showIpAddressesSerial() {
    int ifNumber = GetFirstInterface();

    while (ifNumber) {
        InterfaceBlock *ifBlock = GetInterfaceBlock(ifNumber);

        printf("\nInterface %d, %s, MAC:", ifNumber, ifBlock->GetInterfaceName());
        InterfaceMAC(ifNumber).print();
        printf("\n\n");

        // Display IPv4 information
        printf("IPv4 Addresses:\n");
        printf("IP: %hI\n", InterfaceIP(ifNumber));
        printf("Mask: %hI\n", InterfaceMASK(ifNumber));
    }
}
```

```

printf("DNS:      %hI\n", InterfaceDNS(ifNumber));
printf("DNS2:     %hI\n", InterfaceDNS2(ifNumber));
printf("Gateway:  %hI\n", InterfaceGate(ifNumber));
printf("Auto IP:   %hI\n", InterfaceAutoIP(ifNumber));

#ifdef IPV6
// Display IPv6 information
printf("IPv6 Addresses:\n");
IPv6Interface *pIPv6If = IPv6Interface::GetInterfaceN(ifNumber);
if (pIPv6If) {
    IPV6_PREFIX *pPrefix = pIPv6If->FirstPrefix();
    while (pPrefix) {
        if (pPrefix->bValidForInterface && pPrefix->AgeStillValidTest()) {
            ePrefixSource pSrc = pPrefix->Source();
            printf("%-10s: ", SourceName(pSrc));
            pPrefix->m_IPAddress.print();
            printf("/%d\n", pPrefix->PrefixLen);

            // Display DHCP lease information if applicable
            if (pPrefix->pDHCPD) {
                char buf[120];
                getLeaseTimeInfo(pPrefix, buf, 120);
                printf("      %s\n", buf);
            }
        }
        pPrefix = pPrefix->GetNext();
    }
}
#endif
ifNumber = GetNextInterface(ifNumber);
}
}

```

12.39.1.4.2.2 Web Interface Display Function

```

void showIpAddressesWeb(int socket, const char *url) {
    IPADDR localIP = GetSocketLocalAddr(socket);
    uint32_t remotePort = GetSocketRemotePort(socket);

    fdprintf(socket, "<br>Web browser request came from:");
    localIP.fprintf(socket);
    fdprintf(socket, ": %d<br>\n", remotePort);

    int ifNumber = GetFirstInterface();
    while (ifNumber) {
        InterfaceBlock *ifBlock = GetInterfaceBlock(ifNumber);

        fdprintf(socket, "<hr>\n");
        fdprintf(socket, "<strong>Interface # %d, %s, MAC: ", ifNumber, ifBlock->GetInterfaceName());
        MACADDR MacAddr = InterfaceMAC(ifNumber);
        fdShowMac(socket, &MacAddr);
        fdprintf(socket, "</strong><br>");

        // Create IPv4 information table
        fdprintf(socket, "<br><strong>IPv4 Addresses: </strong><br>");
        fdprintf(socket, "<table border=0>\n");
        fdprintf(socket, "<tr><td>IP:      </td><td>%hI</td></tr>\n", InterfaceIP(ifNumber));
        fdprintf(socket, "<tr><td>Mask:    </td><td>%hI</td></tr>\n", InterfaceMASK(ifNumber));
        fdprintf(socket, "<tr><td>DNS1:   </td><td>%hI</td></tr>\n", InterfaceDNS(ifNumber));
        fdprintf(socket, "<tr><td>DNS2:   </td><td>%hI</td></tr>\n", InterfaceDNS2(ifNumber));
        fdprintf(socket, "<tr><td>Gateway: </td><td>%hI</td></tr>\n", InterfaceGate(ifNumber));
        fdprintf(socket, "<tr><td>AutoIP: </td><td>%hI</td></tr>\n", InterfaceAutoIP(ifNumber));
        fdprintf(socket, "</table>\n");

        // Create clickable IPv4 link
        if (GetInterfaceLink(ifNumber)) {
            fdprintf(socket, "Make an IPv4 request: <a href=\"http://");
            InterfaceIP(ifNumber).fprintf(socket);
            fdprintf(socket, "\">");
            InterfaceIP(ifNumber).fprintf(socket);
            fdprintf(socket, "</a><br>\n");
        }
    }

#ifdef IPV6
// Create IPv6 information table with clickable links
IPv6Interface *pIPv6If = IPv6Interface::GetInterfaceN(ifNumber);
if (pIPv6If) {
    fdprintf(socket, "<br><strong>IPv6 Addresses:</strong><br>\n");
    fdprintf(socket, "<table border=0 cellpadding=5>\n");

    IPV6_PREFIX *pPrefix = pIPv6If->FirstPrefix();
    while (pPrefix) {
        if (pPrefix->bValidForInterface) {
            ePrefixSource pSrc = pPrefix->Source();
            fdprintf(socket, "<tr> <td><a href=\"http://[%I]\">%I</a></td> <td>Assigned by %s",

```

```

        pPrefix->m_IPAddress, pPrefix->m_IPAddress, SourceName(pSrc));

        if (pPrefix->pDHCPCD) {
            char buf[120];
            getLeaseTimeInfo(pPrefix, buf, 120);
            fdprintf(socket, ": %s", buf);
        }
        fdprintf(socket, "</td></tr>\n");
    }
    pPrefix = pPrefix->GetNext();
}
fdprintf(socket, "</table>\n");
}
#endif
    ifNumber = GetNextInterface(ifNumber);
}
}

```

12.39.1.4.2.3 Link Status Monitoring

```

void showLinkStatus() {
    int ifNumber = GetFirstInterface();

    while (ifNumber) {
        InterfaceBlock *ifBlock = GetInterfaceBlock(ifNumber);

        if (ifBlock) {
            printf("Interface %d, %s:\n", ifNumber, ifBlock->GetInterfaceName());
            if (ifBlock->LinkActive()) {
                printf("Link Status: UP, %dMB, ", ifBlock->LinkSpeed());
                if (ifBlock->LinkDuplex()) {
                    printf("Full Duplex\n");
                } else {
                    printf("Half Duplex\n");
                }
            } else {
                printf("Link Status: DOWN\n");
            }
        }
        ifNumber = GetNextInterface(ifNumber);
    }
}

```

12.39.1.4.3 IPv6 Address Source Management

12.39.1.4.3.1 Address Source Identification

```

const char *SourceName(ePrefixSource e) {
    switch (e) {
        case eLinkLocal: return "Link Local";
        case eRouter: return "Router";
        case eDHCP: return "DHCP";
        case eStatic: return "Static";
        case eUnknown:
        default: return "Unknown";
    }
}

```

12.39.1.4.3.2 DHCP Lease Time Tracking

```

void getLeaseTimeInfo(IPV6_PREFIX *pPrefix, char *timeStr, uint32_t maxTimeStrLen) {
    const uint32_t maxLen = 80;
    char strCurrentTime[maxLen];
    char strRenewTime[maxLen];
    char strRebindTime[maxLen];

    uint32_t seconds = TimeTick / TICKS_PER_SECOND;
    getTimeStringFromSeconds(seconds, strCurrentTime, maxLen);

    seconds = (pPrefix->pDHCPCD->m_renewTick) / TICKS_PER_SECOND;
    getTimeStringFromSeconds(seconds, strRenewTime, maxLen);

    seconds = (pPrefix->pDHCPCD->m_rebindTick) / TICKS_PER_SECOND;
    getTimeStringFromSeconds(seconds, strRebindTime, maxLen);

    snprintf(timeStr, maxTimeStrLen,
        "Time Since Boot: %s, Renew: %s, Rebind: %s",
        strCurrentTime, strRenewTime, strRebindTime);
}

```

12.39.1.4.3.3 Time Formatting Utility

```
void getTimeStringFromSeconds(uint32_t seconds, char *strTime, uint32_t maxLen) {
    uint32_t hour = seconds / 3600;
    uint32_t min = (seconds % 3600) / 60;
    uint32_t sec = seconds % 60;

    snprintf(strTime, maxLen, "%02u:%02u:%02u", hour, min, sec);
}
```

12.39.1.5 Web Interface Structure

12.39.1.5.1 Main Page Layout (index.html)

```
<html>
<head>
<title>NetBurner IPv4/IPv6 Show Addresses Example</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#FFFFFF" text="#000000">
<table width="850" border="0" cellpadding="0" cellspacing="0" height="121">
  <tr>
    <td width="200"><a href="http://www.netburner.com"></a></td>
    <td width="50"></td>
    <td width="600" align="right">
      <b><font face="Arial, Helvetica, sans-serif" size="5" color="#0000FF">
        NetBurner IPv4/IPv6 Show Addresses Example</font></b>
    </td>
  </tr>
</table>

<font face="Arial, Helvetica, sans-serif" size="3">
<table width="850" border="0" cellpadding="0" cellspacing="0">
  <tr>
    <td>This example program demonstrates how to access the NetBurner web server
      using either an IPv4 or IPv6 IP address.</td>
  </tr>
</table>

<!--CPPCALL showIpAddressesWeb -->
</font>
</body>
</html>
```

12.39.1.5.2 Dynamic Content Integration

- **CPPCALL tag** - `<!--CPPCALL showIpAddressesWeb -->` dynamically generates network information
- **Interactive links** - IP addresses become clickable links for connectivity testing
- **Real-time data** - Information reflects current network state when page loads

12.39.1.6 Network Information Display

12.39.1.6.1 IPv4 Address Information

- **IP Address** - Current assigned IPv4 address
- **Subnet Mask** - Network mask for local subnet
- **DNS Servers** - Primary and secondary DNS server addresses
- **Default Gateway** - Router address for external connectivity
- **AutoIP Address** - Link-local address (169.254.x.x) if **DHCP** unavailable

12.39.1.6.2 IPv6 Address Information

- **Link Local** - Automatically generated local addresses (fe80::/64)
- **Router Assigned** - Addresses from Router Advertisement messages
- **DHCP Assigned** - Addresses from DHCPv6 server

- **Static Assigned** - Manually configured addresses
- **DNS Servers** - IPv6 DNS server information with source attribution

12.39.1.6.3 Physical Interface Details

- **Interface Number** - System interface identifier
- **Interface Name** - Human-readable interface name (e.g., "Ethernet 0")
- **MAC Address** - Hardware address in standard format
- **Link Status** - UP/DOWN physical connection state
- **Connection Speed** - 10/100/1000 Mbps negotiated speed
- **Duplex Mode** - Half/Full duplex communication mode

12.39.1.6.4 DHCP Lease Management

- **Current Time** - System uptime in HH:MM:SS format
- **Renew Time** - When **DHCP** lease renewal will occur
- **Rebind Time** - When **DHCP** lease rebinding will occur
- **Lease Source** - Whether address came from **DHCP** or Router Advertisement

12.39.1.7 Usage Instructions

12.39.1.7.1 Build and Deployment

1. **Build application:**
2. **Deploy to device:**
3. **Monitor startup:**
 - Connect serial terminal (115200 baud, 8-N-1)
 - Observe network initialization and IP address assignment

12.39.1.7.2 Serial Console Interface

12.39.1.7.2.1 Interactive Operation

1. **Wait for prompt:** "Type any key to display address information"
2. **Press any key** to trigger information display
3. **View output:** Complete network interface information
4. **Repeat as needed** to monitor network changes

12.39.1.7.2.2 Expected Console Output

Application: Show Interfaces
NNDK Revision: 3.x.x Build xxxx

Type any key to display address information
Note that IPv6 routable addresses may take a few seconds to be received

Interface 1, Ethernet 0, MAC: 00:03:F4:XX:XX:XX

IPv4 Addresses:
IP: 192.168.1.100
Mask: 255.255.255.0
DNS: 192.168.1.1
DNS2: 0.0.0.0
Gateway: 192.168.1.1

```

Auto IP: 0.0.0.0

IPv6 Addresses:
Link Local: fe80::203:f4ff:fexx:xxxx/64
Router      : 2001:db8::203:f4ff:fexx:xxxx/64
              Time Since Boot: 00:05:23, Renew: 01:00:00, Rebind: 01:45:00

DNS:
  2001:db8::1 - Source: Router

Interface 1, Ethernet 0:
Link Status: UP, 100MB, Full Duplex

```

12.39.1.7.3 Web Interface Operation

12.39.1.7.3.1 Access and Navigation

1. **Open browser** to device IP address (e.g., <http://192.168.1.100>)
2. **View network information** in formatted HTML tables
3. **Test connectivity** by clicking any displayed IP address
4. **IPv6 testing** - Click IPv6 addresses to test IPv6 connectivity

12.39.1.7.3.2 Interactive Features

- **IPv4 Links** - Format: <http://192.168.1.100>
- **IPv6 Links** - Format: [http://\[2001:db8::203:f4ff:fexx:xxxx\]](http://[2001:db8::203:f4ff:fexx:xxxx])
- **Connectivity Testing** - Each link opens new page to test reachability
- **Real-time Information** - Refresh page to see current network status

12.39.1.7.3.3 Expected Web Display

Web browser request came from: 192.168.1.50: 54321

Device Interfaces:

Interface # 1, Ethernet 0, MAC: 00:03:F4:XX:XX:XX

```

IPv4 Addresses:
IP:      192.168.1.100
Mask:    255.255.255.0
DNS1:    192.168.1.1
DNS2:    0.0.0.0
Gateway: 192.168.1.1
AutoIP:  0.0.0.0

```

Make an IPv4 request: [192.168.1.100] (clickable link)

```

IPv6 Addresses:
[fe80::203:f4ff:fexx:xxxx] (clickable)   Assigned by Link Local
[2001:db8::203:f4ff:fexx:xxxx] (clickable) Assigned by Router: Time Since Boot: 00:05:23, Renew: 01:00:00,
Rebind: 01:45:00

```

12.39.1.8 Advanced Features

12.39.1.8.1 Multiple Interface Support

The application automatically detects and displays all available network interfaces:

```

int ifNumber = GetFirstInterface();
while (ifNumber) {
    // Process each interface
    InterfaceBlock *ifBlock = GetInterfaceBlock(ifNumber);
    // ... display interface information
    ifNumber = GetNextInterface(ifNumber);
}

```

12.39.1.8.2 WiFi Integration

Optional WiFi support can be enabled:

```

#ifdef USE_WIFI
    InitWifi_SPI(); // Initialize WiFi interface
#endif

```


12.39.1.8.3 IPv6 Address Assignment Monitoring

The application tracks IPv6 addresses from multiple sources:

- **Link Local** - Automatic based on MAC address
- **Router Advertisement** - Assigned by network router
- **DHCPv6** - Assigned by [DHCP](#) server
- **Static Configuration** - Manually configured addresses

12.39.1.8.4 DHCP Lease Tracking

For DHCP-assigned addresses, the system tracks:

- **Current system uptime** - Time since device boot
- **Lease renewal time** - When address renewal will occur
- **Lease rebind time** - When address rebinding will happen

12.39.1.8.5 Development and Testing Features

12.39.1.8.5.1 Static IPv6 Testing

```
#ifdef STATIC_IP6_TEST
    IPv6Interface *pIPv6If = IPv6Interface::GetFirst_IP6_Interface();
    IPADDR i6 = IPADDR6::AsciiToIp6("1234:5678:abcd::1234");
    pIPv6If->AddStaticAddress(i6, 64);
#endif
```

12.39.1.8.5.2 Network Timing Considerations

- **IPv6 Address Assignment** - May take 5-30 seconds depending on router
- **DHCP Lease Updates** - Renewals happen automatically in background
- **Link Status Changes** - Detected immediately on physical connection changes

12.39.1.9 Troubleshooting

12.39.1.9.1 Common Issues

12.39.1.9.1.1 No IPv6 Addresses Displayed

Symptoms**: Only IPv4 information shown Solutions**:

- Wait 30 seconds for router advertisement
- Check router IPv6 configuration
- Verify IPv6 enabled in build configuration
- Test with known IPv6-enabled network

12.39.1.9.1.2 Link Status Shows DOWN

Symptoms**: "Link Status: DOWN" in output Solutions**:

- Check Ethernet cable connection
- Verify cable is good (test with PC)
- Check switch/router port status
- Test with different network port

12.39.1.9.1.3 DHCP Address Not Assigned

Symptoms*: IP shows 0.0.0.0 or AutoIP address Solutions*:

- Verify [DHCP](#) server is running on network
- Check [DHCP](#) server has available addresses
- Test with static IP configuration
- Monitor [DHCP](#) server logs for requests

12.39.1.9.1.4 Web Interface Not Accessible

Symptoms*: Browser cannot connect to device Solutions*:

- Verify IP address from serial console
- Check network connectivity with ping
- Confirm HTTP server started successfully
- Test with different browser or incognito mode

12.39.1.9.2 Debug Procedures

12.39.1.9.2.1 Network Connectivity Testing

```
# Test IPv4 connectivity
ping 192.168.1.100

# Test IPv6 connectivity (if supported)
ping6 2001:db8::203:f4ff:fexx:xxxx

# Test HTTP connectivity
curl -v http://192.168.1.100/
curl -v http://[2001:db8::203:f4ff:fexx:xxxx]/
```

12.39.1.9.2.2 Serial Console Monitoring

```
// Add debug output to track network changes
printf("Debug: Interface %d status changed\n", ifNumber);
printf("Debug: Link speed: %d, Duplex: %s\n",
    ifBlock->LinkSpeed(),
    ifBlock->LinkDuplex() ? "Full" : "Half");
```

12.39.1.9.2.3 Address Assignment Verification

```
// Check specific address types
if (InterfaceIP(ifNumber) == IPADDR::NullIP()) {
    printf("Warning: No IPv4 address assigned\n");
}

#ifdef IPV6
if (!pIPv6If || !pIPv6If->FirstPrefix()) {
    printf("Warning: No IPv6 addresses assigned\n");
}
#endif
```

12.39.1.10 Performance Considerations

12.39.1.10.1 Memory Usage

- **Static allocation** - Fixed memory for interface tracking
- **Dynamic enumeration** - Scales with number of interfaces
- **String formatting** - Temporary buffers for display functions
- **Web output** - Streaming output to minimize memory usage

12.39.1.10.2 Network Efficiency

- **On-demand display** - Information gathered only when requested
- **Minimal network overhead** - No background polling or monitoring
- **Efficient iteration** - Single pass through interface list
- **Cached interface data** - Uses system-maintained interface information

12.39.1.10.3 Real-time Considerations

- **Non-blocking display** - Information gathering doesn't block network stack
- **Quick response** - Fast enumeration of current network state
- **Background updates** - Network stack handles address changes automatically
- **Event-driven** - User-initiated display prevents unnecessary processing

12.39.1.11 Build Configuration

12.39.1.11.1 Makefile Components

```
NAME = ShowInterfaces

# Source files
CPP_SRC += src/main.cpp src/ip_util.cpp

# HTML compilation
CREATEDTARGS += src/htmldata.cpp
CPP_SRC += src/htmldata.cpp
src/htmldata.cpp : $(wildcard html/*.*)
    comphtml html -osrc/htmldata.cpp

include $(NNDK_ROOT)/make/boilerplate.mk
```

12.39.1.11.2 Compile-Time Options

- **IPv6 Support** - Enabled via global IPv6 preprocessor definition
- **WiFi Support** - Optional, enabled via `USE_WIFI` definition
- **Static IPv6 Testing** - Optional, enabled via `STATIC_IP6_TEST` definition

12.39.1.11.3 Dependencies

- **NetBurner NNDK** - Network Development Kit with IPv4/IPv6 support
- **Network Interface Libraries** - Built-in interface management
- **IPv6 Libraries** - For IPv6 address and prefix management (optional)
- **WiFi Libraries** - For wireless interface support (optional)

12.39.1.12 Learning Objectives

This example teaches:

- **Network interface enumeration** - How to discover and iterate through interfaces
- **IPv4/IPv6 address management** - Understanding dual-stack networking
- **DHCP lease management** - Tracking address assignment and renewal
- **Link status monitoring** - Physical layer connection management
- **Web-based network tools** - Creating interactive network diagnostic interfaces
- **Cross-platform networking** - Handling multiple interface types and protocols

12.39.1.13 Related Examples

For additional network programming learning, see:

- Basic networking examples for fundamental TCP/IP concepts
- [DHCP](#) client examples for dynamic address configuration
- IPv6 examples for advanced IPv6 features and protocols
- Web server examples for HTTP interface development
- WiFi examples for wireless network integration

12.40 SOCKS5 Client

This program demonstrates how to create a client connection using the SOCKS5 protocol. This protocol is detailed in RFC 1928. The NetBurner libraries support no authorization and plain text username/password options, and provide a callback for the user to implement their own version of GSSAPI, [AuthWithGssApi\(\)](#). This function can be overridden to implement GSSAPI according to specific requirements.

To test this, you can use a Docker image that will run a SOCK5 server, found here: <https://hub.docker.com/r/serjs/go-socks5-proxy/>

12.41 SPI

These SPI examples are common to all 3.x platforms.

Note

These SPI examples work on multiple processor/product architectures. There are additional SPI examples in the processor/product specific folders that may be of interest. For example, the MODM7AE70 has a quad SPI peripheral, but the MOD5441x does not.

- [DSPI Multiplex](#)
- [Serial to SPI](#)
- [SPI PicKit Demo Board](#)

12.41.1 DSPI Multiplex

Supported Platforms: MOD5441X NANO54415 MODM7AE70 SBE70LC SB800EX

This example demonstrates how to use the [DSPIModule](#), object based DSPI driver, to use multiple bus configurations on the same DSPI hardware module without having to re-initialize the driver every time when switching configurations.

12.41.2 Serial to SPI

Supported Platforms: MOD5441X NANO54415 MODM7AE70 SBE70LC

The Serial to SPI example demonstrates the basic configuration of the SPI driver and sending and receiving data over an SPI interface.

12.41.3 SPI PicKit Demo Board

Microchip PicKit Serial SPI Demo Board Driver

Supported Platforms: MODM7AE70, SBE70LC, SOMRT1061

Demonstrates the use of 7 Microchip SPI Peripherals:

- 25LC020A 2K SPI Bus Serial EEPROM
- TC77-5.0 Thermal Sensor with SPI Interface

- MCP3201 2.7V 12-Bit A/D Converter with SPI Serial Interface
- MCP4822 12-Bit DAC with Internal VREF and SPI Interface
- MCP41010 Single/Dual Digital Potentiometer with SPI Interface
- MCP6S92 Single-Ended, Rail-to-Rail I/O, Low-Gain PGA
- MCP23S08 8-Bit I/O Expander with Serial Interface

12.42 SSH

Secure Shell (SSH) Examples:

- [SSH Server](#)
- [SSH Server with User Key](#)
- [SSH Client](#)
- [SSH Server with User Authorization](#)
- [SecureSerToEthFactoryApp](#)

12.42.1 SSH Server

12.42.1.1 SSH Minimal Server Example

12.42.1.1.1 Overview

This NetBurner application implements a minimal SSH server that listens for incoming SSH client connections on TCP port 22. The server demonstrates basic SSH functionality using the NetBurner's built-in SSH library and certificate management system.

12.42.1.1.2 Features

- SSH server listening on port 22
- Automatic certificate and key generation
- Bidirectional communication between SSH client and server
- Serial console integration for monitoring and sending data
- Connection timeout handling (120 seconds)
- TCP error state reporting
- Multi-client support with connection queuing

12.42.1.1.3 Architecture

The application consists of two main components:

12.42.1.1.3.1 SSH Server Task

- Runs as a separate high-priority task
- Listens for incoming SSH connections
- Handles client authentication and session management
- Processes incoming data from SSH clients
- Manages connection lifecycle and cleanup

12.42.1.1.3.2 Main User Interface

- Provides serial console interface for monitoring
- Allows sending data to connected SSH clients via keyboard input
- Displays connection status and diagnostic information

12.42.1.1.4 Usage Instructions

12.42.1.1.4.1 Setup

1. Compile and load the application into your NetBurner device
2. Ensure the device is connected to your network
3. Run MTTTY or similar terminal emulator to connect to the device's USB/Serial port

12.42.1.1.4.2 Connecting SSH Clients

1. Use any SSH client (PuTTY, OpenSSH, etc.)
2. Connect to the NetBurner device's IP address on port 22
3. The server will display a welcome message: "Welcome to the NetBurner SSH Server"
4. Connection details will be shown on both the SSH client and serial console

12.42.1.1.4.3 Sending Data

- **From Serial Console:** Type ASCII strings followed by Enter to send data to connected SSH clients
- **From SSH Client:** Any data sent from the SSH client will be displayed on the serial console

12.42.1.1.5 Configuration

12.42.1.1.5.1 Key Parameters

- `SSH_LISTEN_PORT`: Server listening port (default: 22)
- `CONNECTION_TIMEOUT`: Inactivity timeout in seconds (default: 120)
- `BUFSIZE`: Buffer size for data transmission (default: 4096 bytes)

12.42.1.1.5.2 Security

- Uses NetBurner's built-in SSH key management
- Automatic certificate creation is enabled via `EnableOnboardCertificateCreation()`
- Standard SSH encryption and authentication protocols

12.42.1.1.6 Error Handling

The application includes comprehensive error handling for various TCP and SSH conditions:

- TCP connection errors (timeout, reset, abort)
- SSH negotiation failures
- Session establishment failures
- Socket availability issues
- Connection timeouts

Error states are reported through the serial console with descriptive messages.

12.42.1.1.7 Technical Details

12.42.1.1.7.1 Memory Usage

- SSH Server task stack: 4x standard user task stack size
- Separate receive and transmit buffers (4KB each)
- Efficient memory management with proper cleanup

12.42.1.1.7.2 Network Interface

- Automatically detects and uses the first available network interface
- Displays IP address and interface information on startup
- Supports standard Ethernet connections

12.42.1.1.7.3 Task Management

- SSH server runs at higher priority than UserMain
- Proper task synchronization and resource management
- Clean connection handling and resource cleanup

12.42.1.1.8 Dependencies

- NetBurner RTOS
- NetBurner SSH library ([ssh/NetBurner/NbWolfSsh.h](#))
- Certificate generation library ([crypto/certgen.h](#))
- Standard NetBurner networking and system libraries

12.42.1.1.9 Build Requirements

- NetBurner development environment
- SSH library support enabled in NetBurner configuration
- Certificate generation support

12.42.1.1.10 Troubleshooting

12.42.1.1.10.1 Common Issues

1. **Connection refused:** Verify network connectivity and firewall settings
2. **Authentication failures:** Check SSH key configuration
3. **Timeouts:** Adjust `CONNECTION_TIMEOUT` value if needed
4. **Memory issues:** Monitor task stack usage and buffer sizes

12.42.1.1.10.2 Diagnostic Features

- System diagnostics are enabled for development (consider disabling for production)
- Detailed error reporting through serial console
- Connection state monitoring and logging

12.42.2 SSH Server with User Key

12.42.2.1 SSH Server with User Key Example

12.42.2.1.1 Overview

This NetBurner application demonstrates how to create an SSH server that supports multiple key sources and provides a web-based interface for key management. The application showcases secure SSH connections with user-uploadable keys while maintaining fallback security through default keys.

Additional pdf documentation is located in the example directory

12.42.2.1.2 Features

- **SSH Server:** Accepts secure SSH connections on port 22
- **Multiple Key Sources:**
 - NetBurner SSH library default keys
 - Application compiled-in keys
 - User-uploaded keys via web interface
- **Web Interface:** Upload and manage SSH keys through a browser
- **Key Types Supported:** RSA and ECC (Elliptic Curve Cryptography)
- **File System:** EFFS-STD on-chip flash storage for persistent key storage
- **NTP Time Sync:** Automatic time synchronization for accurate timestamps
- **Serial Debug Interface:** Command-line interface for system diagnostics

12.42.2.1.3 Key Management Hierarchy

The application uses a three-tier key priority system:

1. **User-Uploaded Keys** (Highest Priority): Keys uploaded via web interface
2. **Application Default Keys** (Medium Priority): Keys compiled into the application
3. **Library Default Keys** (Lowest Priority): NetBurner SSH library fallback keys

This ensures the device always has a valid key for SSH connections.

12.42.2.1.4 Hardware Requirements

- NetBurner module with supported flash memory:
 - Spansion AM29LV160B, S29AL016D
 - Atmel AT49BV163D
 - SST39VF040
- Network connection
- Optional: Serial connection for debug interface

12.42.2.1.5 Software Requirements

- NetBurner Development Kit
- NetBurner SSH/SSL Encryption software (optional add-on)
- Compatible platforms: MOD5441X, NANO54415, SB800EX, MODM7AE70, SBE70LC, SOMRT1061, MODRT1171

12.42.2.1.6 Important Configuration Notes

12.42.2.1.6.1 Memory Configuration

CRITICAL**: Before building, you must configure the linker memory range to match your flash chip. Incorrect settings can corrupt the flash memory.

For Spansion AM29LV160B, S29AL016D, or Atmel AT49BV163D:

```
COMPCODEFLAGS = 0xFFC08000 0xFFD80000
```

For SST39VF040:

```
COMPCODEFLAGS = 0xFFC08000 0xFFC70000
```

12.42.2.1.6.2 Library Configuration

Replace `FatFile.a` with `StdFFile.a` in your linker settings to use the on-chip flash file system.

12.42.2.1.7 Usage Instructions

12.42.2.1.7.1 Initial Setup

1. Configure memory settings for your specific flash chip
2. Build and deploy the application to your NetBurner device
3. Use IPSetup to determine the device's IP address
4. Connect to the serial port (115200 baud) for debug output

12.42.2.1.7.2 Web Interface

1. Open a web browser and navigate to the device's IP address
2. View current key status and sources
3. Upload new RSA or ECC keys using the file upload form
4. Reset keys to factory defaults if needed
5. Display public keys for SSH client configuration

12.42.2.1.7.3 SSH Connection

1. Use an SSH client (such as PuTTY) to connect to the device
2. Connect to the device's IP address on port 22
3. Any username/password combination is accepted (for demonstration purposes)
4. The connection timeout is set to 120 seconds

12.42.2.1.7.4 Serial Commands

Access the debug menu via serial connection:

- [1] - EDFS Read/Write Test
- [2] - Display Directory
- [3] - Display TestFile.txt
- [4] - Format Flash (WARNING: Data loss)
- [5] - Display EDFS Space Usage
- [6] - Display System Time
- [7] - Reset to Factory Defaults

12.42.2.1.8 Key Requirements

12.42.2.1.8.1 RSA Keys

- Minimum size: 512 bits
- Maximum size: 4096 bits
- Format: OpenSSH/OpenSSL PEM format
- Must be unencrypted (no passphrase)

12.42.2.1.8.2 ECC Keys

- Minimum size: 192 bits
- Maximum size: 256 bits (384, 521, and ED25519 available with additional defines)
- Format: OpenSSH/OpenSSL PEM format
- Must be unencrypted (no passphrase)

12.42.2.1.9 File Structure

12.42.2.1.9.1 Core Application Files

- `main.cpp` - Main application thread and system initialization
- `sshuser.cpp/h` - SSH authentication and key management functions
- `web.cpp` - Web server interface for key upload and display
- `nvsettings.cpp/h` - Non-volatile settings management

12.42.2.1.9.2 Key Files

- `permanentkeyrsa.h` - Default RSA key compiled into application
- `permanentkeyecc.h` - Default ECC key compiled into application
- `openSsh*.key` - Example key files for testing

12.42.2.1.9.3 Web Interface

- `index.html` - Main web page for key management

12.42.2.1.9.4 File System Support

- `FileSystemUtils.cpp/h` - EDFS-STD file system utilities
- `fs_main.cpp/h` - File system initialization
- `edfs_time.cpp/h` - File timestamps and NTP support

12.42.2.1.10 Security Considerations

- The example uses simple authentication (any username/password where they don't match)
- In production, implement proper user authentication
- The web interface is unencrypted HTTP - consider HTTPS for production use
- Private keys are stored in flash memory - ensure physical security of devices
- Default keys are compiled into the application - replace with your own keys for production

12.42.2.1.11 Troubleshooting

12.42.2.1.11.1 Common Issues

1. **Flash Corruption:** Usually caused by incorrect COMPCODEFLAGS settings
2. **Key Upload Failures:** Verify key format and encryption status
3. **Connection Timeouts:** Check network configuration and firewall settings
4. **File System Errors:** May require formatting the EDFS partition

12.42.2.1.11.2 Debug Output

The serial interface provides detailed debug information including:

- SSH connection status
- Key validation results
- File system operations
- Network configuration
- System diagnostics

12.42.2.1.12 Development Notes

This example demonstrates several NetBurner development concepts:

- EDFS-STD flash file system usage
- Multi-part form handling in web applications
- SSH server implementation
- Non-volatile parameter storage
- Task creation and management
- Network service integration

12.42.3 SSH Client

12.42.3.1 SSH Minimal Client Example

12.42.3.1.1 Overview

This NetBurner application demonstrates a minimal SSH client implementation that connects to an SSH server and reads data continuously until the connection is closed. The client establishes an encrypted SSH connection and can be used as a foundation for building more complex SSH-based applications.

12.42.3.1.2 Features

- Establishes SSH connection to a remote server
- User authentication with password
- Continuous data reading from SSH server
- Automatic reconnection on connection failure
- Built-in diagnostics and logging
- Certificate and key generation support

12.42.3.1.3 Configuration

12.42.3.1.3.1 Server Settings

The application is configured to connect to an SSH server using the following constants in `main.cpp`:

```
#define SSH_SERVER_NAME "10.1.1.131"    // Target SSH server IP address
#define SSH_REMOTE_PORT 22             // SSH server port (standard port 22)
#define CONNECTION_TIMEOUT 120         // Connection timeout in seconds
```

12.42.3.1.3.2 Authentication

The application uses password-based authentication with these default credentials:

- Username: "guest" (specified in `SshConnect` call)
- Password: "TestPassword" (set in `ClientUserAuth` function)

12.42.3.1.4 Hardware Requirements

- NetBurner device with network connectivity
- Active network connection ([DHCP](#) or static IP)
- USB or Serial connection for monitoring

12.42.3.1.5 Setup Instructions

1. Configure Target Server

- Update `SSH_SERVER_NAME` with your SSH server's IP address
- Ensure the SSH server is running and accessible on port 22
- Verify firewall settings allow SSH connections

2. Compile and Deploy

- Compile the application using NetBurner development tools
- Load the compiled firmware onto your NetBurner device

3. Monitor Operation

- Connect to the device using MTTTY via USB or Serial port
- View connection status and received data in the terminal

12.42.3.1.6 Code Structure

12.42.3.1.6.1 Main Components

- **UserMain():** Primary application entry point
 - Initializes system components
 - Configures SSH client settings
 - Manages connection loop
- **ClientUserAuth():** Authentication callback
 - Called when server requests user credentials
 - Returns username and password for authentication

12.42.3.1.6.2 Key Functions

- [NbSshInit\(\)](#): Initializes SSH subsystem
- [SshConnect\(\)](#): Establishes SSH connection to server
- [ReadWithTimeout\(\)](#): Reads data from SSH connection with timeout
- [EnableOnboardCertificateCreation\(\)](#): Enables automatic certificate generation

12.42.3.1.7 Operation Flow

1. System initialization and network setup
2. SSH subsystem initialization
3. Connection attempt to configured server
4. User authentication when prompted
5. Continuous data reading loop
6. Connection monitoring and automatic reconnection

12.42.3.1.8 Buffer Configuration

The application uses a 4KB receive buffer:

```
#define RX_BUF_SIZE (4096)
char RXBuffer[RX_BUF_SIZE];
```

12.42.3.1.9 Task Stack

SSH operations use an extended stack size:

```
#define SSHSERVER_TASK_STACK_SIZE (USER_TASK_STK_SIZE * 4)
```

12.42.3.1.10 Troubleshooting

12.42.3.1.10.1 Common Issues

- **Connection Failed:** Verify server IP address and network connectivity
- **Authentication Failed:** Check username/password credentials
- **Network Timeout:** Ensure [DHCP](#) is working or configure static IP
- **Certificate Errors:** Verify certificate generation is enabled

12.42.3.1.10.2 Debug Output

The application provides detailed logging including:

- SSH initialization status
- Connection attempts and results
- Data received from server
- Connection closure notifications

12.42.3.1.11 Customization

12.42.3.1.11.1 Modifying Authentication

Update the `ClientUserAuth()` function to implement different authentication methods:

```
int ClientUserAuth(const NBString &usernamePtr, NBString &passwordPtr)
{
    // Custom authentication logic here
    passwordPtr = "YourPassword";
    return 1; // Return 1 for success, 0 for failure
}
```

12.42.3.1.11.2 Adding Data Processing

Extend the data reading loop in `UserMain()` to process received data:

```
if (n != 0)
{
    printf("Read %d bytes: %s\n", n, RXBuffer);
    // Add custom data processing here
}
```

12.42.3.1.12 Dependencies

- NetBurner SSH library ([ssh/NetBurner/NbWolfSsh.h](#))
- Certificate generation support ([crypto/certgen.h](#))
- Standard NetBurner system libraries

12.42.3.1.13 Notes

- The application includes system diagnostics which should be disabled for production use
- Automatic certificate creation is enabled for SSL/TLS support
- The client will continuously attempt to reconnect if the connection is lost
- All received data is null-terminated and printed to the console

12.42.4 SSH Server with User Authorization

12.42.4.1 SSH Server with User Authorization

A NetBurner embedded application that implements a secure SSH server with customizable user authentication management.

12.42.4.1.1 Overview

This application creates an SSH server that listens on TCP port 22 for incoming SSH client connections. The server uses an automatically generated onboard key for encryption and implements a flexible user authentication system that supports both password and key-based authentication.

12.42.4.1.2 Key Features

- **SSH Server:** Full SSH server implementation listening on port 22
- **User Authentication Management:** Custom user authentication system with support for:
 - Password-based authentication
 - Key-based authentication
 - User authorization levels
- **Automatic Certificate Generation:** Self-signed certificates generated on first boot
- **Persistent User Storage:** User credentials stored in device UserParam space
- **Interactive Management:** Serial console interface for user management
- **Connection Timeout:** Configurable connection timeout (default: 120 seconds)

12.42.4.1.3 Architecture

12.42.4.1.3.1 Core Components

1. **main.cpp:** Main application with SSH server task and user interface
2. **UserAuth.cpp:** User authentication manager implementation
3. [UserAuth.h](#): Authentication interface definitions

12.42.4.1.3.2 Key Classes and Functions

- `UserAuthManager`: Manages user authentication records and validation
- `SshServerTask()`: Main SSH server listening and connection handling
- `AuthenticateUser()`: SSH authentication callback function
- `SaveAuthRecordsUserParam()` / `LoadAuthRecordsUserParam()`: Persistent storage functions

12.42.4.1.4 Configuration

12.42.4.1.4.1 Default Settings

```
#define SSH_LISTEN_PORT 22      // SSH server port
#define CONNECTION_TIMEOUT 120 // Connection timeout in seconds
#define BUFSIZE (4096)        // Buffer size for data transfer
```

12.42.4.1.4.2 Authentication Types

- `eAuthTypePassword (1)`: Password-based authentication
- `eAuthTypeKey (2)`: SSH key-based authentication

12.42.4.1.5 Usage

12.42.4.1.5.1 Building and Deployment

1. Compile the application using NetBurner development tools
2. Load the firmware onto your NetBurner device
3. Connect via MTTTY to the USB/Serial port for configuration

12.42.4.1.5.2 Initial Setup

1. **Certificate Generation**: On first boot, the device automatically generates SSL certificates
2. **Network Connection**: Wait for active network connection
3. **SSH Server Start**: Server begins listening on port 22

12.42.4.1.5.3 User Management Commands

Access these commands via the serial console when no SSH client is connected:

- **A**: Add new user (prompts for username, auth value, and auth type)
- **D**: Delete existing user (prompts for username)
- **L**: List all configured users
- **?**: Show command list

12.42.4.1.5.4 SSH Client Connection

1. Use any SSH client (PuTTY, OpenSSH, etc.)
2. Connect to the NetBurner device's IP address on port 22
3. Authenticate using configured username and password/key
4. Interactive session allows bidirectional communication

12.42.4.1.6 Security Features

12.42.4.1.6.1 Certificate Management

- Automatic onboard certificate and key generation
- One-year certificate validity period
- Optional automatic certificate regeneration (enable `ENABLE_AUTOCERT_REGEN`)

12.42.4.1.6.2 User Authentication

- Secure password hashing using SHA-256
- Support for SSH public key authentication
- Configurable user authorization levels
- Persistent storage of authentication records

12.42.4.1.6.3 Connection Security

- Full SSH protocol implementation
- Encrypted communication channels
- Connection timeout protection
- Proper session management

12.42.4.1.7 Data Storage

User authentication records are stored in JSON format in the device's UserParam space:

```
{
  "0": {
    "un": "username",
    "at": 1,
    "al": 0,
    "ah": [hash_array]
  }
}
```

Where:

- `un`: Username
- `at`: Authentication type (1=password, 2=key)
- `al`: Authorization level
- `ah`: SHA-256 hash array

12.42.4.1.8 Error Handling

The application includes comprehensive error handling for:

- TCP connection errors (timeout, reset, abort)
- SSH negotiation failures
- Authentication failures
- Certificate generation errors
- User management errors

12.42.4.1.8.1 Common Error Codes

- `TCP_ERR_TIMEOUT`: Connection timeout
- `TCP_ERR_CON_RESET`: Connection reset by peer
- `SSH_ERROR_FAILED_NEGOTIATION`: SSH handshake failure
- `SSH_ERROR_FAILED_SESSION_FAILED`: SSH session establishment failure

12.42.4.1.9 Development Notes

12.42.4.1.9.1 Memory Usage

- SSH Server task stack: `USER_TASK_STK_SIZE * 4`
- Buffer allocation: 4KB for both TX and RX
- JSON buffer: 1.5KB for user record storage

12.42.4.1.9.2 Task Priority

SSH Server task runs at `MAIN_PRIO - 1` (higher priority than UserMain) to ensure responsive connection handling.

12.42.4.1.9.3 Dependencies

- NetBurner SSH library ([ssh/NetBurner/NbWolfSsh.h](#))
- Certificate generation ([crypto/certgen.h](#))
- JSON parsing ([json_lexer.h](#))
- Standard NetBurner networking and serial libraries

12.42.4.1.10 Troubleshooting

12.42.4.1.10.1 Connection Issues

1. Verify network connectivity
2. Check if SSH server task started successfully
3. Confirm firewall settings allow port 22
4. Validate certificate generation completed

12.42.4.1.10.2 Authentication Problems

1. List users to verify account exists
2. Check authentication type matches client configuration
3. Verify user authorization level
4. Review SSH client logs for specific errors

12.42.4.1.10.3 Performance Considerations

- Maximum 5 concurrent connection queue
- 120-second connection timeout
- 4KB buffer size may limit large data transfers
- JSON parsing occurs on user record changes

12.42.4.1.11 Example Session

Starting SSH Server Task, Listening for SSH connection on port 22

```
Waiting for connection on port 22...
Connected to: 192.168.1.100:54321
Welcome to the NetBurner SSH Server
You are connected to IP Address 192.168.1.50, port 22
```

[Interactive SSH session follows...]

12.42.5 SecureSerToEthFactoryApp

12.42.5.1 NetBurner Secure Serial-to-Ethernet Factory Application

12.42.5.1.1 Overview

The NetBurner Secure Serial-to-Ethernet Factory Application is a comprehensive embedded application that provides secure bi-directional communication between serial interfaces and Ethernet networks. This application is pre-programmed into NetBurner Serial-to-Ethernet devices and serves as a bridge between various serial protocols and network connectivity. The application pre-programmed into NetBurner Serial to Ethernet devices. This application source code is provided for customers that wish to customize the application.

Interfaces:

- RS-232
- RS-422
- RS-485
- CAN Bus
- I2C
- Ethernet
- Wifi

Note

Please refer to the Secure Serial Factory App Users Manual for a detailed description of this application at www.netburner.com.

12.42.5.1.2 Application Architecture

12.42.5.1.2.1 Entry Point

The application starts in `UserMain()` (located in `src/main.cpp`), which is the standard entry point for all NetBurner applications, rather than the typical C/C++ `main()` function.

12.42.5.1.2.2 Core Features

Multi-Protocol Support:**

- TCP/IP communication
- UDP communication
- SSH (Secure Shell) connections
- TLS/SSL encrypted connections
- Multiple serial interface protocols (RS-232, RS-485, CAN Bus, I2C)

Network Capabilities:**

- DHCP and static IP addressing

- WiFi connectivity support
- Network Time Protocol (NTP) synchronization
- DNS resolution
- HTTPS web server interface
- Security Features:**
- SSL/TLS encryption for secure data transfer
- SSH authentication and key management
- Certificate and key management system
- Configurable user authentication

12.42.5.1.3 Supported Hardware Platforms

- **SBE70LC** - NetBurner SBE70LC modules
- **SB800EX** - NetBurner SB800EX modules
- **NANO54415** - NetBurner NANO54415 modules
- **MODM7AE70** - NetBurner MODM7AE70 modules
- **SOMRT1061** - NetBurner SOMRT1061 modules
- **MOD5441X** - NetBurner MOD5441X modules

12.42.5.1.4 Key Source Files

12.42.5.1.4.1 Main Application

- `**src/main.cpp**` - Main application entry point and core logic
- `**src/nbfactory.h**` - Factory configuration definitions and platform-specific settings

12.42.5.1.4.2 Serial Communication

- `**src/serialrecord.cpp/h**` - Core serial port management and data handling
- `**src/serialburnerdata.h**` - Serial port configuration structures
- `**src/atparserforssh.cpp**` - AT command parser for SSH configuration

12.42.5.1.4.3 Network & Security

- `**src/sshuser.cpp/h**` - SSH user authentication and key management
- `**src/ssluser.cpp/h**` - SSL/TLS user authentication
- `**src/certificatekey.cpp**` - Certificate and key management
- `**src/ServerKey.cpp**` - Server key handling
- `**src/ServerCert.cpp**` - Server certificate handling

12.42.5.1.4.4 Hardware Interface

- `**src/gpioserver.cpp/h**` - GPIO server for pin control
- `**src/i2cserver.cpp/h**` - [I2C](#) server implementation
- `**src/i2cfuns.cpp/h**` - [I2C](#) utility functions
- `**src/analog.cpp/h**` - Analog input handling

12.42.5.1.4.5 Web Interface

- `**src/configweb.cpp**` - Web-based configuration interface
- `**src/configweb_wifi.cpp**` - WiFi-specific web configuration
- `**src/formtools.cpp/h**` - HTML form processing utilities
- `**src/post.cpp**` - HTTP POST request handling

12.42.5.1.4.6 File System & Storage

- `**src/effs_std.cpp/h**` - EFFS (Embedded Flash File System) STD implementation
- `**src/effsStdPartition.cpp**` - Flash partition management
- `**src/FileSystemUtils.cpp/h**` - File system utility functions
- `**src/ftp_fs.cpp/h**` - FTP file system interface

12.42.5.1.4.7 System Utilities

- `**src/fdtimer.cpp/h**` - File descriptor timer management
- `**src/fnbrtosdump.cpp**` - RTOS debugging and diagnostics
- `**src/autoupdate.cpp**` - Remote firmware update functionality

12.42.5.1.5 Configuration and Web Interface

The application includes a comprehensive web-based configuration interface with the following pages:

12.42.5.1.5.1 Network Configuration

- `**html/network.html**` - Main network settings
- `**html/tcp.html**` - TCP protocol configuration
- `**html/udp.html**` - UDP protocol configuration
- `**html/wifi.html**` - WiFi connectivity settings

12.42.5.1.5.2 Security Configuration

- `**html/ssh.html**` - SSH server configuration
- `**html/ssl.html**` - SSL/TLS settings
- `**html/certificate.html**` - Certificate management
- `**html/keys.html**` - Cryptographic key management

12.42.5.1.5.3 Serial Port Configuration

- `**html/serial.html**` - Serial port settings
- `**html/advancedserial.html**` - Advanced serial configuration
- `**html/i2c.html**` - [I2C](#) interface settings

12.42.5.1.5.4 System Management

- `**html/advanced.html**` - Advanced system settings
- `**html/gpio.html**` - GPIO pin configuration
- `**html/password.html**` - User authentication settings

12.42.5.1.6 Build Configuration

12.42.5.1.6.1 Makefile Structure

The project uses a standardized NetBurner makefile with the following key components:

- **Target Name:** `SecureSerToEthFactoryApp`
- **Supported Platforms:** All major NetBurner platforms
- **Dependencies:** EDFS STD file system library
- **Generated Files:** HTML content is compiled into `src/htmldata.cpp`

12.42.5.1.6.2 Memory Configuration

Platform-specific memory settings:

- **MOD5441X/NANO54415/SB800EX:** 1MB reserved for EDFS STD file system
- **MODM7AE70/SBE70LC:** Application memory relocation disabled

12.42.5.1.7 Key Features Implementation

12.42.5.1.7.1 Data Flow Management

The `SerialRecord` class manages bidirectional data flow between serial ports and network connections, implementing:

- Circular buffering for serial-to-network and network-to-serial data
- Connection state management
- Timeout handling
- Keep-alive functionality

12.42.5.1.7.2 Protocol Handling

The application supports multiple connection modes:

- **Server Mode:** Listens for incoming connections
- **Client Mode:** Initiates outbound connections
- **UDP Mode:** Connectionless datagram communication
- **SSH Mode:** Secure shell connections with authentication

12.42.5.1.7.3 Security Implementation

- Certificate and key management with support for RSA and ECDSA
- SSL/TLS encryption for web interface and data connections
- SSH authentication with public key support
- Configurable user credentials and access control

12.42.5.1.8 Configuration Methods

The device can be configured through multiple interfaces:

1. **Web Interface (HTTP/HTTPS)** - Full-featured browser-based configuration
2. **Configuration Interface** - NetBurner's standard configuration protocol
3. **AT Commands** - Serial port based command interface
4. **Remote Updates** - Over-the-air firmware updates

12.42.5.1.9 Development and Customization

This source code is provided to enable customers to customize the factory application for specific requirements. Key customization points include:

- **Protocol Extensions:** Add support for custom protocols
- **Hardware Integration:** Interface with specific sensors or actuators
- **Security Policies:** Implement custom authentication mechanisms
- **Data Processing:** Add filtering, transformation, or logging capabilities

12.42.5.1.10 System Requirements

- NetBurner development environment with SDK
- Compatible NetBurner hardware platform
- Network connectivity for web-based configuration
- Optional: WiFi module for wireless connectivity

12.42.5.1.11 Additional Resources

For detailed documentation and user manuals, visit www.netburner.com and refer to the Secure Serial Factory App Users Manual.

12.43 SSL/TLS

Obtaining certificates:

- [Easy SSL/TLS Onboard Certificates](#)
- [ACME Certificate Generation \(Easy Servlet\)](#)

SSL/TLS examples. Note that all references to SSL represent TLS.

- [FTPS Server](#)
- [HTML File Post](#)
- [HTML Form Post](#)
- [HTTPS Dual Cert](#)
- [HTTPS Upload Cert](#)
- [HTTPS Web Server Demo](#)
- [HTTPS GET Request](#)
- [SSL/TLS Client](#)
- [SSL/TLS Client Certificate](#)
- [SSL/TLS Client Verify Peer - Basic](#)
- [SSL/TLS Client Verify Peer - EDFS-STD](#)
- [SSL/TLS Receive Mail \(POP3\)](#)
- [SSL/TLS Send Mail](#)
- [SSL/TLS Send Mail w/ Attachment](#)
- [SSL/TLS Send Mail w/ EDFS Attchment](#)
- [SSL/TLS Server](#)
- [HTTPS Configuration Mirror](#)

12.43.1 Easy SSL/TLS Onboard Certificates

For ACME (Public Certificate Authority) Certificate Generation:

- [ACME Certificate Generation \(Easy Servlet\)](#)

For Self-Signed Certificate Generation:

- [On-board Cert Generation - Simple](#)
- [On-board Cert Generation - Advanced](#)
- [On-board Cert Generation - Compiled Certificate Authority](#)

12.43.1.1 ACME Certificate Generation (Easy Servlet)

```
<NNDK install>\examples\SSL\acmeservlet
```

12.43.1.1.1 ACME Certificate Generation (Easy Servlet)

A NetBurner NNDK SSL/TLS example application that demonstrates automatic ACME certificate generation using Let's Encrypt or other ACME-compliant certificate authorities.

12.43.1.1.1.1 Overview

This application provides a web-based demonstration of automatic SSL/TLS certificate generation using the ACME (Automatic Certificate Management Environment) protocol as defined in RFC 8555. The application includes a built-in web server that serves both HTTP and HTTPS content, with automatic certificate provisioning through ACME providers.

SSL/TLS example program that demonstrates getting an ACME cert automatically. It will only work for a device with Internet access with a public IP address and DNS entry. Make sure to change the `DEFAULT_CERT_COM_NAME` constant in `main.cpp` to match.

This example makes use of the [acmeRFC8555Servlet.h](#) builtin to the NNDK which reduces how much code is required to add the functionality to your project and will more easily take advantage of future NNDK updates.

12.43.1.1.1.2 Requirements

- NetBurner device with public IP address
- Public DNS entry pointing to the device
- NNDK (NetBurner Network Development Kit)
- Device must be accessible from the internet for ACME validation

12.43.1.1.1.3 Features

- Automatic SSL/TLS certificate generation via ACME protocol
- Support for multiple ACME providers (Let's Encrypt, BuyPass, custom)
- Web server with both HTTP (port 80) and HTTPS (port 443) support
- Access control based on SSL/TLS connection status
- Certificate reissue functionality
- NTP client integration for accurate time synchronization
- Web-based status and control interface

12.43.1.1.1.4 Configuration

Certificate Information

Before running the application, update the following constants in `main.cpp` to match your environment:

```
const char * DEFAULT_CERT_COUNTRY= "US";
const char * DEFAULT_CERT_STATE= "CA";
const char * DEFAULT_CERT_LOCALITY= "San Diego";
const char * DEFAULT_CERT_ORG= "Netburner";
const char * DEFAULT_CERT_UNIT= "Demo";
const char * DEFAULT_CERT_EMAIL= "changeme@example.com";
const char * DEFAULT_CERT_ALT_NAMES= "test.example.com,example.com";
const char * DEFAULT_CERT_COM_NAME= "test.example.com";
```

Important*: The `DEFAULT_CERT_COM_NAME` must match your device's public DNS name.

ACME Provider Selection

The application supports multiple ACME providers. Choose one by uncommenting the appropriate line:

```
/* Production Let's Encrypt (default): */
LetsEncryptAcmeServletObject AcmeClient;

/* Let's Encrypt Staging (for testing): */
// LetsEncryptAcmeServletObject AcmeClient(true);

/* BuyPass ACME: */
// BuyPassAcmeServletObject AcmeClient;

/* Custom ACME server: */
// AcmeServletObject AcmeClient("https://your-acme-server.com/dir");
```

12.43.1.1.1.5 Building and Running

1. Ensure your NetBurner device has a public IP and DNS entry
2. Update the certificate configuration constants
3. Build the application using the NNDK toolchain
4. Deploy to your NetBurner device
5. Access the web interface via HTTP or HTTPS

12.43.1.1.1.6 Web Interface

The application provides several web endpoints:

- `/index.html` - Main interface page
- Counter functionality - Displays page reload count
- SSL status indicators - Shows connection security status
- Certificate management controls

Access Control

The application implements access control based on connection type:

- Access level 0: Available to all connections
- Access level 1: Requires SSL/TLS connection
- Other levels: Default allow access

12.43.1.1.1.7 Runtime Options

When the application starts, you can:

- Press 'R', 'r', 'Y', or 'y' to reissue certificates
- Monitor certificate status through diagnostic output
- View ACME client state information

12.43.1.1.1.8 ACME Certificate Process

1. Application generates a Certificate Signing Request (CSR)
2. Contacts the ACME server to request a certificate
3. Completes domain validation challenges
4. Receives and installs the signed certificate
5. Automatically renews certificates before expiration

12.43.1.1.1.9 Key Components

- **ACME Client:** Handles certificate requests and renewals
- **Web Server:** Serves HTTP/HTTPS content
- **NTP Client:** Maintains accurate time for certificate validation
- **SSL Context:** Manages SSL/TLS connections and certificates
- **Access Control:** Enforces connection security requirements

12.43.1.1.1.10 Troubleshooting

Common Issues

1. **Certificate generation fails:** Verify public DNS and IP accessibility
2. **ACME validation fails:** Check firewall settings and port accessibility
3. **Time sync issues:** Ensure NTP client can reach time servers
4. **Domain mismatch:** Verify `DEFAULT_CERT_COM_NAME` matches DNS

Debug Options

Enable diagnostic output:

```
AcmeClient.SetDiag(true);
```

Force certificate reissue:

```
AcmeClient.Delete_Everything_Restart();
```

12.43.1.1.1.11 Security Notes

- Certificates are automatically managed by the ACME client
- Production deployment should disable system diagnostics
- Consider implementing additional access controls for production use
- Monitor certificate expiration and renewal processes

12.43.1.1.1.12 License

This example code is provided as part of the NetBurner NNDK and follows the NetBurner licensing terms.

12.43.1.1.1.13 Support

For technical support and documentation:

- NetBurner NNDK Documentation
- NetBurner Support Portal
- ACME RFC 8555 specification

12.43.1.2 On-board Cert Generation - Simple

12.43.1.2.1 On-board Certificate Generation - Simple

A NetBurner embedded application that demonstrates automatic SSL/TLS certificate generation and secure web server functionality.

12.43.1.2.1.1 Overview

This application showcases NetBurner's capability to automatically generate SSL certificates on embedded devices, enabling both HTTP and HTTPS web interfaces. The application sets up a secure web server with automatic certificate creation, time synchronization, and provides multiple access methods for device configuration.

This example shows how to enable auto-generation of self-signed certificates. The certificate will enable secure communication to the system configuration web server and the HTTPS web server of the application. On-board generation of certificates are an alternative to creating your own self-signed certificates and uploading them to your NetBurner device. This simple example should cover most use cases. Please refer to the other examples for additional functionality.

Auto-generated certificates will operate as follows:

- If enabled and no certificate exists, a certificate will be created on first call to `SSL_accept()` or `SslInitServer()`.
- The Common Name (CN) will be the IP address of the device. Alternate names are also supported.
- The application must ensure it has the correct system time, otherwise the certificate dates will be invalid.
- Calling the function `EnableOnboardCertificateCreation()` enables certificate generation.
- If `ENABLE_AUTOCERT_REGEN` is defined in `predef.h`, the certificate will update when it expires, creating a new 1 year certificate. Generation will occur on next SSL/TLS access.
- If `ENABLE_AUTO_CERT_REGEN` is defined in `predef.h`, `AUTO_CERT_GEN_CHECK` will dictate how frequently a certificate is checked for expiration. The default value is one minute.
- The certificate will update if the IP address changes to handle environments such as `DHCP` address assignments.

If the system time is incorrect, the certificate will be invalid.

Additional Examples:

Examples of a HTTPS server with a compiled-in certificate:

- `<NNDK install>\examples\ssl\sslserver`
- `<NNDK install>\examples\ssl\SslWebDemo`

Examples of uploading a certificate/key pair manually:

- `<NNDK install>\examples\ssl\HttpsDualCert`
- `<NNDK install>\examples\ssl\HttpsUploadCert`

Example of handling additional timezones

- `<NNDK install>\examples\TimeFunctions`

12.43.1.2.1.2 Features

- **Automatic Certificate Generation:** Creates SSL certificates on-board without manual intervention
- **Dual Protocol Support:** Serves both HTTP (port 80) and HTTPS (port 443) simultaneously
- **Time Synchronization:** Uses NTP for accurate time setting (required for certificate validity)
- **Timezone Support:** Configurable timezone with automatic daylight savings calculation
- **System Diagnostics:** Built-in diagnostic capabilities for development and troubleshooting
- **Multiple Access Methods:** Provides various ways to access the device via web interfaces

12.43.1.2.1.3 Hardware Requirements

- NetBurner embedded device with network connectivity
- Active network connection for NTP time synchronization
- Internet access (optional, for discover.netburner.com service)

12.43.1.2.1.4 Software Dependencies

This application uses the NetBurner NNDK (NetBurner Network Development Kit) and requires the following libraries:

- [nbrtos.h](#) - NetBurner Real-Time Operating System
- [crypto/certgen.h](#) - Certificate generation functionality
- [netinterface.h](#) - Network interface management
- [nbtime.h](#) - Time management utilities
- [timezones.h](#) - Timezone support

12.43.1.2.1.5 Configuration

Time Zone Setup

The application defaults to Pacific Standard Time (PST). To change the timezone, modify the parameter in `main.c`:

```
cpp:
setSystemTime("PST"); // Change "PST" to your desired timezone
```

Note

Refer to TimeFunctions example for timezone handling

Supported timezone abbreviations include: PST, MST, CST, EST, and others defined in the NetBurner timezone database.

Certificate Configuration

Certificate generation is automatically enabled via:

```
EnableOnboardCertificateCreation();
```

Server Configuration

The secure configuration server is enabled without forcing TLS mode:

```
EnableSecureConfigServer(false);
```

12.43.1.2.1.6 Access Methods

Once running, the device provides several access points:

Web Interfaces

- **HTTP Application:** [http://\[device-ip\]](http://[device-ip])
- **HTTPS Application:** [https://\[device-ip\]](https://[device-ip])
- **HTTP System Config:** [http://\[device-ip\]:20034](http://[device-ip]:20034)
- **HTTPS System Config:** [https://\[device-ip\]:20034](https://[device-ip]:20034)

Discovery Service

- **NetBurner Discovery:** <http://discover.netburner.com> (requires internet access)

12.43.1.2.1.7 Time Synchronization

The application requires accurate time for certificate generation and validation:

1. **NTP Synchronization** (Primary): Automatically attempts to sync with NTP pool servers
2. **Manual Entry** (Fallback): If NTP fails, prompts for manual time entry via serial console

Manual Time Entry

If NTP synchronization fails, the application will prompt for manual time entry through the serial debug port. You'll be asked to provide:

- Year (YYYY format)
- Month (1-12)
- Day of month (1-31)
- Hour (0-23)
- Minute (0-59)
- Seconds (0-59)
- GMT or Local time preference

12.43.1.2.1.8 Usage

1. **Build and Deploy**: Compile using NetBurner development tools and deploy to target hardware
2. **Network Connection**: Ensure device has active network connectivity
3. **Monitor Serial Output**: Connect to serial debug port to monitor startup and get IP address
4. **Access Web Interface**: Use any of the provided URLs to access the device
5. **Certificate Verification**: HTTPS connections will use the automatically generated certificate

12.43.1.2.1.9 Serial Interface

The application provides an interactive serial interface:

- Displays IP address and access links on startup
- Shows time synchronization status
- Press any key to redisplay connection information
- Provides manual time entry interface if needed

12.43.1.2.1.10 Development Notes

- System diagnostics are enabled by default (`EnableSystemDiagnostics()`) - consider removing for production builds
- The application waits up to 5 seconds for network connectivity before proceeding
- Certificate generation happens automatically during startup
- Time synchronization is critical for SSL certificate validity

12.43.1.2.1.11 Troubleshooting

Common Issues

1. **Certificate Warnings:** Browsers may show security warnings for self-signed certificates - this is normal behavior
2. **NTP Sync Failure:** Check network connectivity and firewall settings if NTP synchronization fails
3. **Time Zone Issues:** Verify timezone abbreviation is supported in NetBurner's timezone database
4. **Network Access:** Ensure device IP is accessible from client machines

Debug Information

Monitor the serial debug output for:

- Network connectivity status
- NTP synchronization results
- Certificate generation status
- IP address assignments
- Error messages and diagnostics

12.43.1.3 On-board Cert Generation - Advanced

12.43.1.3.1 NetBurner On-board Certificate Generation - Advanced

12.43.1.3.1.1 Overview

This application demonstrates advanced on-board SSL certificate generation capabilities for NetBurner embedded devices. It provides interactive control over certificate lifecycle management, including creation, validation, expiration checking, and deletion of self-signed certificates.

The simple auto-generate self-signed certificate example is what is normally used in applications. This example provide advanced options, such as interactive control to:

- Generate the certificate
- Delete the certificate
- View certificate expiration date
- Check for a valid certificate
- Manually enter the system time

12.43.1.3.1.2 Features

Certificate Management

- **Automatic Certificate Generation:** Creates self-signed certificates with configurable parameters
- **Certificate Validation:** Verifies certificate and key pair validity
- **Expiration Monitoring:** Checks certificate expiration status and displays expiration dates
- **Certificate Deletion:** Removes existing certificates (requires reboot to take effect)

Time Management

- **NTP Time Synchronization:** Automatically syncs system time with NTP servers
- **Manual Time Entry:** Fallback option for manual time configuration
- **Timezone Support:** Configurable timezone settings with automatic daylight savings

Network Connectivity

- **Dual Protocol Support:** Serves both HTTP (port 80) and HTTPS (port 443)
- **Web Interface:** Provides web-based access to the application
- **System Configuration:** Access to NetBurner system configuration pages
- **Network Discovery:** Integration with NetBurner discovery service

12.43.1.3.1.3 Application Structure

Main Components

`main.cpp`

The primary application file containing:

- **Certificate Generation Logic:** `createCertAndKey()` function
- **Certificate Validation:** `certExpired()` and `printCertExpiration()` functions
- **Interactive Menu System:** Serial console interface for user interaction
- **Web Server Integration:** HTTP/HTTPS server startup and link display
- **Network Configuration:** IP address detection and interface management

`TimeUtil.cpp / TimeUtil.h`

Time management utilities including:

- **NTP Synchronization:** Automatic time setting from network time servers
- **Manual Time Entry:** Interactive time configuration via serial console
- **Timezone Management:** Support for various timezones with DST calculation
- **Time Display Functions:** Formatted time output utilities

`CaCrt.cpp`

Auto-generated certificate data file containing:

- **Certificate Array:** Binary certificate data (501 bytes)
- **Certificate Length:** Size definition for the embedded certificate

12.43.1.3.1.4 Certificate Configuration

The application generates certificates with the following default parameters:

```
Country: US
State: California
Locality: San Diego
Organization: NetBurner
Unit: CodeDemo
Email: Sales@NetBurner.com
Common Name: MyNetburner
Validity Period: 1 year
```

Subject Alternative Names (SAN)

The certificate includes both IP address and DNS entries based on the device's network configuration.

12.43.1.3.1.5 Interactive Menu Options

The application provides a serial console menu with the following options:

1. **Check Certificate Expiration:** Determines if the current certificate is expired
2. **Show Certificate Expiration Date:** Displays the exact expiration timestamp
3. **Generate New Certificate:** Creates a fresh certificate/key pair
4. **Erase Certificate:** Removes the current certificate (requires reboot)
5. **Display System Time:** Shows current system time and timezone
6. **Reboot:** Restarts the device
7. **? - Display Menu:** Shows the menu options

12.43.1.3.1.6 Network Access Points

Once running, the device can be accessed through multiple endpoints:

Discovery Service

- **URL:** <http://discover.netburner.com>
- **Description:** NetBurner's network discovery service

Application Pages

- **HTTP:** [http://\[device-ip\]](http://[device-ip])
- **HTTPS:** [https://\[device-ip\]](https://[device-ip])

System Configuration

- **HTTP:** [http://\[device-ip\]:20034](http://[device-ip]:20034)
- **HTTPS:** [https://\[device-ip\]:20034](https://[device-ip]:20034)

12.43.1.3.1.7 Security Features

Certificate Generation

- **ECC Key Generation:** Uses Elliptic Curve Cryptography for key pairs
- **Self-Signed Certificates:** Generates certificates signed by the device itself
- **Secure Storage:** Certificates and keys are stored in device flash memory
- **Random Number Generation:** Utilizes hardware random number generation

SSL/TLS Support

- **Dual Protocol:** Supports both HTTP and HTTPS simultaneously
- **Certificate Validation:** Built-in certificate and key validation
- **Secure Configuration:** HTTPS access to system configuration pages

12.43.1.3.1.8 Prerequisites

Hardware Requirements

- NetBurner embedded device with SSL/TLS capability
- Network connectivity (Ethernet)
- Sufficient flash memory for certificate storage

Software Requirements

- NetBurner NNDK (NetBurner Network Development Kit)
- Compatible compiler toolchain
- Serial console access for interactive features

Network Requirements

- Active network connection for NTP synchronization
- Internet access (optional, for NTP and discovery service)

12.43.1.3.1.9 Build and Deployment

1. **Compile:** Build the application using the NetBurner development environment
2. **Deploy:** Upload the compiled application to the target device
3. **Network Setup:** Ensure the device has network connectivity
4. **Time Synchronization:** The application will attempt NTP sync on startup
5. **Certificate Generation:** Certificates can be generated via the interactive menu

12.43.1.3.1.10 Usage Instructions

Initial Setup

1. Connect to the device via serial console
2. Wait for network initialization
3. The application will attempt automatic time synchronization
4. Use the interactive menu to manage certificates

Certificate Lifecycle

1. **Generate:** Use menu option 3 to create a new certificate
2. **Validate:** Use menu option 1 to check expiration status
3. **Monitor:** Use menu option 2 to view expiration dates
4. **Refresh:** Generate new certificates before expiration
5. **Clean:** Use menu option 4 to remove certificates when needed

Web Access

- Navigate to the device's IP address using HTTP or HTTPS
- Use the secure configuration pages for advanced device management
- The discovery service can help locate devices on the network

12.43.1.3.1.11 Troubleshooting

Common Issues

- **Time Synchronization Failures:** Check network connectivity and DNS resolution
- **Certificate Generation Errors:** Ensure sufficient entropy for random number generation
- **HTTPS Access Problems:** Verify certificate validity and browser security settings
- **Menu Unresponsive:** Check serial console connection and baud rate settings

Error Codes

The application provides specific error codes for certificate generation failures:

- **CERT_GEN_RETURN_INVALID_RANDOM:** Insufficient random number generation

12.43.1.3.1.12 Advanced Features

Custom Certificate Parameters

The certificate generation structure can be modified to customize:

- Subject distinguished name fields
- Validity periods
- Subject alternative names
- Key algorithms and parameters

Integration Options

- **Web Server Customization:** Modify HTTP/HTTPS server behavior
- **Certificate Storage:** Custom certificate and key storage implementations
- **Time Sources:** Alternative time synchronization methods
- **Network Interfaces:** Multi-interface support and configuration

12.43.1.4 On-board Cert Generation - Compiled Certificate Authority

12.43.1.4.1 SSL Certificate Generation Application using a Self-Signed Certificate Authority Certificate

12.43.1.4.1.1 Overview

The simple auto-generate self-signed certificate example is the most common method used for most applications. This example provides advanced users with a demonstration of how to write their own onboard certificate generation function, including signing the generated certificate with a compiled in Certificate Authority certificate. To understand how to generate Certificate Authority certificate as well as how to compile it into your program, please see our documentation on creating self-signed certificates, found in the Programmers Guide and other SSL examples.

If you generate your own Certificate Authority (CA) for any network device you must accept responsibility for the security of the CA certificate. If a malicious party obtains your CA from the device or any other means, they will be able to use it to create their own "trusted" certificate and access your devices. This may be less of an issue if you are operating on a closed/private network, rather than devices on the Internet.

Note

This example demonstrates advanced certificate generation techniques beyond the simple self-signed approach. It requires familiarity with the wolfSSL API for full customization. Refer to the NetBurner Programmer's Guide for additional details on Certificate Authority creation and compilation.

12.43.1.4.1.2 Features

- **Automatic Certificate Generation:** Creates self-signed SSL certificates with ECC keys
- **Certificate Authority Support:** Signs certificates with a compiled-in Certificate Authority
- **Interactive Menu System:** Command-line interface for certificate management
- **Time Synchronization:** NTP-based time setting with manual fallback
- **Certificate Validation:** Checks certificate validity and expiration status
- **Web Server Integration:** Supports both HTTP and HTTPS connections
- **Configurable Parameters:** Customizable certificate attributes (country, state, organization, etc.)

12.43.1.4.1.3 Application Components

Main Application (main.cpp)

The core application provides:

- Interactive menu for certificate operations
- System time configuration
- HTTP/HTTPS web server startup
- Certificate creation and validation workflows

Certificate Generation (CertGen.cpp)

Custom certificate generation implementation that:

- Creates ECC private keys with configurable key sizes
- Generates certificates signed by a compiled Certificate Authority
- Supports various ECC curves (secp256r1, secp384r1, secp521r1, etc.)
- Implements proper error handling and cleanup
- Stores certificates and keys in device flash memory

Certificate Authority (CaCrt.cpp)

Contains a compiled-in Certificate Authority certificate for signing generated certificates. This CA is embedded as a byte array in the application.

Time Utilities (TimeUtil.cpp/TimeUtil.h)

Provides comprehensive time management:

- NTP synchronization with fallback to manual entry
- Time zone configuration with automatic daylight savings
- System time display and formatting functions

12.43.1.4.1.4 Security Considerations

Important**: When using a custom Certificate Authority, you must secure the CA certificate. If compromised, an attacker could create trusted certificates to access your devices. This risk is reduced in closed/private networks compared to Internet-facing deployments.

12.43.1.4.1.5 Interactive Menu Options

1. **Check Certificate Expiration** - Verify if the current certificate is expired
2. **Show Certificate Expiration** - Display the certificate expiration date
3. **Generate New Certificate** - Create a new certificate/key pair
4. **Erase Certificate** - Remove existing certificates (requires reboot)
5. **Display System Time** - Show current system time and date
6. **Reboot** - Restart the device

12.43.1.4.1.6 Certificate Configuration

Default certificate attributes can be customized in the `GetDataForCertGen()` function:

- **Country:** US
- **State:** CA (California)
- **Locality:** San Diego
- **Organization:** NetBurner
- **Unit:** CodeDemo
- **Email:** `Sales@NetBurner.com`
- **Common Name:** MyNetburner
- **Validity Period:** 1 year
- **Alternative Names:** Device IP address (both as IP and DNS entry)

12.43.1.4.1.7 Key Features

ECC Key Generation

- Supports multiple ECC curves based on `DEFAULT_KEY_TYPE`
- Key sizes: 24-66 bytes depending on curve selection
- Hardware random number generation on supported platforms
- Secure key storage in device flash memory

Certificate Signing

- Uses compiled Certificate Authority for signing
- SHA256 with RSA signature algorithm
- Proper ASN.1 DER encoding
- Alternative names support for IP and DNS entries

Time Management

- Automatic NTP synchronization from pool.ntp.org
- Manual time entry fallback if NTP fails
- Time zone support with daylight savings calculation
- Minimum time validation for certificate generation

12.43.1.4.1.8 Web Server Access

Once running, the device can be accessed via:

- HTTP: `http://[device-ip]`
- HTTPS: `https://[device-ip]`
- Configuration: `http://[device-ip]:20034` or `https://[device-ip]:20034`
- Discovery service: discover.netburner.com (if Internet access available)

12.43.1.4.1.9 Build Requirements

- NetBurner NNDK (NetBurner Network Development Kit)
- wolfSSL cryptographic library
- Compatible NetBurner hardware platform (MODM7AE70 recommended for hardware RNG)

12.43.1.4.1.10 Usage Notes

1. **Random Number Generation:** The application waits for valid entropy before generating keys. Network traffic helps build entropy on Coldfire platforms.
2. **System Time:** Accurate time is required for certificate generation and validation. The application attempts NTP synchronization first, then falls back to manual entry.
3. **Memory Management:** The application uses dynamic memory allocation for cryptographic operations and implements proper cleanup on errors.
4. **Certificate Storage:** Generated certificates and keys are stored in device flash memory and persist across reboots.

12.43.1.4.1.11 Error Handling

The application includes comprehensive error handling for:

- Memory allocation failures
- Cryptographic operation errors
- Certificate generation and signing failures
- File system operations
- Network connectivity issues

12.43.1.5 ACME Certificate Generation (Easy Servlet)

```
<NNDK install>\examples\SSL\acmeservlet
```

12.43.1.5.1 ACME Certificate Generation (Easy Servlet)

A NetBurner NNDK SSL/TLS example application that demonstrates automatic ACME certificate generation using Let's Encrypt or other ACME-compliant certificate authorities.

12.43.1.5.1.1 Overview

This application provides a web-based demonstration of automatic SSL/TLS certificate generation using the ACME (Automatic Certificate Management Environment) protocol as defined in RFC 8555. The application includes a built-in web server that serves both HTTP and HTTPS content, with automatic certificate provisioning through ACME providers.

SSL/TLS example program that demonstrates getting an ACME cert automatically. It will only work for a device with Internet access with a public IP address and DNS entry. Make sure to change the `DEFAULT_CERT_COM_NAME` constant in `main.cpp` to match.

This example makes use of the [acmeRFC8555Servlet.h](#) builtin to the NNDK which reduces how much code is required to add the functionality to your project and will more easily take advantage of future NNDK updates.

12.43.1.5.1.2 Requirements

- NetBurner device with public IP address
- Public DNS entry pointing to the device
- NNDK (NetBurner Network Development Kit)
- Device must be accessible from the internet for ACME validation

12.43.1.5.1.3 Features

- Automatic SSL/TLS certificate generation via ACME protocol
- Support for multiple ACME providers (Let's Encrypt, BuyPass, custom)
- Web server with both HTTP (port 80) and HTTPS (port 443) support
- Access control based on SSL/TLS connection status
- Certificate reissue functionality
- NTP client integration for accurate time synchronization
- Web-based status and control interface

12.43.1.5.1.4 Configuration

Certificate Information

Before running the application, update the following constants in `main.cpp` to match your environment:

```
const char * DEFAULT_CERT_COUNTRY= "US";
const char * DEFAULT_CERT_STATE= "CA";
const char * DEFAULT_CERT_LOCALITY= "San Diego";
const char * DEFAULT_CERT_ORG= "Netburner";
const char * DEFAULT_CERT_UNIT= "Demo";
const char * DEFAULT_CERT_EMAIL= "changeme@example.com";
const char * DEFAULT_CERT_ALT_NAMES= "test.example.com,example.com";
const char * DEFAULT_CERT_COM_NAME= "test.example.com";
```

Important*: The `DEFAULT_CERT_COM_NAME` must match your device's public DNS name.

ACME Provider Selection

The application supports multiple ACME providers. Choose one by uncommenting the appropriate line:

```
/* Production Let's Encrypt (default): */
LetsEncryptAcmeServletObject AcmeClient;

/* Let's Encrypt Staging (for testing): */
// LetsEncryptAcmeServletObject AcmeClient(true);

/* BuyPass ACME: */
// BuyPassAcmeServletObject AcmeClient;

/* Custom ACME server: */
// AcmeServletObject AcmeClient("https://your-acme-server.com/dir");
```

12.43.1.5.1.5 Building and Running

1. Ensure your NetBurner device has a public IP and DNS entry
2. Update the certificate configuration constants
3. Build the application using the NNDK toolchain
4. Deploy to your NetBurner device
5. Access the web interface via HTTP or HTTPS

12.43.1.5.1.6 Web Interface

The application provides several web endpoints:

- `/index.html` - Main interface page
- Counter functionality - Displays page reload count
- SSL status indicators - Shows connection security status
- Certificate management controls

Access Control

The application implements access control based on connection type:

- Access level 0: Available to all connections
- Access level 1: Requires SSL/TLS connection
- Other levels: Default allow access

12.43.1.5.1.7 Runtime Options

When the application starts, you can:

- Press 'R', 'r', 'Y', or 'y' to reissue certificates
- Monitor certificate status through diagnostic output
- View ACME client state information

12.43.1.5.1.8 ACME Certificate Process

1. Application generates a Certificate Signing Request (CSR)
2. Contacts the ACME server to request a certificate
3. Completes domain validation challenges
4. Receives and installs the signed certificate
5. Automatically renews certificates before expiration

12.43.1.5.1.9 Key Components

- **ACME Client:** Handles certificate requests and renewals
- **Web Server:** Serves HTTP/HTTPS content
- **NTP Client:** Maintains accurate time for certificate validation
- **SSL Context:** Manages SSL/TLS connections and certificates
- **Access Control:** Enforces connection security requirements

12.43.1.5.1.10 Troubleshooting

Common Issues

1. **Certificate generation fails:** Verify public DNS and IP accessibility
2. **ACME validation fails:** Check firewall settings and port accessibility
3. **Time sync issues:** Ensure NTP client can reach time servers
4. **Domain mismatch:** Verify `DEFAULT_CERT_COM_NAME` matches DNS

Debug Options

Enable diagnostic output:

```
AcmeClient.SetDiag(true);
```

Force certificate reissue:

```
AcmeClient.Delete_Everything_Restart();
```

12.43.1.5.1.11 Security Notes

- Certificates are automatically managed by the ACME client
- Production deployment should disable system diagnostics
- Consider implementing additional access controls for production use
- Monitor certificate expiration and renewal processes

12.43.1.5.1.12 License

This example code is provided as part of the NetBurner NNDK and follows the NetBurner licensing terms.

12.43.1.5.1.13 Support

For technical support and documentation:

- NetBurner NNDK Documentation
- NetBurner Support Portal
- ACME RFC 8555 specification

12.43.2 FTPS Server

12.43.2.1 FTPS Server Example

A simple FTPS (FTP over SSL/TLS) server implementation for NetBurner embedded devices that demonstrates secure file transfer capabilities with SSL encryption.

12.43.2.1.1 Overview

This example creates a basic FTPS server that runs on a NetBurner device and listens for secure client connections on port 990. The server implements the core functionality needed for encrypted file transfers while maintaining simplicity for educational and development purposes.

Key Features:**

- SSL/TLS encrypted FTP server (FTPS)
- Self-signed certificate authentication
- File download capability
- File upload with content display
- Real-time connection monitoring
- Configurable data channel encryption

12.43.2.1.2 Architecture

The application consists of several key components:

12.43.2.1.2.1 Core Files

- `main.cpp` - Main server implementation and callback functions
- `ServerCert.cpp` - Embedded self-signed SSL certificate
- `ServerKey.cpp` - Embedded SSL private key

12.43.2.1.2.2 SSL Configuration

- **Command Channel:** Always encrypted when FTPS is enabled
- **Data Channel:** Optionally encrypted (configurable)
- **Certificate:** Built-in self-signed certificate for testing
- **Port:** 990 (standard FTPS implicit SSL port)

12.43.2.1.3 Functionality

12.43.2.1.3.1 File Operations

Download (Client -> Server)

- **Available File:** `ReadFile.txt`
- **Content:** Hardcoded text string demonstrating file serving
- **Purpose:** Shows how to serve files to FTPS clients

Upload (Server -> Client)

- **Accepted File:** `WriteFile.txt` only
- **Processing:** Content is displayed on the serial debug port
- **Storage:** Files are not permanently stored (memory only)
- **Purpose:** Demonstrates file reception and processing

12.43.2.1.3.2 Authentication

- Simple authentication mechanism (currently accepts any credentials)
- Extensible design for custom authentication logic
- Session management with start/end callbacks

12.43.2.1.3.3 Directory Operations

- Basic file listing support
- Single-level directory structure
- No subdirectory creation/deletion (returns `FTPD_FAIL`)
- No file deletion or renaming capabilities

12.43.2.1.4 Configuration Options

12.43.2.1.4.1 Network Settings

```
const int listenPort = 990;           // FTPS port (21 for plain FTP)
const bool enableFTPS = true;        // Enable SSL encryption
const bool enableEncryptedDataChannel = false; // Data channel encryption
```

12.43.2.1.4.2 Security Notes

- Uses implicit SSL (connection encrypted from start)
- Self-signed certificate included for testing
- Data channel encryption disabled by default for compatibility
- Authentication currently accepts any credentials

12.43.2.1.5 Usage Instructions

12.43.2.1.5.1 Setup Requirements

1. NetBurner device with network connectivity
2. FTPS client software (WinSCP, FileZilla with FTPS support)
3. Serial terminal software (MTTTY) for monitoring

12.43.2.1.5.2 Running the Server

1. Flash the application to your NetBurner device
2. Connect serial terminal to debug port
3. Wait for network initialization (5 seconds timeout)
4. Server will start and display status messages

12.43.2.1.5.3 Connecting with FTPS Client

1. Configure client for FTPS (not SFTP)
2. Connect to NetBurner device IP on port 990
3. Use any username/password (authentication currently accepts all)
4. Accept the self-signed certificate when prompted

12.43.2.1.5.4 File Operations

- **Download:** Select `ReadFile.txt` to download sample content
- **Upload:** Send a text file named `WriteFile.txt` to see content on serial port

12.43.2.1.6 Development Notes

12.43.2.1.6.1 Extending the Example

This basic implementation can be enhanced by:

- Adding proper file system support (EFFS-FAT or EFFS-STD)
- Implementing robust authentication
- Adding directory navigation
- Supporting file deletion and renaming
- Using production certificates

12.43.2.1.6.2 Callback Functions

The server implements these FTPD callback functions:

- `FTPD_ListFile()` - File listing
- `FTPD_SendFileToClient()` - File downloads
- `FTPD_GetFileFromClient()` - File uploads
- `FTPD_FileExists()` - File existence checks
- `FTPD_AbleToCreateFile()` - Upload permissions
- `FTPDSessionStart()` / `FTPDSessionEnd()` - Session management

12.43.2.1.6.3 Monitoring and Debugging

- Serial port displays server status and connection information
- SSL socket status updated every 5 seconds
- File upload content shown in real-time
- Error messages and diagnostic information available

12.43.2.1.7 Security Considerations

12.43.2.1.7.1 Production Deployment

For production use, consider:

- Replace self-signed certificate with CA-signed certificate
- Implement proper user authentication
- Enable data channel encryption if required
- Add input validation and error handling
- Implement proper file system permissions

12.43.2.1.7.2 Current Limitations

- Self-signed certificate (browser warnings)
- Basic authentication (accepts any credentials)
- No persistent file storage
- Limited file operations
- Single file download/upload support

12.43.2.1.8 Compatibility

12.43.2.1.8.1 FTPS Clients

Tested with:

- WinSCP (recommended)
- FileZilla (FTPS mode)
- Other RFC 4217 compliant FTPS clients

12.43.2.1.8.2 NetBurner Platform

- Requires NNDK (NetBurner Network Development Kit)
- SSL/TLS library support
- Network stack initialization
- Serial debugging capabilities

12.43.2.1.9 Troubleshooting

12.43.2.1.9.1 Common Issues

- **Connection refused:** Check network connectivity and port configuration
- **Certificate errors:** Accept self-signed certificate in client
- **Authentication failed:** Verify FTPS (not SFTP) mode in client
- **File not found:** Ensure exact filename match (`ReadFile.txt`)
- **Upload rejected:** Use exact filename `WriteFile.txt`

12.43.2.1.9.2 Debug Information

Monitor serial output for:

- Server startup status
- SSL socket connections
- File transfer progress
- Error messages and diagnostics

12.43.3 HTML File Post

12.43.3.1 SSL/TLS File Post with Multipart Form

12.43.3.1.1 Overview

This NetBurner application demonstrates secure file uploading capabilities using SSL/TLS encryption. It provides a web interface that allows users to select and upload files to the NetBurner device through an HTTPS connection. The uploaded files are then processed and displayed back to the user on a web page.

Note

Browsers require a valid device certificate signed by a Certificate Authority known to the browser. Unless you install your self-signed Certificate Authority certificate in the web browser, you will see a warning message in the browser and should select the option to "continue anyway".

12.43.3.1.2 Features

- **SSL/TLS Security:** Secure file transfers using HTTPS protocol
- **Multipart Form Support:** Handles file uploads through HTML multipart forms
- **File Display:** Shows uploaded file content in both text and binary formats
- **Web Interface:** Simple HTML form for file selection and upload
- **Real-time Processing:** Immediate processing and display of uploaded files

12.43.3.1.3 Application Components

12.43.3.1.3.1 Core Files

- **main.cpp:** Main application entry point that initializes the network stack, enables HTTPS server, and manages the main application loop
- **formcode.cpp:** Contains the file processing logic and HTTP POST callback handlers
- **index.html:** Web interface with file upload form
- **ServerCert.cpp:** SSL certificate data for HTTPS connections
- **ServerKey.cpp:** SSL private key data for HTTPS connections

12.43.3.1.3.2 Key Functions

File Processing (`formcode.cpp`)

- `ProcessPostFile()`: Reads uploaded file data into a buffer
- `ShowFileData()`: Displays file content with optional binary format
- `PostCallBack()`: Handles HTTP POST events during file upload process

Main Application (`main.cpp`)

- `UserMain()`: Initializes network, starts HTTPS server, and waits for network connectivity

12.43.3.1.4 How It Works

1. **Initialization:** The application starts by initializing the network stack and enabling system diagnostics
2. **HTTPS Server:** Starts the HTTPS web server with SSL/TLS encryption
3. **Network Wait:** Waits up to 5 seconds for an active network connection ([DHCP](#))
4. **File Upload Process:**
 - User accesses the web interface through HTTPS
 - Selects a file using the HTML form
 - Clicks "Send File" to initiate upload
 - Server processes the multipart form data
 - File content is read and stored in a buffer
 - Processed file is displayed back to the user

12.43.3.1.5 Security Features

12.43.3.1.5.1 SSL/TLS Implementation

- Uses embedded SSL certificate and private key
- Provides encrypted communication between browser and device
- Protects file uploads from interception

12.43.3.1.5.2 Browser Requirements

Modern browsers require valid certificates for file uploads. This application includes:

- Self-signed certificate embedded in the firmware
- Certificate Authority (CA) certificate may need installation in browser
- Warning messages in browsers must be handled properly for file uploads to work

12.43.3.1.6 Technical Specifications

12.43.3.1.6.1 Buffer Management

- **File Buffer Size:** 10,000 bytes maximum
- **Read Buffer:** 16-byte chunks for binary display
- **Memory Safety:** Prevents buffer overflows during file processing

12.43.3.1.6.2 File Display Formats

- **Text Mode:** Shows readable characters with HTML entity encoding
- **Binary Mode:** Displays hexadecimal values alongside ASCII representation
- **HTML Safe:** Converts special characters (&, <, >, ") to HTML entities

12.43.3.1.7 Usage Instructions

1. **Deploy the Application:** Flash the firmware to your NetBurner device
2. **Network Connection:** Ensure the device has network connectivity
3. **Access Web Interface:** Navigate to the device's IP address using HTTPS
4. **Upload Files:** Use the file selection form to choose and upload files
5. **View Results:** The uploaded file content will be displayed on the result page

12.43.3.1.8 Testing

A quick test can be performed by uploading the included `ReadMe.txt` file to verify the application is working correctly.

12.43.3.1.9 Important Notes

12.43.3.1.9.1 Certificate Warnings

- Browsers may display security warnings for self-signed certificates
- For form variables, users can typically choose to continue despite warnings
- For file uploads, proper certificate handling is more critical
- Consider installing a valid CA certificate in browsers for production use

12.43.3.1.9.2 File Size Limitations

- Maximum file size is limited to 10,000 bytes
- Larger files will be truncated to fit within the buffer
- Consider increasing `FILE_BUFFER_SIZE` for larger file support

12.43.3.1.9.3 Production Considerations

- Remove `EnableSystemDiagnostics()` for production deployments
- Consider implementing additional file type validation
- Add error handling for network connectivity issues
- Implement proper certificate management for production environments

12.43.3.1.10 Development Environment

This application is built for the NetBurner platform using the NetBurner Network Development Kit (NNDK). The code utilizes NetBurner-specific libraries for HTTP handling, SSL/TLS support, and network operations.

12.43.3.1.11 Related Examples

This example builds upon the standard `examples\web\HtmlFilePost` example by adding SSL/TLS encryption capabilities for secure file transfers.

12.43.4 HTML Form Post

12.43.4.1 NetBurner SSL HTML Form Post Example

12.43.4.1.1 Overview

This application demonstrates how to implement secure HTML form processing using SSL/TLS on NetBurner embedded devices. It extends the basic HTML form post functionality by adding SSL encryption for secure web communication.

12.43.4.1.2 Application Description

The **HTML Form Post Example** is a web-based application that runs on NetBurner devices and provides:

- SSL/TLS encrypted web server functionality
- Multiple HTML forms with POST data processing
- Server-side form validation and data handling
- Page redirection after form submission
- Embedded SSL certificate and private key management

12.43.4.1.3 Key Features

12.43.4.1.3.1 Security

- **SSL/TLS Support:** Uses HTTPS for encrypted communication
- **Embedded Certificates:** Self-contained SSL certificate and private key
- **Secure Form Processing:** All form data transmitted over encrypted connections

12.43.4.1.3.2 Web Interface

- **Multi-page Workflow:** Three-page form processing sequence
- **Two Form Types:** Different forms demonstrating various input handling
- **Dynamic Content:** Server-side variable substitution in HTML pages
- **Error Handling:** Form validation with error reporting

12.43.4.1.3.3 Form Processing

- **Form 1:** Single text input field with simple processing
- **Form 2:** Multiple variable input fields (Var0-Var3)
- **Callback System:** Event-driven form processing with multiple callback points
- **Data Persistence:** Form data maintained between page transitions

12.43.4.1.4 File Structure

```
\src
  main.cpp           # Main application entry point
  web.cpp            # Web server and form processing logic
  ServerCert.cpp     # Embedded SSL certificate data
  ServerKey.cpp      # Embedded SSL private key data

\html
  index.html         # First form page
  page2.html         # Second form page
  complete.html      # Completion/results page
```

12.43.4.1.5 Application Flow

1. Initialization

- Network stack initialization
- HTTPS server startup (SSL-enabled)
- **DHCP** network configuration
- System diagnostics enablement

2. Form Processing Workflow

`index.html > form1 POST > page2.html > form2 POST > complete.html`

3. Form Callbacks

- `eStartingPost`: Initialize form processing
- `eVariable`: Process each form variable
- `eFile`: Handle file uploads (if present)
- `eEndOfPost`: Complete processing and redirect

12.43.4.1.6 Technical Implementation

12.43.4.1.6.1 SSL Configuration

- Uses embedded certificate and private key data
- Certificate validity: 2019-2029
- Elliptic Curve (EC) cryptography
- Self-signed certificate for development/testing

12.43.4.1.6.2 Form Handlers

- **postForm1**: Handles single text input from `index.html`
- **postForm2**: Processes multiple variables from `page2.html`
- **URL Patterns**: "form1*" and "form2*" for flexible routing

12.43.4.1.6.3 Data Storage

- **Form 1**: Single 80-character text buffer
- **Form 2**: Array of 4 variables, 80 characters each
- **Error Tracking**: Boolean flags for form validation

12.43.4.1.6.4 HTML Integration

- **Dynamic Content**: `<!--CPPCALL FunctionName -->` for server-side includes
- **WebTextForm1()**: Displays Form 1 data
- **WebTextForm2()**: Displays Form 2 data with formatting

12.43.4.1.7 Network Configuration

- **Protocol**: HTTPS (SSL/TLS encrypted HTTP)
- **Port**: Default HTTPS port (443)
- **Network**: **DHCP** client configuration
- **Timeout**: 5-second network initialization wait

12.43.4.1.8 Development Notes

12.43.4.1.8.1 Security Considerations

- Uses development/test SSL certificate
- Replace embedded certificate for production use
- Certificate includes localhost and IP address subjects
- Self-signed certificate will trigger browser warnings

12.43.4.1.8.2 Memory Usage

- Fixed-size buffers for form data
- Embedded certificate data: 672 bytes
- Embedded private key data: 310 bytes
- Form variables: configurable buffer sizes

12.43.4.1.8.3 Customization Points

- Form variable names and counts
- Buffer sizes for form data
- SSL certificate replacement
- HTML page content and styling

12.43.4.1.9 Building and Deployment

1. Compile with NetBurner NNDK (NetBurner Network Development Kit)
2. Flash to NetBurner device
3. Configure network settings ([DHCP](#) or static IP)
4. Access via HTTPS in web browser
5. Accept security warning for self-signed certificate

12.43.4.1.10 Browser Compatibility

- Supports all modern web browsers
- Requires JavaScript disabled for basic functionality
- HTML form POST method compatibility
- SSL/TLS certificate acceptance required

12.43.4.1.11 Troubleshooting

12.43.4.1.11.1 Common Issues

- **Certificate Warnings:** Expected with self-signed certificates
- **Network Connectivity:** Verify [DHCP](#) configuration
- **Form Submission:** Check POST URL patterns and handlers
- **SSL Handshake:** Ensure certificate validity and browser compatibility

12.43.4.1.11.2 Debug Output

- Console logging for all form processing events
- Variable value tracking
- Error condition reporting
- Network initialization status

12.43.4.1.12 API Reference

12.43.4.1.12.1 Key Functions

- `init()`: Initialize network stack
- `StartHttps()`: Start SSL-enabled web server
- `WaitForActiveNetwork()`: Wait for network connectivity
- `RedirectResponse()`: Send HTTP redirect response
- `fdprintf()`: Formatted output to socket

12.43.4.1.12.2 Callback Events

- `eStartingPost`: Form processing initialization
- `eVariable`: Individual variable processing
- `eFile`: File upload handling
- `eEndOfPost`: Form completion processing

12.43.5 HTTPS Dual Cert

12.43.5.1 NetBurner HTTPS Dual Certificate Server

12.43.5.1.1 Overview

This NetBurner application demonstrates a dual-certificate HTTPS/HTTP server implementation that supports both compiled-in certificates and dynamically loaded certificates from external storage. The application provides a web interface accessible via both secure (HTTPS) and non-secure (HTTP) connections, along with FTP server functionality for remote file management.

12.43.5.1.2 Supported Platforms

- MODM7AE70
- SBE70LC
- MOD5441X
- NANO54415
- SOMRT1061
- MODRT1171

12.43.5.1.3 Key Features

12.43.5.1.3.1 Dual Certificate Support

- **Compiled-in Certificate:** Server includes a built-in certificate and private key for immediate SSL/TLS functionality
- **External Certificate Loading:** Automatically detects and uses certificate files (`DEVICE.CRT`) and private key files (`DEVICE.KEY`) from external flash storage when available
- **Fallback Mechanism:** Gracefully falls back to compiled-in certificates if external files are not found or invalid

12.43.5.1.3.2 Network Services

- **HTTPS Server:** Secure web server running on port 443
- **HTTP Server:** Non-secure web server running on port 80
- **FTP Server:** File transfer protocol server on port 21 for remote file management

12.43.5.1.3.3 External Storage Support

- **Multiple Flash Card Types:** Supports SD/SDHC, MMC, and Compact Flash cards
- **FAT32 File System:** Full read/write access to external storage using EFFS FAT library
- **Hot-swappable Storage:** Runtime detection of card insertion/removal with write protection checks

12.43.5.1.3.4 Time Management

- **NTP Client:** Automatic time synchronization from network time servers
- **Manual Time Setting:** Fallback manual time configuration
- **Timezone Support:** Configurable timezone settings (defaults to PST/PDT)
- **RTC Integration:** Support for hardware real-time clock when available

12.43.5.1.3.5 Web Interface Features

- **Dynamic Content:** Web pages with embedded function calls for dynamic content generation
- **SSL/HTTP Switching:** Links to switch between secure and non-secure versions of pages
- **Auto-refresh Pages:** Demonstration pages with automatic reload functionality
- **Flash Card Formatting:** Web-based flash card formatting utility

12.43.5.1.4 File Structure

12.43.5.1.4.1 Core Application Files

- `main.cpp` - Main application entry point and initialization
- `cardtype.h` - Flash card type configuration and selection
- `dual_mode_file_key.cpp` - Dual certificate management implementation

12.43.5.1.4.2 File System Utilities

- `FileSystemUtils.cpp/.h` - EFFS FAT file system wrapper functions
- `effs_time.cpp/.h` - Time management and NTP client implementation

12.43.5.1.4.3 Network Services

- `ftp_f.cpp/.h` - FTP server implementation for EDFS file systems
- `http_f.h` - HTTP server function registration

12.43.5.1.4.4 Certificate Management

- `ServerCert.cpp` - Compiled-in server certificate (PEM format)
- `ServerKey.cpp` - Compiled-in private key (PEM format)

12.43.5.1.4.5 Web Content

- `index.html` - Main web page with SSL/HTTP switching links
- `format.html` - Flash card formatting utility page
- `repeat.html` - Auto-refreshing demonstration page
- `robots.txt` - Web crawler control file

12.43.5.1.4.6 Platform-Specific Files

- `ramdrv_mcf.cpp` - RAM drive configuration for testing

12.43.5.1.5 Configuration Options

12.43.5.1.5.1 Flash Card Types

The application supports multiple flash card types configured in `cardtype.h`:

- **USE_SDHC**: SD/SDHC cards (default for SB800EX and SOMRT1061)
- **USE_MMC**: SD/MMC cards
- **USE_CFC**: Compact Flash cards
- **USE_RAM**: RAM-based file system for testing

12.43.5.1.5.2 Certificate Loading Priority

1. External certificates from flash card (`DEVICE.CRT` and `DEVICE.KEY`)
2. Compiled-in certificates (if external files not found or invalid)

12.43.5.1.5.3 Network Configuration

- HTTPS Port: 443
- HTTP Port: 80
- FTP Port: 21

12.43.5.1.6 Usage Instructions

12.43.5.1.6.1 Initial Setup

1. Configure flash card type in `cardtype.h` based on your hardware
2. Insert properly formatted flash card (FAT32 recommended)
3. Optionally copy certificate files (`DEVICE.CRT`, `DEVICE.KEY`) to flash card root directory
4. Compile and deploy application to NetBurner device

12.43.5.1.6.2 Web Interface Access

- **HTTP:** `http://[device-ip]/index.html`
- **HTTPS:** `https://[device-ip]/index.html`
- **Format Flash:** `http://[device-ip]/format.html`

12.43.5.1.6.3 FTP Access

- Connect to `ftp://[device-ip]` for file management
- Upload/download certificate files
- Manage web content and application files

12.43.5.1.6.4 Serial Console Commands

- Press 'F' to format the flash card
- Monitor application status and debug information

12.43.5.1.7 Technical Implementation Details

12.43.5.1.7.1 Certificate Validation

The application validates external certificates using the NetBurner SSL library function `IsSSL_CertNKeyValid()` to ensure compatibility before use.

12.43.5.1.7.2 Memory Management

- External certificate files are loaded into dynamically allocated memory
- Automatic cleanup of memory buffers after certificate processing
- Graceful handling of memory allocation failures

12.43.5.1.7.3 Error Handling

- Comprehensive EDFS error code reporting with descriptive messages
- Network retry mechanisms for robust file transfers
- Graceful fallback to compiled-in certificates on external certificate failure

12.43.5.1.7.4 Security Considerations

- Example certificates included are for demonstration only
- Production deployments should use properly signed certificates
- Web browsers will display warnings for self-signed certificates

12.43.5.1.8 Build Requirements

12.43.5.1.8.1 Required Libraries

- **FatFile.a:** EDFS FAT file system library
- **SSL/TLS Libraries:** NetBurner cryptographic libraries
- **Network Libraries:** HTTP/HTTPS and FTP server libraries

12.43.5.1.8.2 NBEclipse Configuration

To include the FatFile library in NBEclipse:

1. Right-click project and select "Properties"
2. Navigate to "C/C++ Build -> Settings"
3. Select "GNU C/C++ Linker -> Libraries"
4. Add "FatFile" to the libraries list

12.43.5.1.9 Development Notes

This application serves as a comprehensive example of:

- SSL/TLS certificate management in embedded systems
- External storage integration with embedded web servers
- Network service coordination (HTTP/HTTPS/FTP)
- Time synchronization in embedded applications
- Robust error handling and fallback mechanisms

12.43.6 HTTPS Upload Cert

12.43.6.1 NetBurner SSL Certificate Upload Application

12.43.6.1.1 Overview

This NetBurner embedded application provides a comprehensive SSL/TLS certificate management system with web-based certificate upload functionality. The application enables users to replace default compiled-in certificates with their own SSL/TLS certificates and private keys through a secure web interface, supporting modern HTTPS connectivity for embedded devices.

12.43.6.1.2 Key Features

12.43.6.1.2.1 Certificate Management

- **Web-based Upload Interface:** User-friendly web forms for uploading SSL/TLS certificates and private keys
- **Multiple Certificate Sources:** Support for library default, compiled-in default, and user-installed certificates
- **Certificate Validation:** Automatic validation of certificate and private key pairs before installation
- **PEM Format Support:** Handles Privacy-Enhanced Mail (PEM) encoded certificates and keys
- **Certificate Authority Management:** Support for multiple CA certificates for client verification

12.43.6.1.2.2 File System Integration

- **EFFS-STD File System:** Uses NetBurner's Enhanced Flash File System Standard for persistent storage
- **On-chip Flash Storage:** Certificates stored in protected flash memory
- **Atomic Updates:** Safe certificate installation with automatic rollback on failure
- **File Size Limits:** 3KB maximum for certificates, 4KB maximum for private keys

12.43.6.1.2.3 Network and Security

- **Dual Stack Support:** Full IPv4 and IPv6 HTTPS connectivity
- **NetBIOS Integration:** Automatic device name publishing
- **NTP Time Synchronization:** Required for proper certificate validation
- **Debug Console:** Serial interface for system monitoring and configuration

12.43.6.1.3 Supported Platforms

- **MOD5441X**: 32MB parallel flash with 128KB sectors
- **NANO54415**: 8MB SPI flash with 4KB sectors
- **SB800EX**: 8MB SPI flash with 4KB sectors
- **MODM7AE70**: 2MB parallel flash with 8KB sectors
- **SBE70LC**: 2MB parallel flash with 8KB sectors

12.43.6.1.4 Application Architecture

12.43.6.1.4.1 Core Components

Main Application (`main.cpp`)

The main application handles system initialization and provides the primary user interface:

- **System Initialization**: Network configuration, SSL service startup, and file system mounting
- **NTP Time Sync**: Automatic time synchronization using configurable timezone (default PST/PDT)
- **Debug Interface**: Serial console with commands for system management
- **NetBIOS Configuration**: Automatic device name publishing based on MAC address
- **Certificate Scanning**: Automatic CA certificate discovery and validation

Certificate Management (`certificatekey.cpp`)

Handles all certificate-related operations:

- Certificate and key file upload processing
- Validation of certificate/key pairs
- CA certificate management (up to 20 certificates)
- Web interface functions for certificate display and status

SSL User Interface (`ssluser.cpp`)

Provides certificate retrieval and management functions:

- Default certificate management
- PEM format handling and conversion
- Certificate source switching between library default, compiled default, and user-installed

File System (`effsStdFlashDrv.cpp`, `FileSystemUtils.cpp`)

Platform-specific flash drivers and utilities:

- Hardware abstraction for different flash types (parallel vs SPI)
- File system operations (read, write, delete, format)
- Error handling and reporting

12.43.6.1.5 File System Structure

```
/
cert.crt           # Active SSL certificate (max 3KB)
cert.key           # Corresponding private key (max 4KB)
CACert0.crt        # Certificate Authority cert #0
CACert1.crt        # Certificate Authority cert #1
...                # Additional CA certs (up to CACert19.crt)
```

12.43.6.1.6 Web Interface

The main configuration page ([https://...html](https://...)) provides:

- **Certificate Status Display:** Shows current certificate source (Default vs User Installed)
- **Upload Forms:** Separate forms for certificate and key file uploads
- **Certificate Information:** View current certificate details and validity
- **Factory Reset:** Option to restore default certificates
- **Network Information:** Display of HTTP and HTTPS access addresses

12.43.6.1.7 Debug Console Commands

Available via serial interface:

- 0 - Turn off debug messages
- 1 - Turn on debug messages
- D - Reset to factory default parameters
- F - Show file system information and statistics
- G - Format file system (WARNING: Erases all data)
- N - Display current NV settings
- T - Display current system time
- B - Force system reboot

12.43.6.1.8 Certificate Requirements

12.43.6.1.8.1 Supported Formats

- **Encoding:** PEM (Privacy-Enhanced Mail) format only
- **Key Types:** RSA, DSA, and ECDSA keys supported
- **Certificate Chain:** Multiple CA certificates supported for client verification

12.43.6.1.8.2 Size Limitations

- **Certificate Files:** Maximum 3KB in PEM format
- **Private Key Files:** Maximum 4KB in PEM format
- **CA Certificates:** Maximum 3KB each, up to 20 total

12.43.6.1.9 Configuration

12.43.6.1.9.1 Network Settings

- **Device Naming:** Automatic device name based on MAC address
- **DHCP Support:** Automatic IP address assignment
- **Dual Stack:** IPv4/IPv6 support for maximum compatibility

12.43.6.1.9.2 SSL/TLS Settings

- **Certificate Sources:** Configurable between library default, compiled default, and user-installed
- **Client Verification:** Optional client certificate verification using CA certificates
- **Peer Verification:** Configurable peer certificate validation

12.43.6.1.10 Security Features

12.43.6.1.10.1 Certificate Validation

- Automatic verification of certificate and private key pairs
- Certificate format and structure validation
- Certificate chain verification for CA certificates
- Expiration date checking (requires accurate system time)

12.43.6.1.10.2 Secure Storage

- Protected flash memory storage for certificates and keys
- Atomic update operations prevent corruption during uploads
- Automatic rollback to default certificates on validation failure
- Memory protection and cleanup for sensitive data

12.43.6.1.11 Getting Started

12.43.6.1.11.1 Initial Setup

1. **Build and Deploy:** Compile for your target NetBurner platform
2. **Network Connection:** Connect device to network ([DHCP](#) recommended)
3. **Initial Access:** Browse to device IP address via HTTP
4. **Time Synchronization:** Verify NTP time sync (required for certificate validation)

12.43.6.1.11.2 Certificate Installation

1. **Access Web Interface:** Navigate to `https.html` page
2. **Upload Certificate:** Use the certificate upload form (PEM format, max 3KB)
3. **Upload Private Key:** Use the key upload form (PEM format, max 4KB)
4. **Verification:** System automatically validates certificate/key pair
5. **HTTPS Access:** Test secure connectivity using HTTPS

12.43.6.1.11.3 CA Certificate Management

1. **Upload CA Certificates:** Use CA certificate upload form
2. **Client Verification:** Enable client certificate verification if needed
3. **Multiple CAs:** Upload additional CA certificates as required (max 20)

12.43.6.1.12 Troubleshooting

12.43.6.1.12.1 Common Issues

Certificate Upload Problems

- **File Size Errors:** Verify certificate < 3KB and key < 4KB in PEM format
- **Certificate Validation Failures:** Ensure certificate and key are properly paired and in valid PEM format
- **Upload Timeouts:** Large files may timeout; check network connectivity and file size limits
- **Invalid File Format:** Only PEM-encoded certificates and keys are supported

Time Synchronization Issues

- **NTP Failures:** Check network connectivity, DNS resolution, and firewall settings
- **Certificate Validation Errors:** Accurate system time required for certificate expiry checking
- **Timezone Problems:** Use `tzsetchar()` for proper timezone configuration

Browser Connection Issues

- **Self-Signed Certificate Warnings:** Install CA certificate in browser or use proper CA-signed certificates
- **HTTPS Connection Failures:** Verify certificate installation and SSL service startup
- **IPv6 Connectivity:** Ensure proper IPv6 configuration if using IPv6 addresses

File System Problems

- **Flash Corruption:** May require file system format and factory reset
- **Insufficient Space:** Check available flash memory before certificate upload
- **Mount Failures:** Verify flash hardware and driver configuration

12.43.6.1.12.2 Recovery Options

Factory Reset Procedures

- **Debug Command:** Use serial command 'D' to reset to factory defaults
- **Web Interface:** Use factory reset option on configuration page
- **NV Settings Reset:** Clears user certificates and restores default configuration
- **File System Impact:** May require file system format depending on corruption level

File System Recovery

- **Format Command:** Use debug command 'G' to format file system (WARNING: Erases all data)
- **Automatic Recovery:** System detects corruption and attempts automatic recovery
- **Manual Recovery:** Delete specific certificate files via FTP or debug interface

Certificate Recovery

- **Default Fallback:** System automatically uses compiled-in certificates on errors
- **Library Defaults:** Use `SslUserSetDefault()` to restore NetBurner default certificates
- **Backup Restoration:** Restore certificates from backup files if available

12.43.6.1.12.3 Debug and Monitoring

Serial Console Diagnostics

- **Debug Level Control:** Commands '0' and '1' to control debug message verbosity
- **File System Status:** Command 'F' shows detailed file system information
- **Network Status:** Command 'N' displays current NV settings
- **Time Verification:** Command 'T' shows current system time

Log Analysis

- **SSL Errors:** Look for certificate validation failures in debug output
- **File System Errors:** Check for flash read/write/erase failures
- **Network Issues:** Monitor for [DHCP](#), DNS, or NTP synchronization problems
- **Memory Issues:** Watch for allocation failures during file uploads

Performance Monitoring

- **Flash Wear:** Monitor bad block count and wear leveling statistics
- **Memory Usage:** Check heap usage during certificate operations
- **Network Performance:** Monitor connection establishment times for HTTPS
- **Certificate Chain Length:** Verify CA certificate chain length for client verification

12.43.6.1.12.4 Platform-Specific Issues

MOD5441X Parallel Flash

- **Address Conflicts:** Verify COMPCODEFLAGS settings in makefile
- **Flash Programming:** Check parallel flash programming voltage levels
- **Sector Alignment:** Ensure proper 128KB sector boundary alignment

NANO54415/SB800EX SPI Flash

- **SPI Interface:** Verify SPI clock frequency and signal integrity
- **RAM Mirror:** Monitor RAM usage for file system mirror
- **Write Performance:** SPI flash writes slower than parallel flash

MODM7AE70/SBE70LC On-Chip Flash

- **Background Erase:** Disable background erase feature in NBEclipse
- **Fixed Address:** Enable "Locate Application at Fixed Address in Flash"
- **Library Linking:** Ensure StdFFile.a library is properly linked

12.43.6.1.13 Development Notes

12.43.6.1.13.1 Memory Management

- Dynamic memory allocation for file uploads with automatic cleanup
- Buffer size limits prevent memory exhaustion
- Careful handling of certificate data to prevent memory leaks

12.43.6.1.13.2 Error Handling

- Comprehensive error checking for all file operations
- Graceful degradation on certificate validation failures
- User feedback through both web interface and debug console
- Automatic recovery mechanisms for corrupted installations

12.43.6.1.13.3 Browser Compatibility

Modern browsers require valid certificates and may reject self-signed certificates. For development:

- Install the Certificate Authority certificate in the browser
- Use NetBurner SDK certificate generation scripts (`makeca`, `makeserver`)
- Consider using proper CA-signed certificates for production deployment

12.43.6.1.14 Version Information

- **Application Version:** 01.00.0000
- **Supported OS:** NetBurner RTOS 3.x
- **File System:** EFFS-STD (Enhanced Flash File System Standard)
- **SSL/TLS:** NetBurner SSL library with OpenSSL compatibility

12.43.7 HTTPS Web Server Demo

The `HTTP_ACCESS CheckHttpAccess(int sock, int access_level, HTTP_Request &Req)` function is used to authenticate the web page access. It will allow access or redirect depending on the file and access rights.

12.43.7.1 SSL/TLS Web Server Demo

A NetBurner embedded web server application demonstrating secure HTTPS and non-secure HTTP access controls with SSL/TLS certificate authentication.

12.43.7.1.1 Overview

This application creates a dual-protocol web server that serves content over both HTTP (port 80) and HTTPS (port 443). It implements access control mechanisms to restrict certain content to secure connections only, demonstrating how to build embedded web applications with security-conscious file serving.

12.43.7.1.2 Features

- **Dual Protocol Support:** Serves content over both HTTP and HTTPS
- **Access Control:** Implements file-level security restrictions
- **SSL/TLS Authentication:** Uses embedded certificates for secure connections
- **Dynamic Content:** Provides callback functions for server-side processing
- **Automatic Redirection:** Handles unauthorized access attempts gracefully

12.43.7.1.3 Directory Structure

```
html
|-- index.html           # Public page (HTTP/HTTPS accessible)
|-- images               # Image directory (HTTP/HTTPS accessible)
    |-- SSL-Good.gif      # Displayed for HTTPS connections
    |-- SSL-Bad.gif       # Displayed for HTTP connections
    |-- Other misc images
|-- httpsdir             # Secure directory (HTTPS only)
    |-- repeat.html       # Secure page requiring SSL/TLS
```

12.43.7.1.4 Access Control System

The application implements a two-tier access control system:

12.43.7.1.4.1 Access Level 0 (Public)

- **Files:** `index.html`, content in `images/` directory
- **Access:** Available via both HTTP and HTTPS
- **Authentication:** None required

12.43.7.1.4.2 Access Level 1 (Secure)

- **Files:** Content in `httpsdir/` directory (e.g., `repeat.html`)
- **Access:** HTTPS connections only
- **Authentication:** Valid SSL/TLS connection required
- **Behavior:** HTTP requests return `HTTP_FORBIDDEN` status

12.43.7.1.5 Core Components

12.43.7.1.5.1 Main Application (`main.cpp`)

Key Functions:**

- `UserMain()`: Application entry point, initializes network stack and starts web server
- `CheckHttpAccess()`: Custom access control function that overrides system defaults
- `DoCounter()`: Demonstrates server-side content generation for SSL connections
- `SSL_Image()`: Returns different images based on connection security
- `GetHost()` / `ShowHost()`: Host header parsing utilities
- `HTTPS_Ref()` / `HTTP_Ref()`: Generate protocol-specific URL references

Network Initialization:**

- Waits for `DHCP` address assignment (5-second timeout)
- Starts web server on default ports (80 for HTTP, 443 for HTTPS)
- Enables system diagnostics for debugging

12.43.7.1.5.2 Security Certificates

Server Certificate (`ServerCert.cpp`)**

- Contains embedded SSL/TLS certificate data
- 672-byte PEM-encoded certificate
- Used for HTTPS connection authentication

Private Key (`ServerKey.cpp`)**

- Contains embedded private key data
- 310-byte PEM-encoded EC private key
- Used for SSL/TLS handshake and encryption

12.43.7.1.6 How It Works

1. **Server Startup:** Application initializes network stack and starts dual-protocol web server
2. **Connection Handling:** Server accepts both HTTP and HTTPS connections
3. **Access Control:** `CheckHttpAccess()` function evaluates each request:
 - Level 0: Grants access regardless of protocol
 - Level 1: Requires SSL/TLS connection (`IsSSLfd()` check)
4. **Content Serving:**
 - Public content served over any protocol
 - Secure content only served over HTTPS
 - Dynamic content generated via callback functions

12.43.7.1.7 Security Features

- **Protocol Enforcement:** Secure directories reject non-SSL connections
- **Certificate-Based Authentication:** Uses embedded certificates for identity verification
- **Graceful Degradation:** Non-secure requests to secure content return proper HTTP error codes
- **Dynamic Security Indicators:** Different visual feedback based on connection security

12.43.7.1.8 Network Configuration

- **DHCP:** Automatically obtains IP address via DHCP
- **Ports:**
 - HTTP: 80 (configurable)
 - HTTPS: 443 (configurable)
- **Protocols:** HTTP/1.1, HTTPS with SSL/TLS

12.43.7.1.9 Usage Example

1. Deploy application to NetBurner device
2. Connect device to network (DHCP-enabled)
3. Access public content:
 - `http://device-ip/index.html` (works)
 - `https://device-ip/index.html` (works)
4. Access secure content:
 - `http://device-ip/httpsdir/repeat.html` (forbidden)
 - `https://device-ip/httpsdir/repeat.html` (works)

12.43.7.1.10 Customization

To modify access control:

1. Edit `CheckHttpAccess()` function logic
2. Adjust access levels in HTML files or `.nbaccess` files
3. Modify directory structure as needed
4. Update certificate/key files for production deployment

12.43.7.1.11 Dependencies

- NetBurner RTOS
- NetBurner HTTP/HTTPS libraries
- SSL/TLS crypto libraries
- Network interface libraries

12.43.8 HTTPS GET Request

12.43.8.1 SSL HTTPS GET Example

12.43.8.1.1 Overview

This application demonstrates how to create an SSL/TLS client that can retrieve web pages from HTTPS servers using the NetBurner RTOS platform. The example shows basic secure HTTP communication over SSL/TLS connections.

12.43.8.1.2 Description

The SSL HTTPS GET Example is a command-line application that allows users to:

- Connect to HTTPS servers by hostname
- Establish secure SSL/TLS connections on port 443
- Send HTTP GET requests to retrieve web page content
- Display the response data in a terminal interface

12.43.8.1.3 Key Features

- **DNS Resolution:** Resolves hostnames to IP addresses using the built-in DNS client
- **SSL/TLS Support:** Establishes secure connections with certificate verification
- **Interactive Interface:** Command-line prompt for entering server hostnames
- **Error Handling:** Comprehensive error reporting for SSL connection failures
- **Network Diagnostics:** Optional system diagnostics for debugging

12.43.8.1.4 Prerequisites

Important Certificate Requirements:**

You must either:

1. Include a CA (Certificate Authority) certificate list in your project, OR
2. Recompile the cryptolib with `NB_SSL_CLIENT_CERTIFICATE_CHECKING_ENABLED` defined in `sslclient.cpp`

Without proper certificate configuration, SSL connections may fail with certificate verification errors.

12.43.8.1.5 Usage

1. **Build and Deploy:** Compile and deploy the application to your NetBurner device
2. **Run Terminal:** Start the MTTY serial terminal application
3. **Network Setup:** Ensure the device has obtained a [DHCP](#) address (5-second timeout)
4. **Enter Hostname:** When prompted, enter an HTTPS server name (e.g., `www.google.com`)
5. **View Results:** The application will display the retrieved web page content

12.43.8.1.5.1 Example Session

```
Enter the destination HTTPS server name: www.google.com

Getting [www.google.com]

www.google.com = 172.217.164.100

We connected
Read 1024 bytes
[HTTP/1.1 200 OK
Content-Type: text/html
...]
```

12.43.8.1.6 Application Flow

1. **Initialization:** Network stack initialization and [DHCP](#) address acquisition
2. **User Input:** Prompts for HTTPS server hostname
3. **DNS Lookup:** Resolves hostname to IP address
4. **SSL Connection:** Establishes secure connection on port 443
5. **HTTP Request:** Sends GET / request to the server
6. **Response Handling:** Reads and displays up to 20KB of response data
7. **Cleanup:** Closes connection and repeats for next request

12.43.8.1.7 Error Handling

The application provides detailed error messages for common SSL connection failures:

- `SSL_ERROR_FAILED_NEGOTIATION`: SSL handshake failed
- `SSL_ERROR_CERTIFICATE_UNKNOWN`: Server certificate not recognized
- `SSL_ERROR_CERTIFICATE_NAME_FAILED`: Certificate hostname mismatch
- `SSL_ERROR_CERTIFICATE_VERIFY_FAILED`: Certificate verification failed
- DNS resolution failures

12.43.8.1.8 Technical Details

12.43.8.1.8.1 Buffer Specifications

- **Receive Buffer:** 20KB capacity for response data
- **Display Limit:** First 512 bytes shown in terminal output
- **Hostname Buffer:** 255 character maximum length

12.43.8.1.8.2 Network Configuration

- **Protocol:** HTTPS over SSL/TLS
- **Port:** 443 (standard HTTPS port)
- **Timeout:** 400 ticks for SSL connection attempts
- **Retry Logic:** Automatic retry on initial connection failure

12.43.8.1.8.3 Dependencies

- NetBurner RTOS
- SSL/TLS crypto library
- DNS resolution system
- Network I/O subsystem

12.43.8.1.9 Debugging

The application includes optional network debugging capabilities when compiled with the `_DEBUG` flag. System diagnostics are enabled by default but should be removed for production deployments.

12.43.8.1.10 Limitations

- Maximum response size: 20KB
- Single-threaded operation
- Basic HTTP GET requests only
- No support for HTTP authentication
- Limited certificate validation options

12.43.9 SSL/TLS Client

12.43.9.1 SSL/TLS Client Example

A NetBurner embedded device application that demonstrates SSL/TLS client connections with automated testing and connection monitoring.

12.43.9.1.1 Overview

This application performs automated SSL/TLS client connections to a specified server, tracking both successful and failed connection attempts. It provides real-time status updates through both network messages and serial debug output, making it useful for testing SSL/TLS connectivity and performance analysis.

12.43.9.1.2 Features

- Automated SSL/TLS client connections with configurable target server
- Connection attempt tracking (successes and failures)
- Performance timing measurements
- Dual output channels: network messages to server and serial debug
- Comprehensive error reporting with specific SSL error codes
- Built-in retry mechanism with configurable delays
- DNS hostname resolution support

12.43.9.1.3 Configuration

12.43.9.1.3.1 Server Settings

Modify the following definitions in `main.cpp` to configure your target SSL server:

```
#define SSL_SERVER_NAME "192.168.0.53" // Server IP address or hostname
#define SSL_SERVER_PORT 4433          // Server port number
```

12.43.9.1.3.2 Certificate Checking (Optional)

Certificate checking is disabled by default. To enable certificate verification:

1. Include a CA certificate list in your project
2. Uncomment `#define NB_SSL_CLIENT_CERTIFICATE_CHECKING_ENABLED` in `sslclient.cpp`
3. Rebuild the system libraries

12.43.9.1.4 Application Behavior

12.43.9.1.4.1 Connection Process

1. **Initialization:** Waits for active network ([DHCP](#)) for up to 5 seconds
2. **DNS Resolution:** Resolves the server hostname to IP address (15-second timeout)
3. **SSL Connection:** Attempts SSL/TLS connection (20-second timeout)
4. **Message Exchange:** Sends connection statistics to the server
5. **Cleanup:** Closes connection and waits before next attempt

12.43.9.1.4.2 Test Parameters

- **Maximum Attempts:** 50 connections
- **Connection Timeout:** 20 seconds
- **DNS Timeout:** 15 seconds
- **Retry Delay:** 0.5 seconds between attempts
- **Network Wait:** 5 seconds for [DHCP](#)

12.43.9.1.4.3 Output Information

For each connection attempt, the application reports:

- Target server and port
- Connection start timestamp
- Success/failure status with error details
- Total connection time in system ticks and seconds
- Running totals of successful and failed attempts

12.43.9.1.5 Testing with OpenSSL

You can test the client using OpenSSL's server functionality:

```
openssl s_server -accept 4433 -cert Server.crt -key Server.key
```

Replace `Server.crt` and `Server.key` with your certificate files. These can be created using:

- NetBurner development tools
- OpenSSL certificate generation commands
- Certificate Authority (CA) issued certificates

Refer to the OpenSSL `s_server` documentation for additional configuration options.

12.43.9.1.6 Error Handling

The application provides detailed error reporting for SSL connection failures:

- **SSL_ERROR_FAILED_NEGOTIATION:** SSL handshake negotiation failed
- **SSL_ERROR_CERTIFICATE_UNKNOWN:** Unknown certificate error
- **SSL_ERROR_CERTIFICATE_NAME_FAILED:** Certificate name verification failed
- **SSL_ERROR_CERTIFICATE_VERIFY_FAILED:** Certificate verification failed
- **Other errors:** Generic SSL connection errors

12.43.9.1.7 Performance Monitoring

Connection timing is measured and reported for each attempt:

- System tick timestamps for precise timing
- Approximate duration in seconds
- Running statistics for performance analysis

12.43.9.1.8 Build Requirements

- NetBurner development environment
- SSL/TLS libraries enabled
- Network stack with DNS resolution support
- Serial debug output capability

12.43.9.1.9 Usage Notes

- The application runs continuously until 50 connections are completed
- After completion, the device must be reset to repeat the test
- Enable system diagnostics for development (remove for production)
- Ensure target server is accessible and properly configured
- Monitor both serial output and network messages for complete status

12.43.9.1.10 Development Considerations

- Remove `EnableSystemDiagnostics()` call for production builds
- Adjust timeout values based on network conditions
- Consider certificate validation requirements for production use
- Monitor memory usage during extended connection testing

12.43.10 SSL/TLS Client Certificate

12.43.10.1 SSL Client Certificate Example

12.43.10.1.1 Overview

This application demonstrates how to implement an SSL/TLS client with client certificate authentication using the NetBurner platform. The example shows how to establish secure connections to an SSL server that requires client certificates for authentication.

12.43.10.1.2 Program Description

The SSL Client Certificate Example demonstrates three key concepts:

1. **SSL Client Connections** - How to establish SSL/TLS connections from a NetBurner device
2. **Client Certificate Loading** - How to load and use client certificates compiled into the application
3. **Certificate-Based Authentication** - How to authenticate with servers that require client certificates

12.43.10.1.3 How It Works

The application performs the following operations:

1. **Network Initialization** - Initializes the network stack and waits for [DHCP](#) configuration
2. **SSL Setup** - Initializes SSL and configures client certificate credentials
3. **Connection Loop** - Repeatedly attempts to connect to the specified SSL server
4. **Status Reporting** - Tracks and reports successful and failed connection attempts

12.43.10.1.4 Key Features

12.43.10.1.4.1 Connection Process

- Connects to SSL server on port 4433
- Uses embedded client certificate and private key for authentication
- Sends connection statistics to both the server and debug console
- Handles various SSL error conditions with detailed error reporting

12.43.10.1.4.2 Error Handling

The application provides detailed error reporting for common SSL failures:

- `SSL_ERROR_FAILED_NEGOTIATION` - SSL handshake negotiation failed
- `SSL_ERROR_CERTIFICATE_UNKNOWN` - Certificate not recognized
- `SSL_ERROR_CERTIFICATE_NAME_FAILED` - Certificate name validation failed
- `SSL_ERROR_CERTIFICATE_VERIFY_FAILED` - Certificate verification failed

12.43.10.1.5 File Structure

```
main.cpp           # Main application logic
clientcert.cpp     # Embedded client certificate data
clientkey.cpp      # Embedded private key data
```

12.43.10.1.6 Configuration

12.43.10.1.6.1 Target Server

Modify the `SSL_SERVER_IP` definition in `main.cpp` to specify your target SSL server:

```
#define SSL_SERVER_IP "192.168.1.100" // Replace with your server IP
```

12.43.10.1.6.2 Certificates

The client certificate and private key are embedded as C arrays in the application:

- Certificate: 672 bytes in PEM format
- Private key: 310 bytes in PEM format (EC parameters and private key)

12.43.10.1.7 Important Distinctions

This example demonstrates **client certificate authentication**, which is different from standard certificate authority (CA) validation:

12.43.10.1.7.1 CA Certificate Checking

- Client validates server certificate against known Certificate Authorities
- Requires modification to `predef.h` to enable
- Standard SSL/TLS client behavior

12.43.10.1.7.2 Client Certificate Authentication

- Client sends its own certificate to the server for authentication
- Server validates the client certificate
- Provides mutual authentication (both parties verify each other)
- **This is what this example demonstrates**

12.43.10.1.8 Usage

1. **Setup:** Configure the target server IP address
2. **Compile:** Build the application with the embedded certificates
3. **Deploy:** Flash to NetBurner device
4. **Run:** The application will wait for network initialization, then prompt for connection attempts
5. **Interact:** Press any key to initiate connection attempts, press 'C' to retry after failures

12.43.10.1.9 Requirements

- NetBurner development platform with SSL/TLS support
- SSL server configured to request and validate client certificates
- Network connectivity ([DHCP](#) recommended)

12.43.10.1.10 Output

The application provides real-time feedback including:

- Connection attempt notifications
- Success/failure statistics
- Detailed error codes for troubleshooting
- Performance timing information

12.43.10.1.11 Security Notes

- Client certificates provide strong authentication but must be properly managed
- Private keys should be protected and not exposed in production environments
- Consider using secure storage mechanisms for production deployments
- Ensure proper certificate lifecycle management (expiration, revocation, etc.)

12.43.10.1.12 Troubleshooting

Common issues and solutions:

- **Connection timeouts:** Verify server IP and network connectivity
- **Certificate errors:** Ensure server is configured to accept the client certificate
- **SSL negotiation failures:** Check SSL/TLS version compatibility
- **Network issues:** Verify [DHCP](#) configuration and routing

12.43.11 SSL/TLS Client Verify Peer - Basic

12.43.11.1 SSL/TLS Client Verify Peer - Basic

12.43.11.1.1 Overview

This application demonstrates how to implement SSL/TLS client connections with peer certificate verification using embedded CA (Certificate Authority) lists. The program establishes secure connections to HTTPS servers and validates their certificates against a trusted root certificate.

12.43.11.1.2 Features

- **SSL/TLS Client Connection:** Establishes secure connections to HTTPS servers on port 443
- **Certificate Verification:** Validates server certificates using embedded CA certificates
- **Interactive Domain Testing:** Allows users to test connections to any domain
- **NTP Time Synchronization:** Automatically synchronizes system time for accurate certificate validation
- **Comprehensive Error Handling:** Provides detailed SSL error reporting

12.43.11.1.3 Application Components

12.43.11.1.3.1 Core Files

- **main.cpp:** Main application logic with SSL connection handling
- **caList.h:** Contains embedded DigiCert High Assurance EV Root CA certificate for GitHub
- **ServerCert.cpp:** Auto-generated server certificate data (binary format)
- **ServerKey.cpp:** Auto-generated EC private key data (binary format)

12.43.11.1.3.2 Key Functionality

SSL Connection Process

1. Prompts user for domain name input
2. Resolves domain to IP address using DNS
3. Establishes SSL connection with certificate verification enabled
4. Sends basic HTTP GET request
5. Reads and displays server response
6. Provides detailed error information on connection failures

Time Synchronization

- Uses NTP (Network Time Protocol) to set system time
- Connects to `pool.ntp.org` for time synchronization
- Ensures accurate certificate validity period checking

12.43.11.1.4 Usage Instructions

12.43.11.1.4.1 Running the Application

1. **Start the Application:** The program initializes network connections and synchronizes time
2. **Access Debug Menu:** Press any key to display available commands
3. **Test SSL Connection:** Press 'C' to initiate an SSL connection test
4. **Enter Domain:** Type the domain name you want to test (e.g., `github.com`, `google.com`)
5. **View Results:** The application will display connection status and any errors

12.43.11.1.4.2 Sample Session

```
Commands :  
C: Attempt SSL Connection  
  
[Press 'C']  
  
Enter the domain you would like to connect to: github.com  
  
Attempting to connect to: github.com:443 (140.82.114.4)  
Good SSL connection  
Read 1024 bytes  
Read 512 bytes  
Connection closed with response: 0  
Read 1536 total bytes
```

12.43.11.1.5 Certificate Verification

The application uses a hardcoded DigiCert High Assurance EV Root CA certificate to verify connections. This certificate is specifically configured for GitHub connections but can validate any server using certificates signed by this CA.

12.43.11.1.5.1 Supported Error Types

- **SSL_ERROR_FAILED_NEGOTIATION:** SSL handshake failed
- **SSL_ERROR_CERTIFICATE_UNKNOWN:** Certificate not recognized
- **SSL_ERROR_CERTIFICATE_NAME_FAILED:** Certificate name mismatch
- **SSL_ERROR_CERTIFICATE_VERIFY_FAILED:** Certificate validation failed

12.43.11.1.6 Technical Details

12.43.11.1.6.1 Dependencies

- **NBRTOS:** Real-time operating system
- **DNS:** Domain name resolution
- **TCP:** Network communication
- **SSL/TLS:** Secure socket layer implementation
- **NTP:** Network time protocol

12.43.11.1.6.2 Memory Usage

- **RX Buffer:** 20KB buffer for reading server responses
- **Certificate Storage:** Embedded certificates stored in program memory
- **Domain Buffer:** 255 character buffer for domain name input

12.43.11.1.6.3 Network Configuration

- **SSL Port:** 443 (HTTPS)
- **Connection Timeout:** 60 seconds
- **Read Timeout:** 5 seconds per operation
- **NTP Timeout:** 10 seconds for time synchronization

12.43.11.1.7 Security Considerations

12.43.11.1.7.1 Certificate Validation

- Peer verification is enabled for all connections
- Uses embedded root CA certificates for trust chain validation
- Validates certificate names against requested domains
- Checks certificate validity periods (requires accurate system time)

12.43.11.1.7.2 Limitations

- Only includes DigiCert root CA certificate
- Cannot verify certificates from other CAs without modification
- No certificate revocation checking (CRL/OCSP)

12.43.11.1.8 Development Notes

12.43.11.1.8.1 Customization

To add support for additional CAs:

1. Obtain the root CA certificate in PEM format
2. Add certificate data to `caList.h`
3. Update certificate length and data arrays
4. Recompile the application

12.43.11.1.8.2 Error Handling

The application provides comprehensive error reporting for:

- DNS resolution failures
- Network connection issues
- SSL handshake problems
- Certificate validation errors

12.43.11.1.9 Example Use Cases

- **Security Testing:** Verify SSL configuration of web servers
- **Certificate Validation:** Test certificate chain validity
- **Network Diagnostics:** Debug SSL connection issues
- **Educational Purposes:** Learn SSL/TLS implementation details

12.43.11.1.10 Build Requirements

- NBR-TOS development environment
- Network-enabled target hardware
- Internet connectivity for DNS and NTP
- Sufficient memory for SSL operations and certificate storage

12.43.12 SSL/TLS Client Verify Peer - EFFS-STD

12.43.12.1 SSL/TLS Client Verify Peer - EFFS-STD

12.43.12.1.1 Overview

This application demonstrates how to implement SSL/TLS client-side certificate verification using Certificate Authority (CA) lists stored in the EFFS Standard on-chip embedded flash file system (EFFS-STD). The application provides both a web interface and serial debug menu for managing CA certificates and testing SSL connections.

12.43.12.1.2 Features

12.43.12.1.2.1 SSL/TLS Certificate Verification

- Client-side verification of server certificates using uploaded CA certificates
- Support for PEM-format certificate files
- Dynamic loading and management of CA certificate lists
- Secure storage of certificates in on-chip flash memory using EFFS-STD

12.43.12.1.2.2 Web Interface

- Upload CA certificates through a web interface
- View and manage stored certificates
- Delete certificates with confirmation dialogs
- Display certificate details and public keys
- Automatic certificate scanning and loading

12.43.12.1.2.3 File System Management

- Embedded Flash File System (EFFS-STD) for persistent storage
- Automatic formatting and initialization
- File system statistics and directory listing
- Support for multiple platforms (MOD5441X, NANO54415, MODM7AE70, SB800EX, SBE70LC)

12.43.12.1.2.4 Debug and Testing

- Serial console interface for debugging
- Interactive SSL connection testing
- System time synchronization via NTP
- Comprehensive logging and error reporting

12.43.12.1.3 Hardware Support

The application supports multiple NetBurner platforms:

- MOD5441X (32MB flash with 128KB sectors)
- NANO54415 (8MB SPI flash with 4KB sectors)
- MODM7AE70 (2MB flash with 8KB sectors)
- SB800EX (8MB SPI flash with 4KB sectors)
- SBE70LC (2MB flash with 8KB sectors)

12.43.12.1.4 File Structure

12.43.12.1.4.1 Core Application Files

- `main.cpp` - Main application entry point and user interface
- `certificatekey.cpp` - CA certificate management and SSL operations
- `configweb.cpp` - Web interface configuration and NV settings
- `post.cpp` - HTTP POST handlers for web interface

12.43.12.1.4.2 File System Components

- `effs_std.cpp` - EFFS-STD initialization and operations
- `effsStdFlashDrv.cpp` - Platform-specific flash drivers
- `FileSystemUtils.cpp` - File system utility functions
- `fs_main.cpp` - File system initialization

12.43.12.1.4.3 Supporting Modules

- `effs_time.cpp` - Time management and NTP synchronization
- `ftp_fs.cpp` - FTP server integration with file system
- `ramdrv_mcf.cpp` - RAM drive configuration

12.43.12.1.4.4 Configuration Headers

- `nbfactory.h` - Platform-specific factory settings
- `serialburnerdata.h` - Configuration structures and constants
- `effs_std.h` - EFFS-STD definitions and prototypes

12.43.12.1.5 Quick Start

12.43.12.1.5.1 Initial Setup

1. Flash the application to your NetBurner device
2. Connect to the device's network interface
3. Access the web interface at the device's IP address
4. The default page will be `CaCert.html` for certificate management

12.43.12.1.5.2 Adding CA Certificates

1. Navigate to the certificate management page via web interface
2. Upload PEM-format CA certificate files
3. Certificates are automatically scanned and loaded into the SSL engine
4. Certificate files are stored as `CACertN.crt` where N is an incremental number

12.43.12.1.5.3 Testing SSL Connections

1. Access the serial debug interface
2. Press 'C' to attempt an SSL connection
3. Enter the domain name when prompted
4. The application will verify the server certificate against loaded CA certificates

12.43.12.1.6 Debug Commands

Access the debug menu through the serial interface:

- 0 - Turn debug messages off
- 1 - Turn debug messages on
- C - Attempt SSL connection test
- D - Reset to factory default parameters
- F - Show file system information
- G - Format file system (caution: erases all data)
- N - Display NV settings
- T - Display system time

12.43.12.1.7 Certificate Management

12.43.12.1.7.1 Supported Formats

- PEM-encoded X.509 certificates
- Maximum certificate size: 5KB
- Automatic parsing of certificate common names

12.43.12.1.7.2 Storage Details

- Certificates stored in on-chip flash memory
- Maximum 20 CA certificates supported
- Automatic verification of certificate validity
- Certificates persist across power cycles

12.43.12.1.7.3 Web Interface Operations

- Upload new certificates
- View certificate details and public keys
- Delete certificates with confirmation
- Automatic rescanning after modifications

12.43.12.1.8 Network Configuration

12.43.12.1.8.1 Default Settings

- [DHCP](#) client enabled by default
- NetBIOS name publishing
- HTTPS server on port 443
- NTP time synchronization

12.43.12.1.8.2 Time Management

- Automatic NTP synchronization on startup
- Configurable timezone support (defaults to PST8PDT)
- Manual time setting capability
- RTC support on compatible platforms

12.43.12.1.9 Error Handling

The application includes comprehensive error handling for:

- File system operations
- SSL/TLS connection failures
- Certificate validation errors
- Network connectivity issues
- Memory allocation failures

Common SSL error codes:

- `SSL_ERROR_FAILED_NEGOTIATION` - SSL handshake failed
- `SSL_ERROR_CERTIFICATE_UNKNOWN` - Unknown certificate error
- `SSL_ERROR_CERTIFICATE_NAME_FAILED` - Certificate name verification failed
- `SSL_ERROR_CERTIFICATE_VERIFY_FAILED` - Certificate verification failed

12.43.12.1.10 Security Considerations

- Certificates are validated before storage
- SSL connections require valid certificate chains
- Peer verification is enabled by default when CA certificates are loaded
- Factory reset capability to clear all certificates
- Secure storage in flash memory with verification keys

12.43.12.1.11 Troubleshooting

12.43.12.1.11.1 Common Issues

1. **SSL Connection Failures:** Ensure CA certificates are properly loaded and system time is correct
2. **File System Errors:** Use format command ('G') to reinitialize file system
3. **Certificate Upload Failures:** Verify certificate is in PEM format and under 5KB
4. **Time Synchronization:** Check network connectivity and DNS settings for NTP

12.43.12.1.11.2 Factory Reset

Use debug command 'D' to reset to factory defaults, which will:

- Clear all user configuration
- Format the file system
- Remove all uploaded certificates
- Reset network settings

12.43.12.1.12 Example Usage

The application includes `github.pem` as a test certificate. To test:

1. Upload the `github.pem` file through the web interface
2. Use the serial debug menu (command 'C')
3. Enter "github.com" as the test domain
4. Verify successful SSL connection with certificate validation

12.43.13 SSL/TLS Receive Mail (POP3)

12.43.13.1 SSL POP3 Mail Example

12.43.13.1.1 Overview

This NetBurner embedded application demonstrates how to retrieve email from a server that requires SSL/TLS using the POP3 protocol. The application provides a web interface for configuring mail server settings and retrieving messages, which are then saved to external flash storage.

Note

When connecting to a Google Account, you will likely need to enable less secure application access. If you use two-factor authentication with your account, you will need to setup an application password for the account being connected to. This can be done at the following URL: <https://myaccount.google.com/apppasswords>. The application password is not saved and will need to be reentered for every request.

12.43.13.1.2 Key Features

- **SSL/TLS POP3 Email Retrieval:** Secure connection to POP3 mail servers
- **Web-based Configuration Interface:** HTML forms for setting up mail server credentials
- **File System Integration:** Saves retrieved emails to SD/MMC/SDHC cards or Compact Flash
- **FTP Server:** Built-in FTP server for file management
- **HTTPS Web Server:** Secure web interface on port 443
- **Multiple Storage Support:** Compatible with various flash storage types

12.43.13.1.3 Application Components

12.43.13.1.3.1 Main Application (main.cpp)

The main application initializes:

- Network stack with [DHCP](#) support
- File system for multiple task priorities
- External flash storage (SD/MMC/SDHC/CFC)
- HTTPS web server on port 443
- FTP server on port 21

12.43.13.1.3.2 File System Utilities

- **FileSystemUtils.cpp/h:** Comprehensive file system operations including mount/unmount, format, read/write operations, and directory management
- **cardtype.h:** Hardware abstraction for different storage types (SD/MMC, SDHC, CFC, RAM)

12.43.13.1.3.3 Network Services

- **FTP Server (ftp_f.cpp/h):** Full FTP implementation with file transfer, directory operations, and multi-drive support
- **Web Interface (webfuncs.cpp):** HTTP POST/GET handlers for email configuration and retrieval

12.43.13.1.3.4 Time Management

- **effs_time.cpp/h:** Time synchronization using NTP servers, manual time setting, and RTC integration for file timestamps

12.43.13.1.4 Web Interface

The application provides an HTML-based interface accessible via HTTPS:

12.43.13.1.4.1 Main Page (index.html)

- Network configuration display (IP, gateway, DNS)
- POP3 server settings form:
 - Mail username and password
 - POP3 server hostname/IP
 - Port number (default 995 for SSL)
 - Number of messages to retrieve

12.43.13.1.4.2 Results Page (sent.html)

- Displays retrieval status
- Provides links to downloaded email files
- Shows error details if retrieval fails

12.43.13.1.5 Storage Configuration

The application supports multiple storage types configured in `cardtype.h`:

- **SDHC:** SD/SDHC cards (default for SB800EX and SOMRT1061 platforms)
- **MMC:** SD/MMC cards
- **CFC:** Compact Flash cards
- **RAM:** RAM-based file system for testing

12.43.13.1.6 Gmail Integration Notes

When connecting to Gmail accounts:

1. Enable "Less secure app access" or use App Passwords for 2FA accounts
2. Configure App Password at: <https://myaccount.google.com/apppasswords>
3. Use server: pop.gmail.com, port: 995
4. The App Password must be re-entered for each session (not stored)

12.43.13.1.7 File Operations

Retrieved emails are automatically:

- Converted to HTML format for web viewing
- Saved with sequential filenames (email1.html, email2.html, etc.)
- Accessible via web browser or FTP client
- Timestamped using system time

12.43.13.1.8 Error Handling

The application includes comprehensive error handling for:

- Network connectivity issues
- DNS resolution failures
- SSL/TLS connection problems
- File system errors
- Storage device detection and write protection

12.43.13.1.9 Build Requirements

- NetBurner development environment
- SSL/TLS library support
- EDFS (Embedded Flash File System) libraries
- Network stack with [DHCP](#) and DNS support

12.43.13.1.10 Usage Instructions

1. Connect and power the NetBurner device
2. Ensure SD card or other storage is inserted
3. Access the web interface via HTTPS
4. Configure network settings if using static IP
5. Enter POP3 server credentials
6. Click "Get Mail" to retrieve messages
7. Access downloaded emails via provided links or FTP

12.43.14 SSL/TLS Send Mail

12.43.14.1 SSL/TLS Send Mail Example

A NetBurner embedded application that demonstrates secure email transmission using SSL/TLS protocols.

12.43.14.1.1 Overview

This application illustrates how to send email messages from a NetBurner device to mail servers that require SSL/TLS encryption. The implementation provides a web-based interface for configuring email settings and sending messages securely over HTTPS connections.

12.43.14.1.2 Key Features

- SSL/TLS encrypted email transmission
- Web-based configuration interface
- Support for both direct SSL and STARTTLS negotiation methods
- SMTP authentication support
- Network time synchronization via NTP
- File system utilities for external storage devices

12.43.14.1.3 SSL/TLS Connection Methods

The application supports two SSL/TLS handshake methods:

12.43.14.1.3.1 Direct SSL Connection

- Connects directly to SSL/TLS ports (465 or 587)
- Uses `SSL_connect` for immediate secure connection
- Recommended for servers known to support SSL/TLS
- Suitable for services like Yahoo that expect SSL connections on specific ports

12.43.14.1.3.2 STARTTLS Method

- Initially connects to standard SMTP port 25
- Performs `EHLO` command to check server capabilities
- If server responds with "STARTTLS", initiates SSL negotiation
- Required for Gmail and servers that mandate TLS encryption

12.43.14.1.4 Configuration Requirements

12.43.14.1.4.1 Network Setup

- Valid IP address, network mask, gateway, and DNS configuration
- Internet connectivity for NTP time synchronization and mail transmission
- [DHCP](#) support available (set IP address to 0.0.0.0 to enable)

12.43.14.1.4.2 Gmail Configuration

When connecting to Gmail accounts:

1. Enable "less secure application access" in account settings
2. For two-factor authentication accounts, create an application-specific password
3. Use server: `smtp.gmail.com`
4. Use port: 465 or 587
5. Leave STARTTLS unchecked
6. Generate app passwords at: <https://myaccount.google.com/apppasswords>

12.43.14.1.4.3 Yahoo Configuration

- Requires premium services (mailplus or bizmail) - free accounts not supported
- Does not support STARTTLS detection
- Connect to port 465/587 for SSL or port 25 for non-SSL

12.43.14.1.5 Application Structure

12.43.14.1.5.1 Core Components

main.cpp**

- Application entry point and initialization
- Network stack setup and HTTPS server startup
- NTP time synchronization
- System diagnostics enablement

webfuncs.cpp**

- Web interface implementation
- Form handling for email configuration
- SSL email transmission logic
- Network and DNS management

index.html**

- Web-based user interface
- Dynamic port selection based on STARTTLS setting
- Network configuration display
- Email form with validation

12.43.14.1.5.2 File System Support

The application includes comprehensive file system utilities supporting:

- SD/MMC cards
 - SDHC cards
 - Compact Flash (CFC)
 - RAM drives
- Supported Operations:**
- File creation, reading, writing, and deletion
 - Directory management
 - FTP server functionality
 - Multiple storage device types

12.43.14.1.6 Usage Instructions

1. Network Configuration

- Verify IP gateway and DNS settings
- Enable [DHCP](#) by setting IP to 0.0.0.0 (optional)

2. Email Configuration

- Enter mail username and password
- Specify SMTP server hostname
- Set appropriate port (25, 465, or 587)
- Configure STARTTLS if required by server

3. Send Email

- Fill in sender and recipient addresses
- Enter subject and message body
- Click "Send Mail" to transmit

12.43.14.1.7 Technical Implementation

12.43.14.1.7.1 Security Features

- SSL/TLS encryption for all mail transmission
- HTTPS web interface
- Secure credential handling
- DNS lookup validation

12.43.14.1.7.2 Error Handling

- Comprehensive DNS failure detection
- SSL connection error reporting
- File system error codes with descriptions
- Network timeout management

12.43.14.1.7.3 Development Requirements

NBEclipse Linker Configuration:** Add `libStdFFile.a` library:

1. Right-click project, select "Properties"
2. Navigate to "C/C++ Builds -> Settings"
3. Select "GNU C/C++ Linker -> Libraries"
4. Add "StdFFile" to Libraries list

12.43.14.1.8 Platform Support

- NetBurner embedded platforms
- Multiple storage device interfaces
- Real-time clock integration
- Timezone management with DST support

12.43.14.1.9 Error Codes and Troubleshooting

The application provides detailed error reporting for:

- File system operations (38 defined error codes)
- Network connectivity issues
- SSL/TLS handshake failures
- SMTP authentication problems
- DNS resolution failures

12.43.15 SSL/TLS Send Mail w/ Attachment

12.43.15.1 SSL/TLS SendMail with Attachment

12.43.15.1.1 Overview

This application demonstrates how to send emails with plain text attachments using a secure SSL/TLS connection. The application provides a web-based interface for composing and sending emails through an SMTP server with authentication.

12.43.15.1.2 Features

- Secure SSL/TLS email transmission
- Web-based user interface
- SMTP authentication support
- Plain text email attachments
- Real-time feedback and error reporting
- DNS resolution for SMTP servers

12.43.15.1.3 Requirements

- Network connection with [DHCP](#) support
- SSL/TLS capable SMTP server
- Compatible web browser for interface access

12.43.15.1.4 Application Components

12.43.15.1.4.1 main.cpp

The main application entry point that:

- Initializes the network stack
- Enables system diagnostics
- Starts the HTTPS web server on default port 443
- Waits for network connectivity via [DHCP](#)
- Runs the main application loop

12.43.15.1.4.2 webfuncs.cpp

Contains the web interface functionality including:

- Form data handling for email parameters
- POST request processing
- Email composition and transmission
- Error handling and status reporting
- MIME attachment support

12.43.15.1.4.3 Web Interface

The application provides form fields for:

- **Username:** SMTP authentication username
- **Password:** SMTP authentication password
- **Server:** SMTP server hostname or IP address
- **Port:** SMTP server port (default: 465 for SSL/TLS)
- **From:** Sender email address
- **To:** Recipient email address
- **Subject:** Email subject line
- **Body:** Email message content
- **Attachment:** Plain text attachment content

12.43.15.1.5 Configuration

12.43.15.1.5.1 SMTP Server Settings

- Default port: 465 (SSL/TLS)
- Authentication: Required
- Connection type: SSL/TLS (not STARTTLS)

12.43.15.1.5.2 Google Account Setup

When using Gmail or Google Workspace accounts:

1. Enable "Less secure app access" in account settings
2. For 2FA-enabled accounts, create an application password at: <https://myaccount.google.com/apppasswords>

12.43.15.1.6 Usage

1. **Network Setup:** Ensure the device has network connectivity and can obtain a [DHCP](#) address
2. **Access Interface:** Navigate to the device's HTTPS web interface
3. **Configure Email:** Fill in all required form fields
4. **Send Email:** Submit the form to transmit the email
5. **View Results:** Check the results page for success/failure status

12.43.15.1.7 Technical Details

12.43.15.1.7.1 Memory Allocation

- Username: 80 characters
- Password: 80 characters
- Server: 80 characters
- Server Port: 6 characters
- From/To addresses: 80 characters each
- Subject: 256 characters
- Body: 16,000 characters
- Attachment: 16,000 characters

12.43.15.1.7.2 Error Handling

The application provides detailed error reporting including:

- DNS resolution status
- SMTP server connection results
- Authentication success/failure
- Detailed server response logs

12.43.15.1.7.3 Security Considerations

- Uses SSL/TLS encryption for all SMTP communications
- Passwords are handled securely in memory
- Web interface operates over HTTPS

12.43.15.1.8 Limitations

- Supports plain text attachments only
- Single attachment per email
- Does not support STARTTLS (use dedicated SSL/TLS connection)
- No file system integration (attachment content entered via web form)

12.43.15.1.9 Troubleshooting

12.43.15.1.9.1 Common Issues

1. **DNS Resolution Failed:** Verify SMTP server hostname is correct
2. **Authentication Failed:** Check username/password and account settings
3. **Connection Timeout:** Verify server address and port number
4. **SSL/TLS Errors:** Ensure server supports SSL/TLS on specified port

12.43.15.1.9.2 Debug Information

Enable system diagnostics in the code to view detailed connection and transmission logs through the serial console.

12.43.15.1.10 Related Examples

For applications requiring file system integration, refer to the EDFS (Embedded Flash File System) SendMail examples that support file-based attachments.

12.43.16 SSL/TLS Send Mail w/ EDFS Attachment

12.43.16.1 SSL Send Mail with EDFS Attachment

12.43.16.1.1 Overview

This NetBurner application demonstrates how to send email messages with file attachments using SSL/TLS encryption and the EDFS (Embedded Flash File System) FAT32 file system. The application provides a web-based interface for configuring email settings and sending messages with attachments from SD/MMC cards or other supported storage devices.

12.43.16.1.2 Features

- **Secure Email Transmission:** Uses SSL/TLS encryption for secure SMTP connections
 - **Web-Based Interface:** Complete email configuration and sending through a web page
 - **File Attachments:** Attach files from the EDFS FAT32 file system
 - **Text Attachments:** Create text file attachments directly from web form input
 - **SMTP Authentication:** Supports authenticated SMTP connections
 - **FTP Server:** Built-in FTP server for file management on the storage device
 - **Multiple Storage Types:** Supports SD/MMC, SDHC, Compact Flash, and RAM drives
 - **Time Synchronization:** NTP time synchronization with manual fallback
- NBEclipse Linker Configuration:** Add `LibStdFFile`, a library:

1. Right-click project, select "Properties"
2. Navigate to "C/C++ Builds -> Settings"
3. Select "GNU C/C++ Linker -> Libraries"
4. Add "StdFFile" to Libraries list

12.43.16.1.3 Application Components

12.43.16.1.3.1 Core Modules

- **main.cpp:** Main application entry point and initialization
- **webfuncs.cpp:** Web interface functions for email configuration and sending
- **FileSystemUtils.cpp/h:** File system utilities for EDFS operations
- **effs_time.cpp/h:** Time management functions (NTP, manual, RTC)
- **ftp_f.cpp/h:** FTP server implementation for file system access
- **cardtype.h:** Storage device type configuration

12.43.16.1.3.2 Web Interface

- **index.html:** Main configuration page for email settings
- Web form includes fields for:
 - SMTP server settings (server, port, username, password)
 - Email details (from, to, subject, message body)
 - Attachment options (text creation, SD card file selection)

12.43.16.1.4 Getting Started

12.43.16.1.4.1 Hardware Requirements

1. NetBurner development board from supported platforms
2. SD/MMC card, SDHC card, or Compact Flash card (depending on configuration)
3. Network connection (Ethernet)
4. Storage device properly inserted and formatted as FAT32

12.43.16.1.4.2 Network Configuration

1. Configure IP address, gateway, and DNS settings
2. For [DHCP](#), set IP address to 0.0.0.0 using IPSetup
3. Ensure internet connectivity for NTP time synchronization

12.43.16.1.4.3 Email Server Configuration

The application uses SSL/TLS on port 465 by default. For Gmail:

1. Server: `smtp.gmail.com`
2. Port: 465
3. Enable "Less secure app access" or use an application-specific password
4. For two-factor authentication, generate an app password at: <https://myaccount.google.com/apppasswords>

12.43.16.1.4.4 Usage

1. **Power on and Initialize:** The device will automatically:
 - Initialize the network stack
 - Mount the storage device
 - Synchronize time via NTP
 - Start the web and FTP servers
2. **Access Web Interface:** Connect to the device's IP address via HTTPS
 - All connections must use HTTPS due to SSL configuration
3. **Configure Email Settings:** Fill out the web form with:
 - SMTP server credentials
 - Email addresses and content
 - Attachment options
4. **Send Email:** Click "Send Mail" to transmit the email with attachments

12.43.16.1.4.5 File Management

The application includes an FTP server for convenient file management:

- Upload files to the storage device via FTP
- View and organize files on the SD card
- Prepare files for email attachment

12.43.16.1.5 Storage Device Support

The application supports multiple storage types through compile-time configuration:

- **SD/MMC Cards:** Standard SD and MultiMedia cards
- **SDHC Cards:** High-capacity SD cards
- **Compact Flash:** CF cards with appropriate hardware interface
- **RAM Drive:** In-memory file system for testing

Storage type is configured in `cardtype.h` - only one type can be used at a time.

12.43.16.1.6 Security Features

- **SSL/TLS Encryption:** All SMTP communications are encrypted
- **HTTPS Web Interface:** Secure web-based configuration
- **Authentication Support:** SMTP username/password authentication
- **Application Passwords:** Support for modern email provider security

12.43.16.1.7 Error Handling

The application includes comprehensive error handling:

- EDFS file system error codes and descriptions
- Network timeout and retry mechanisms
- DNS resolution error reporting
- Detailed error logging for troubleshooting

12.43.16.1.8 Time Management

Accurate timestamps are maintained through:

1. **NTP Synchronization:** Primary time source from internet time servers
2. **Manual Time Setting:** Fallback when NTP is unavailable
3. **RTC Support:** Real-time clock integration where available
4. **Timezone Configuration:** Proper timezone handling for file timestamps

12.43.16.1.9 File System Operations

The EDFS utilities provide:

- File reading/writing operations
- Directory listing and navigation
- Space usage statistics
- File deletion and management
- Support for long filenames (platform dependent)

12.43.16.1.10 Development Notes

- The application uses the HCC Embedded EDFS FAT32 file system
- File system access requires proper task initialization with `f_enterFS()`
- Multiple tasks can access the file system simultaneously (up to 10)
- Storage devices must be properly formatted as FAT32
- Write protection detection and handling included

12.43.16.1.11 Troubleshooting

12.43.16.1.11.1 Common Issues

1. **Storage Device Not Detected:** Ensure card is properly inserted and supported
2. **Write Protection:** Remove write protection from SD card
3. **Email Send Failures:** Verify SMTP settings and network connectivity
4. **Time Sync Issues:** Check internet connectivity for NTP
5. **File Access Errors:** Ensure proper FAT32 formatting

12.43.16.1.11.2 Error Codes

The application provides detailed EDFS error codes for file system operations. Common errors include:

- `F_ERR_CARDREMOVED`: Storage device was removed
- `F_ERR_WRITEPROTECT`: Device is write-protected
- `F_ERR_NOTFORMATTED`: Device needs formatting
- `F_ERR_NOMOREENTRY`: Directory full

12.43.17 SSL/TLS Server

12.43.17.1 NetBurner SSL/TLS Server Example

12.43.17.1.1 Overview

This application demonstrates a simple SSL/TLS server implementation using the NetBurner RTOS platform. The server uses Elliptic Curve Cryptographic (ECC) keys to establish secure connections and serves as an example of how to implement SSL/TLS communication in embedded systems.

12.43.17.1.2 Features

- SSL/TLS server listening on port 8883 ([MQTT](#) secure port)
- Elliptic Curve Cryptography (ECC) support
- Client connection handling with secure data transmission
- Large data buffer transmission demonstration
- Real-time connection status reporting

12.43.17.1.3 Application Architecture

12.43.17.1.3.1 Main Components

1. **TcpServerTask:** The main server task that handles incoming SSL/TLS connections
2. **Certificate and Key Files:** Pre-generated ECC certificate and private key
3. **Network Interface Management:** Automatic network configuration and IP address handling

12.43.17.1.3.2 Key Files

- `main.cpp` - Main application logic and server implementation
- `ServerCert.cpp` - Auto-generated SSL certificate data (672 bytes)
- `ServerKey.cpp` - Auto-generated ECC private key data (310 bytes)

12.43.17.1.4 How It Works

1. **Initialization:** The application initializes the NetBurner system and waits for network connectivity
2. **Server Setup:** Creates a listening socket on port 8883 using SSL/TLS
3. **Client Handling:** Accepts incoming connections and establishes secure sessions
4. **Data Exchange:**
 - Sends a welcome message to connected clients
 - Transmits a large buffer containing pi digits (demonstration data)
 - Displays client IP address and port information
 - Echoes back any data received from clients
5. **Connection Management:** Properly closes connections and cleans up resources

12.43.17.1.5 Testing the Application

12.43.17.1.5.1 Using OpenSSL Command Line

You can test the SSL/TLS server using OpenSSL from a command line:

For ECC Cipher Suite

```
openssl s_client -cipher ECDHE-ECDSA-AES256-GCM-SHA384 -connect <ip_address>:8883
```

Example:

```
openssl s_client -cipher ECDHE-ECDSA-AES256-GCM-SHA384 -connect 10.1.1.191:8883
```

For RSA (if using RSA keys)

```
openssl s_client -connect <ip_address>:8883
```

Example:

```
openssl s_client -connect 10.1.1.191:8883
```

12.43.17.1.5.2 Expected Behavior

When a client connects successfully, you should see:

1. Welcome message: "Welcome to the NetBurner SSL/TLS Server Example"
2. A large buffer of pi digits (demonstration data)
3. Server IP address and port information
4. Echo of any data you send to the server

12.43.17.1.6 Configuration

12.43.17.1.6.1 Network Settings

- **Listen Port:** 8883 ([MQTT](#) secure port)
- **Buffer Size:** 4096 bytes for receiving data
- **Network Wait:** 5 seconds timeout for [DHCP](#) address acquisition

12.43.17.1.6.2 Task Configuration

- **Stack Size:** USER_TASK_STK_SIZE
- **Priority:** MAIN_PRIO - 1 (higher priority than main task)
- **Task Name:** "TcpServerTask"

12.43.17.1.7 Customization Options

12.43.17.1.7.1 Using RSA Keys Instead of ECC

If your application requires RSA keys instead of ECC:

1. Navigate to `\nburn\CreateCerts\RSA` directory
2. Run the key generation scripts
3. Replace `ServerKey.cpp` and `ServerCert.cpp` with the newly generated files
4. Rebuild the application

The RSA version can be tested with the standard OpenSSL command without specifying cipher suites.

12.43.17.1.8 Technical Details

12.43.17.1.8.1 SSL/TLS Implementation

- Uses NetBurner's built-in SSL/TLS stack
- Supports modern cipher suites including ECDHE-ECDSA-AES256-GCM-SHA384
- Automatic certificate and key loading from embedded data

12.43.17.1.8.2 Memory Usage

- RX Buffer: 4KB for incoming data
- Task Stack: Configurable via USER_TASK_STK_SIZE
- Certificate: 672 bytes
- Private Key: 310 bytes

12.43.17.1.8.3 Error Handling

- Connection timeout handling
- Graceful client disconnection
- Resource cleanup on connection termination

12.43.17.1.9 Security Considerations

- Uses strong ECC encryption by default
- Certificates are embedded in the application binary
- For production use, consider:
 - Using production certificates from a trusted CA
 - Implementing certificate validation
 - Adding additional security measures as required

12.43.17.1.10 Dependencies

- NetBurner RTOS
- NetBurner SSL/TLS library
- Standard NetBurner networking stack

12.43.17.1.11 Troubleshooting

12.43.17.1.11.1 Common Issues

1. **Connection Refused:** Ensure the device has a valid IP address and port 8883 is not blocked
2. **SSL Handshake Failure:** Verify that the client supports the cipher suites used by the server
3. **Network Issues:** Check [DHCP](#) configuration and network connectivity

12.43.18 HTTPS Configuration Mirror

12.43.18.1 SSL Configuration Mirror

12.43.18.1.1 Overview

This NetBurner embedded web server application demonstrates SSL/TLS security implementation with mixed HTTP/HTTPS access control. The application showcases how to create a web server that serves both secure and non-secure content while enforcing SSL requirements for protected resources.

SSL/TLS Server example program with web page redirection for unauthorized access. The example demonstrates how you can have both secure and non-secure access to files and directories.

The application starts the web server with SSL/TLS capability. The directory structure is such that index.html and the files in the images subdirectory can be viewed with a http or https connection. However, files in the httpsdir directory (ie repeat.html) can only be viewed with a secure https connection.

```
html
|-- index.html
|-- images
|   |-- (various image files)
|-- httpsdir
|   |-- repeat.html
```

The `HTTP_ACCESS CheckHttpAccess(int sock, int access_level, HTTP\_Request &Req)` function is used to authenticate the web page access. It will allow access or redirect depending on the file and access rights.

12.43.18.1.2 Application Features

12.43.18.1.2.1 Core Functionality

- **Dual Protocol Support:** Operates on both HTTP (port 80) and HTTPS (port 443)
- **Access Control:** Implements granular security policies for different content areas
- **SSL Redirection:** Automatically redirects HTTP requests to HTTPS for protected content
- **Configuration Management:** Provides secure configuration interface with JSON API endpoints
- **Post Data Logging:** Captures and stores HTTP POST requests with timestamps and payloads

12.43.18.1.2.2 Security Implementation

- **Certificate-based SSL/TLS:** Uses embedded server certificate and private key
- **Access Groups:** Three-tier authentication system (0=public, 1=SSL required, 2=password protected)
- **Automatic Redirection:** Non-secure requests to protected content trigger HTTPS redirects
- **Authentication:** HTTP Basic Auth for password-protected areas

12.43.18.1.3 Directory Structure

```
html/
|-- index.html           # Main page (accessible via HTTP/HTTPS)
|-- images/             # Public image assets
|   |-- NetBurnerSSLDemo.gif
|   |-- SSL-Good.gif     # Shown for HTTPS connections
|   |-- SSL-Bad.gif      # Shown for HTTP connections
|-- httpsdir/           # SSL-only protected directory
|   |-- repeat.html      # Requires HTTPS access
|-- Put.json            # Protected JSON endpoint (password required)
|-- robots.txt          # Search engine directives
```

12.43.18.1.4 Access Control Levels

12.43.18.1.4.1 Level 0 (Public Access)

- No authentication required
- Accessible via HTTP or HTTPS
- Examples: `index.html`, `images/` directory

12.43.18.1.4.2 Level 1 (SSL Required)

- Requires HTTPS connection
- HTTP requests receive 302 redirect to HTTPS
- Examples: `httpsdir/` content, configuration endpoints

12.43.18.1.4.3 Level 2 (Password Protected)

- Requires HTTPS + HTTP Basic Authentication
- Default credentials: `username="user"`, `password="password"`
- Examples: `Put.json`, administrative functions

12.43.18.1.5 Key Components

12.43.18.1.5.1 Main Application (`main.cpp`)

- **UserMain()**: Entry point that initializes network stack, SSL, and web server
- **CheckHttpAccess()**: Central access control function implementing security policies
- **SSL Helper Functions**: Utilities for SSL connection detection and certificate management
- **Post Handlers**: JSON POST request processing with data storage and retrieval

12.43.18.1.5.2 SSL Configuration (`HTTPSSConfigMirror.cpp`)

- **CheckSSL()**: Validates SSL connections and handles redirections
- **Configuration Handlers**: Secure endpoints for system configuration management
- **Schema Support**: JSON schema generation for configuration validation

12.43.18.1.5.3 Certificate Management

- **ServerCert.cpp**: Embedded SSL certificate data
- **ServerKey.cpp**: Embedded private key data
- Self-signed certificate valid for IP address access

12.43.18.1.6 API Endpoints

12.43.18.1.6.1 Configuration APIs (SSL Required)

- GET /Config/* - Retrieve configuration data as JSON
- POST /Config/* - Update configuration settings
- GET /Schema/* - Get JSON schema for configuration validation

12.43.18.1.6.2 Data APIs

- GET /GetJson* - Public JSON endpoint
- GET /PGetJson* - Password-protected JSON endpoint
- POST /Post* - Accept and log POST data
- GET /Show* - Display logged POST data as HTML table
- GET /JShow* - Display logged POST data as JSON

12.43.18.1.7 Build and Deployment

12.43.18.1.7.1 Prerequisites

- NetBurner development environment
- NNDK (NetBurner Network Development Kit)
- Target NetBurner device with network capability

12.43.18.1.7.2 Compilation

The application uses standard NetBurner build tools. Key dependencies:

- [nbrtos.h](#) - Real-time operating system
- [crypto/ssl.h](#) - SSL/TLS implementation
- [config_server.h](#) - Configuration management
- [httppost.h](#) - HTTP POST handling

12.43.18.1.7.3 Network Configuration

- [DHCP](#) client automatically obtains IP address
- NTP time synchronization from pool servers
- Pacific Time Zone configuration (PST8PDT)

12.43.18.1.8 Usage Examples

12.43.18.1.8.1 Basic Access

1. Connect to device IP via HTTP - shows "NOT secure" indicator
2. Access `httpsdir/repeat.html` via HTTP - receives FORBIDDEN response
3. Click "View index.html with HTTPS" - establishes secure connection
4. Access protected content successfully via HTTPS

12.43.18.1.8.2 Configuration Management

1. Navigate to <https://device-ip/Config/AppData> (requires SSL)
2. Authenticate with system credentials if configured
3. View/modify JSON configuration data
4. Changes automatically saved to persistent storage

12.43.18.1.8.3 POST Data Testing

1. Send JSON POST to `/Post*` endpoints
2. View logged data at `/Show*` for HTML table format
3. View logged data at `/JShow*` for JSON format
4. Access individual payloads via `/PlShow*` endpoints

12.43.18.1.9 Security Considerations

- Self-signed certificate generates browser warnings in production
- Default passwords should be changed for production deployment
- SSL implementation uses industry-standard encryption
- All configuration changes require secure HTTPS connection
- Automatic certificate validation for embedded applications

12.43.18.1.10 Technical Notes

- Uses NetBurner RTOS with cooperative multitasking
- Memory-efficient design suitable for embedded systems
- Real-time timestamp logging with timezone support
- Graceful handling of SSL handshake failures
- Configurable reboot capability after configuration changes

12.43.18.1.11 Development Features

- Diagnostic monitoring integration
- Build timestamp tracking
- Runtime statistics collection
- Comprehensive error handling and logging
- Extensible callback architecture for custom handlers

12.44 Stack Protection

12.44.1 Stack Protection Example

This application demonstrates various stack overflow and underflow protection methods for embedded systems using the NBRTOS real-time operating system.

12.44.1.1 Overview

The Stack Protection Example is designed to showcase how stack protection mechanisms work by intentionally triggering stack overflow conditions in a controlled manner. This educational tool helps developers understand stack protection features and their importance in embedded system development.

12.44.1.2 Features

- **Buffer Overflow Detection:** Demonstrates overflow caused by writing past buffer boundaries
- **Call Stack Overflow Detection:** Shows overflow through infinite recursion
- **Cross-Platform Support:** Works on both ARM Cortex-M7 and Coldfire processors
- **Real-time Stack Monitoring:** Provides live stack usage information
- **Interactive Menu System:** User-selectable overflow scenarios

12.44.1.3 Architecture Support

12.44.1.3.1 ARM Cortex-M7

- Supports monitoring of **both** stack overflow AND underflow simultaneously
- Can monitor 2 memory ranges concurrently
- Full stack protection capabilities enabled

12.44.1.3.2 Coldfire

- Limited to monitoring a **single** memory range
- Must choose between overflow OR underflow protection (not both)
- Typically configured for overflow protection

12.44.1.4 Prerequisites

12.44.1.4.1 Required Configuration

To enable full functionality, the following preprocessor definitions must be set:

```
#define NBRTOS_STACKCHECK (1)
#define NBRTOS_STACKOVERFLOW (1)
#define NBRTOS_STACKUNDERFLOW (1)
```

Note**: For Coldfire processors, only enable either NBRTOS_STACKOVERFLOW or NBRTOS_STACKUNDERFLOW, not both.

12.44.1.4.2 Compiler Settings

The application requires the `-fstack-check` compiler flag to be enabled:

- **NBEclipse IDE:** Right-click project > Properties > C/C++ Build > Settings > GNU C++ Compiler > Debugging > Enable "Enable stack checking (-fstack-check)"
- **Command Line:** The flag is automatically included in the provided makefile

12.44.1.4.3 File Configuration

The configuration is handled through the `predef-overload.h` file, which automatically sets appropriate definitions based on the target processor:

- **Coldfire:** Enables only stack overflow protection
- **Cortex-M7:** Enables both overflow and underflow protection

12.44.1.5 Application Structure

12.44.1.5.1 Main Components

1. **bufferOverflow()**: Creates a buffer overflow by writing beyond array boundaries
2. **callStackOverflow()**: Triggers overflow through recursive function calls with multiple parameters
3. **UserMain()**: Main application loop with interactive menu system

12.44.1.5.2 Menu Options

The application presents an interactive menu with the following options:

1. **Buffer Underflow** - Demonstrates buffer boundary violations
2. **Call Stack Overflow** - Shows recursive call stack exhaustion

12.44.1.6 Usage

1. **Build and Deploy**: Compile the application with the provided makefile or IDE project
2. **Run Application**: Execute on target hardware
3. **Monitor Output**: Observe stack range information displayed at startup
4. **Select Test**: Choose overflow scenario from the interactive menu
5. **Trigger Event**: Press any key when prompted to initiate the overflow condition
6. **Observe Results**: Watch as the stack protection mechanisms detect and respond to violations

12.44.1.7 Expected Output

Upon startup, the application displays:

- Current stack space range (top and bottom addresses)
- Available protection features status
- Interactive menu for selecting overflow scenarios
- Real-time feedback during overflow events

12.44.1.8 Safety Features

- **Stack Range Monitoring**: Displays current stack boundaries
- **Controlled Overflow**: Intentional violations for educational purposes
- **System Diagnostics**: Built-in diagnostic capabilities
- **Protection Mechanisms**: Hardware-assisted stack monitoring

12.44.1.9 Development Notes

12.44.1.9.1 Stack Monitoring

The application uses `OSDumpTCBStacks()` to display current stack information and `EnableOSStack↵ Protector()` to activate protection mechanisms.

12.44.1.9.2 Memory Layout

Stack boundaries are defined by:

- `OSTCBStkTop`: Top of stack address
- `OSTCBStkBot`: Bottom of stack address

12.44.1.9.3 Time Delays

Strategic delays (`OSTimeDly(1)`) are inserted to allow observation of the overflow progression.

12.44.1.10 Important Warnings

This application is designed for **educational and testing purposes only**. It intentionally creates stack overflow conditions that would be catastrophic in production code. Do not use these techniques in production systems.

12.44.1.11 Build Instructions

12.44.1.11.1 Command Line

```
make clean
make all
```

12.44.1.11.2 IDE

Import the project into NBEclipse and build using the standard build process to rebuild libraries.

12.44.1.12 Troubleshooting

If stack protection features don't work as expected:

1. Verify preprocessor definitions are correctly set
2. Ensure `-fstack-check` compiler flag is enabled
3. Confirm target processor type matches configuration
4. Check that NBRTOS stack protection is supported on your platform

12.45 Syslog

12.45.1 NetBurner Syslog Demo

12.45.1.1 Overview

This application demonstrates how to use the NetBurner Syslog utility to send logging information to a destination host computer via UDP on port 514. The demo sends a simple counting variable that increments every second to show elapsed time.

12.45.1.2 Features

- Network stack initialization with [DHCP](#) support
- Built-in web server on port 80
- Syslog messaging via UDP
- Support for both broadcast and targeted syslog transmission
- Real-time counting demonstration

12.45.1.3 How It Works

The application initializes the NetBurner network stack, waits for [DHCP](#) configuration to complete, and then begins sending syslog messages every second. Each message contains an incrementing counter showing elapsed seconds since startup.

12.45.1.3.1 Syslog Modes

The application supports two syslog transmission modes:

1. **Broadcast Mode:** When `SysLogAddress` is not set, syslog data is sent as UDP broadcast to 255.255.255.255:514
2. **Targeted Mode:** When `SysLogAddress` is configured, syslog data is sent to the specified IP address on port 514

12.45.1.4 Setup Instructions

12.45.1.4.1 Prerequisites

- NetBurner device with network connectivity
- Properly configured IP address and subnet mask
- NetBurner "UDP Terminal Tool" application (for receiving syslog messages)

12.45.1.4.2 Configuration Steps

1. Verify Network Connection

- Ensure your NetBurner device has proper network communication
- Verify the device has obtained an IP address and subnet mask

2. Configure Target Address (Optional)

- In `main.cpp`, locate the line: `SysLogAddress =AsciiToIp ("10.1.1.193") ;`
- Replace "10.1.1.193" with your computer's IP address
- Comment out this line to use broadcast mode instead

3. Setup Syslog Receiver

- Run the NetBurner "UDP Terminal Tool" application
- Set the local listening port to 514 (default syslog port)

4. Deploy Application

- Download and run the application on your NetBurner device

12.45.1.5 Code Structure

12.45.1.5.1 Main Components

- **Network Initialization:** Sets up network stack and waits for [DHCP](#) completion
- **System Diagnostics:** Enables diagnostic features (should be removed for production)
- **Web Server:** Starts HTTP server on default port 80
- **Syslog Loop:** Continuously sends elapsed time messages every second

12.45.1.5.2 Key Functions

- `init ()` : Initializes the network stack
- `StartHttp ()` : Starts the web server
- `WaitForActiveNetwork ()` : Waits for network connectivity
- `showIpAddresses ()` : Displays current IP configuration
- `SysLog ()` : Sends formatted syslog messages

12.45.1.6 Expected Output

Once running, the application will:

1. Display startup messages on the console
2. Show IP address information
3. Indicate whether syslog is running in broadcast or targeted mode
4. Begin sending messages like: "Seconds elapsed: 0", "Seconds elapsed: 1", etc.

12.45.1.7 Network Requirements

- **Protocol:** UDP
- **Port:** 514 (standard syslog port)
- **Default Target:** 255.255.255.255 (broadcast) or configured IP address

12.45.1.8 Production Considerations

- Remove `EnableSystemDiagnostics()` call for production deployments
- Consider implementing proper error handling for network failures
- Adjust timing intervals based on logging requirements
- Implement log rotation or filtering for long-running applications

12.45.1.9 Troubleshooting

- **No syslog messages received:** Verify firewall settings and UDP port 514 access
- **DHCP issues:** Check network configuration and DHCP server availability
- **Wrong target address:** Verify the configured IP address in `SysLogAddress`
- **Network connectivity:** Ensure proper physical network connection and IP configuration

12.46 System Diagnostics

12.46.1 System Diagnostics Example

12.46.1.1 Overview

This application demonstrates how to integrate system diagnostics into a NetBurner device application. The program enables a web-based diagnostic interface that displays both system information and custom application variables through the device's Configuration Server web page.

12.46.1.2 Features

- **System Diagnostics:** Enables built-in system diagnostic information display
- **Custom Variable Monitoring:** Demonstrates how to add application-specific variables to the diagnostic web page
- **Web Interface:** Provides access to diagnostics through HTTP server on port 80
- **Real-time Updates:** Variables are updated in real-time and can be monitored through the web interface

12.46.1.3 Application Components

12.46.1.3.1 Main Functionality

The application performs the following operations:

1. Initializes the network stack
2. Enables system diagnostics functionality
3. Starts the HTTP web server on default port 80
4. Waits for network connectivity
5. Continuously increments a counter variable every second

12.46.1.3.2 Monitored Variables

The application includes two example variables that are displayed on the diagnostic web page:

- **myInt**: An integer counter that increments every second
- **myString**: A string variable initialized with "My String Variable"

12.46.1.4 Code Structure

12.46.1.4.1 Key Components

- **Network Initialization**: Standard NetBurner network stack setup
- **Diagnostic Integration**: Uses `EnableSystemDiagnostics()` to activate the diagnostic interface
- **Variable Monitoring**: Implements `DiagVarMon` and `DiagStrMon` for custom variable display
- **Web Server**: HTTP server provides access to the diagnostic interface

12.46.1.4.2 Header Dependencies

```
#include <init.h>
#include <nbrtos.h>
#include <system.h>
#include <string.h>
#include <diagnostics.h>
```

12.46.1.5 Usage Instructions

12.46.1.5.1 Building and Running

1. Compile the application using the NetBurner development tools
2. Deploy to your NetBurner device
3. Ensure the device is connected to your network
4. The application will display network information on the console

12.46.1.5.2 Accessing Diagnostics

1. Open a web browser
2. Navigate to your device's IP address (port 80)
3. Access the Configuration Server web page
4. Click the "Diagnostics" button to view system and application variables

12.46.1.6 Adding Custom Variables

To add your own variables to the diagnostic display:

12.46.1.6.1 For Integer Variables

```
int yourVariable;  
static DiagVarMon YourVarMon("Variable Description", yourVariable);
```

12.46.1.6.2 For String Variables

```
char yourString[SIZE];  
static DiagStrMon YourStrMon("String Description", yourString);
```

12.46.1.6.3 Supported Variable Types

- Integers ([DiagVarMon](#))
- Strings ([DiagStrMon](#))
- Floats (using appropriate diagnostic monitor classes)
- Other data types as supported by the diagnostics library

12.46.1.7 Important Notes

12.46.1.7.1 Production Considerations

- The `EnableSystemDiagnostics()` function should likely be removed or conditionally compiled for production code
- Consider security implications of exposing diagnostic information in production environments
- The web server runs on port 80 by default and may conflict with other web services

12.46.1.7.2 Network Requirements

- Device must be connected to an active network
- The application waits up to 5 seconds for network connectivity before proceeding
- HTTP server accessibility depends on network configuration and firewall settings

12.46.1.8 Output Example

When running, the application displays:

```
Web Application: System Diagnostics Example  
NNDK Revision: [Current NetBurner SDK Version]
```

The web diagnostic interface will show:

- Standard system diagnostic information
- "My Int": Incrementing counter value
- "My String": "My String Variable"

12.47 TCP

TCP Examples:

- [TCP Client](#)
- [TCP No Block Connect Test](#)
- [TCP Keepalive Simple](#)
- [TCP Keepalive Multiple](#)
- [Multiple TCP Interface Test](#)
- [TCP Multi-Socket Server](#)

- [TCP Resource Information](#)
- [TCP Server](#)
- [TCP Server Using Via](#)
- [TCP Speed Test](#)
- [TCP Stress Test](#)

12.47.1 TCP Client

12.47.1.1 TCP Client Example Application

12.47.1.1.1 Overview

This is a simple TCP client application built for NetBurner hardware that demonstrates how to create and use a TCP client connection. The application provides a web-based interface that allows users to send messages to any TCP server by specifying the destination IP address, port number, and message content.

12.47.1.1.2 Features

- **Web-based Interface:** Complete interaction through a web page interface
- **Configurable Connection:** Set destination IP address and port number
- **Message Transmission:** Send custom messages to TCP servers
- **Auto-fill Functionality:** Automatically fills in the requesting client's IP address as default destination
- **Error Handling:** Comprehensive error reporting and logging
- **Real-time Feedback:** Connection status and transmission results displayed

12.47.1.1.3 Application Structure

12.47.1.1.3.1 Core Files

- **main.cpp:** Main application entry point and initialization
- **clientweb.cpp:** Web interface implementation and TCP client functionality
- **clientweb.h:** Header file for web interface functions

12.47.1.1.3.2 Key Components

Main Application (main.cpp)

- Initializes the NetBurner system
- Enables system diagnostics
- Starts HTTP server for web interface
- Waits for active network connection
- Displays IP addresses
- Runs main application loop

Web Interface (clientweb.cpp)

- Implements web page form handling
- Processes POST variables from web form
- Manages TCP client connections
- Handles message transmission
- Provides real-time status feedback

12.47.1.1.4 How It Works

1. **Initialization:** The application starts up, initializes the network, and launches the HTTP server
2. **Web Interface:** Users access the web page which displays form fields for:
 - Destination IP address (auto-filled with requesting client's IP)
 - Destination port number (defaults to 2000)
 - Message content
3. **Message Transmission:** When the form is submitted:
 - A TCP connection is established to the specified server
 - The message is transmitted
 - Connection status and results are logged
 - The connection is closed
4. **Response:** The web page redirects back to the main interface

12.47.1.1.5 Configuration

12.47.1.1.5.1 Default Settings

- **Default Port:** 2000
- **Connection Timeout:** 5 seconds
- **Maximum Message Size:** 1024 characters

12.47.1.1.5.2 Web Form Fields

- `tfDestIpAddr`: Destination IP address field
- `tfDestPortNum`: Destination port number field
- `tfMessage`: Message content field
- `SendMessage`: Submit button trigger

12.47.1.1.6 Usage Instructions

1. **Setup:** Deploy the application to NetBurner hardware
2. **Network:** Ensure the device is connected to your network
3. **Access:** Open a web browser and navigate to the device's IP address
4. **Configure:**
 - Enter the destination TCP server IP address
 - Set the destination port number
 - Type your message
5. **Send:** Click the submit button to transmit the message
6. **Monitor:** Check the console output (MTTTY) for connection status and errors

12.47.1.1.7 Testing

12.47.1.1.7.1 TCP Server Requirement

A TCP server must be running and listening on the specified IP address and port before sending messages.

12.47.1.1.7.2 Included Test Server

NetBurner provides a Windows-based TCP server for testing:

- **Location:** \nburn\pctools\TCP\TcpServerWin
- **Includes:** Executable and source code
- **Purpose:** Simple TCP server for testing client connections

12.47.1.1.8 Technical Details

12.47.1.1.8.1 Network Functions Used

- `connect()` : Establishes TCP connection with timeout
- `write()` : Sends data over TCP connection
- `close()` : Closes TCP connection
- `GetSocketRemoteAddr()` : Gets client IP address

12.47.1.1.8.2 Error Handling

- Connection failure detection and reporting
- Invalid parameter validation
- Timeout handling
- Detailed error logging to console

12.47.1.1.8.3 Memory Management

- Static buffers for message storage
- Automatic cleanup on connection close
- Bounds checking for message size

12.47.1.1.9 Troubleshooting

12.47.1.1.9.1 Common Issues

1. **Connection Failed:** Ensure TCP server is running and accessible
2. **Network Not Ready:** Wait for network initialization to complete
3. **Invalid IP Address:** Verify IP address format and reachability
4. **Port Conflicts:** Ensure destination port is not blocked or in use

12.47.1.1.9.2 Debug Output

All debug information is sent to stdout and can be viewed using MTTY terminal application.

12.47.2 TCP No Block Connect Test

12.47.2.1 TCP No Block Connect Test

12.47.2.1.1 Overview

This application demonstrates the difference between blocking and non-blocking TCP connections using NetBurner embedded networking hardware. The program provides an interactive menu system that allows users to test both standard blocking `connect()` calls and non-blocking `NoBlockConnect()` calls.

12.47.2.1.2 Purpose

The standard TCP connect call will block until the connection is either made or fails. The non-blocking connect call will initiate a TCP connection and return immediately, allowing the application to continue processing while the connection is established in the background.

12.47.2.1.3 Features

- Interactive menu-driven interface
- Support for both blocking and non-blocking TCP connections
- Real-time connection state monitoring
- Automatic periodic data transmission (every 5 seconds)
- User input forwarding over established connections
- Configurable destination IP address and port

12.47.2.1.4 Menu Options

The application provides the following commands:

- **C** - Make a connection using standard blocking `connect()`
- **N** - Make a connection using non-blocking `NoBlockConnect()`
- **X** - Close the current connection
- **I** - Specify the destination IP address and port number
- **Any other character** - Send the character over the TCP connection (if established)

12.47.2.1.5 How It Works

12.47.2.1.5.1 Blocking Connection (C command)

1. Prompts for destination IP and port if not already set
2. Calls `connect()` which blocks until connection succeeds or fails
3. Immediately reports success or failure
4. Connection is ready for data transmission upon success

12.47.2.1.5.2 Non-Blocking Connection (N command)

1. Prompts for destination IP and port if not already set
2. Calls `NoBlockConnect()` which returns immediately
3. Reports "Connect started" if the call succeeds
4. Continuously monitors connection state using `TcpGetSocketState()`
5. Reports "NoBlock connection is done" when `TCP_STATE_ESTABLISHED` is reached

12.47.2.1.5.3 Automatic Features

- Sends a timestamp message every 5 seconds over established connections
- Displays current TCP state during non-blocking connection attempts
- Forwards any typed characters to the remote host when connected

12.47.2.1.6 Technical Details

- **Key APIs Used:**
 - `connect()` - Standard blocking TCP connection
 - `NoBlockConnect()` - Non-blocking TCP connection initiation
 - `TcpGetSocketState()` - Connection state monitoring
 - Standard socket operations (`write()`, `close()`)

12.47.2.1.7 Build Requirements

- NetBurner development environment
- NetBurner hardware platform
- Standard NetBurner libraries (`tcp.h`, `utils.h`, `iosys.h`, etc.)

12.47.2.1.8 Usage Example

1. Run the application on NetBurner hardware
2. When prompted, enter the destination IP address
3. Enter the destination port number
4. Use menu commands to test different connection methods:
 - Try 'C' for blocking connection
 - Try 'N' for non-blocking connection
 - Use 'X' to close connections
 - Use 'I' to change destination settings
5. Type any characters to send data over established connections

12.47.2.1.9 Connection State Monitoring

For non-blocking connections, the application continuously displays the TCP state:

- State values correspond to standard TCP states
- Connection is ready when `TCP_STATE_ESTABLISHED` is reached
- Failed connections will show appropriate error states

12.47.2.1.10 Notes

- Only one connection can be active at a time
- IP address and port cannot be changed while connected
- System diagnostics are enabled for debugging (should be disabled in production)
- The application runs in an infinite loop until manually terminated

12.47.3 TCP Keepalive Simple

12.47.3.1 TCP Keep-Alive Server

12.47.3.1.1 Overview

This NetBurner application demonstrates a TCP server implementation with keep-alive functionality. The server listens for incoming TCP connections and uses keep-alive packets to detect when clients become unresponsive or disconnect unexpectedly.

Keep-alive is implemented with the `TcpGetLastRxTime()` and `TcpSendKeepAlive()` functions. `TcpGetLastRxTime()` returns the number of time ticks since the last time a packet was received. `TcpSendKeepAlive()` sends a keep-alive packet to the client, which is used when no data has been transmitted within the timeout period.

The general concept is to call `TcpGetLastRxTime()`, call `TcpSendKeepAlive()`, wait a bit, then call `TcpSendKeepAlive()` a second time and verify the number of time ticks is different. If not, no packets were received and the client is not responding. Make sure to allow time for the client to respond to the keep alive packet. Do not call `TcpGetLastRxTime` more often than once every second to avoid performance issues.

12.47.3.1.2 Features

- TCP server listening on port 23 (Telnet)
- Automatic keep-alive packet transmission
- Client connection monitoring and detection of unresponsive clients
- Connection cleanup for dead connections
- Debug output to console/serial port

12.47.3.1.3 How It Works

12.47.3.1.3.1 Keep-Alive Implementation

The keep-alive functionality is implemented using two key NetBurner functions:

- `TcpGetLastRxTime()` - Returns the number of time ticks since the last received packet
- `TcpSendKeepAlive()` - Sends a keep-alive packet to the client

12.47.3.1.3.2 Keep-Alive Process

The example creates a TCP server task that listens on port 23 by default. The server blocks with "ReadWithTimeOut" until data is received or the timer times out. If a timeout occurs the time of the last received packet is recorded and a keep-alive packet is sent. Then, the server goes back to waiting for a read again. If ReadWithTimeOut times out a second time, the value of the last received TCP packet is checked again. If the client is still active the value of lastRxTime will be different than the previous value (due to the keep-alive packet that was sent). If the client did not respond to the keep-alive packet the number will remain the same and the client is assumed to be non-responsive, and the connection is closed.

1. **Initial Read:** Server waits for incoming data using `ReadWithTimeout()` with a 1-second timeout
2. **First Timeout:** If no data is received within the timeout period:
 - Record the timestamp of the last received packet
 - Send a keep-alive packet to the client
 - Continue waiting for data
3. **Second Timeout:** If timeout occurs again:
 - Check if the last received packet timestamp has changed
 - If timestamp is different: Client responded to keep-alive (connection is alive)
 - If timestamp is unchanged: Client is unresponsive, close the connection

12.47.3.1.3.3 Connection Flow

Client Connects -> Welcome Message -> Data Loop -> Keep-Alive Check -> Connection Management

12.47.3.1.4 Configuration

12.47.3.1.4.1 Default Settings

- **Listen Port:** 23 (Telnet)
- **Read Timeout:** 1 second
- **Buffer Size:** 4096 bytes
- **Maximum Pending Connections:** 5

12.47.3.1.4.2 Customization

To modify the listening port, change the `TCP_LISTEN_PORT` constant:

```
#define TCP_LISTEN_PORT 23 // Change to desired port number
```

12.47.3.1.5 Testing the Application

12.47.3.1.5.1 Using Telnet

From a Windows command prompt:

```
telnet <NetBurner_IP_Address>
```

From Linux/Mac terminal:

```
telnet <NetBurner_IP_Address> 23
```

12.47.3.1.5.2 Using PuTTY

1. Open PuTTY
2. Select "Raw" or "Telnet" connection type
3. Enter the NetBurner device IP address
4. Set port to 23
5. Click "Open"

12.47.3.1.5.3 Expected Behavior

1. Upon connection, you'll see: "Welcome to the NetBurner TCP Server"
2. The server will display your connection information
3. Any data you type will be echoed back with byte count information
4. If you stop sending data, keep-alive packets will be sent automatically
5. Status messages appear on the NetBurner's debug/console serial port

12.47.3.1.6 Code Structure

12.47.3.1.6.1 Main Components

- **UserMain():** Initializes the system and creates the TCP server task
- **TcpServerTask():** Main server loop that handles client connections
- **Keep-Alive Logic:** Integrated within the data reading loop

12.47.3.1.6.2 Key Variables

- `fdListen`: Listening socket file descriptor
- `fdNet`: Client connection file descriptor
- `LastRxTime`: Timestamp of last received packet
- `keepAliveSent`: Flag to track keep-alive state
- `RXBuffer`: Buffer for incoming data

12.47.3.1.7 Performance Considerations

- Do not call `TcpGetLastRxTime()` more frequently than once per second to avoid performance issues
- Allow sufficient time for clients to respond to keep-alive packets before determining they are unresponsive
- The server can handle multiple sequential connections but processes them one at a time

12.47.3.1.8 Debug Output

The application provides detailed debug information including:

- Connection establishment notifications
- Data reception logs with byte counts
- Keep-alive packet transmission notifications
- Client response status
- Connection closure notifications

All debug output is sent to the NetBurner's debug/console serial port.

12.47.4 TCP Keepalive Multiple

12.47.4.1 TCP Multi-Socket Server with Keep-Alive

A robust TCP server implementation that handles multiple concurrent connections with automatic dead connection detection using TCP keep-alive mechanisms.

12.47.4.1.1 Overview

This application demonstrates a production-ready TCP server that can:

- Handle up to 10 simultaneous TCP connections
- Automatically detect and close dead/disconnected clients
- Implement TCP keep-alive to prevent resource waste from zombie connections
- Use efficient I/O multiplexing with `select()` for scalable connection management

12.47.4.1.2 Features

12.47.4.1.2.1 Multi-Socket Connection Handling

- **Concurrent Connections:** Supports up to 10 simultaneous TCP connections
- **Non-blocking I/O:** Uses `select()` for efficient multiplexing of multiple sockets
- **Dynamic Connection Management:** Automatically accepts new connections and manages connection slots

12.47.4.1.2.2 Keep-Alive Mechanism

- **Idle Detection:** Monitors connections for inactivity (default: 5 seconds)
- **Automatic Probing:** Sends keep-alive packets to idle connections
- **Dead Connection Cleanup:** Closes connections that don't respond to keep-alive probes (timeout: 3 seconds)
- **Resource Protection:** Prevents server resource exhaustion from zombie connections

12.47.4.1.2.3 Connection Monitoring

- **Real-time Status:** Displays connection events, data transfers, and keep-alive activities
- **Client Information:** Shows client IP addresses and port numbers
- **Debug Output:** Comprehensive logging of connection lifecycle events

12.47.4.1.3 Configuration

12.47.4.1.3.1 Server Settings

```
#define listenPort (23)           // TCP port (Telnet)
#define maxConnections (10)      // Max simultaneous connections
#define readBufferSize (1024)   // Read buffer size per connection
```

12.47.4.1.3.2 Keep-Alive Timers

```
#define keepAliveTimeout (5 * TICKS_PER_SECOND) // 5 seconds idle timeout
#define keepAliveResponseTimeout (3 * TICKS_PER_SECOND) // 3 seconds response timeout
```

12.47.4.1.4 Usage

12.47.4.1.4.1 Starting the Server

1. Compile and deploy the application to your NetBurner device
2. The server automatically starts listening on port 23 (Telnet)
3. Monitor the console for connection status and debug information

12.47.4.1.4.2 Connecting Clients

```
# Connect using telnet
telnet <device_ip> 23

# Or using netcat
nc <device_ip> 23
```

12.47.4.1.4.3 Client Commands

- **Send any data:** Resets the keep-alive timer and echoes confirmation
- **Send 'Q' or 'q':** Gracefully disconnects from the server
- **Stay idle:** Triggers keep-alive mechanism after 5 seconds

12.47.4.1.5 How It Works

12.47.4.1.5.1 Connection Lifecycle

1. **Accept:** New connections are accepted and assigned to available slots
2. **Monitor:** Each connection is monitored for data activity using `select()`
3. **Keep-Alive:** Idle connections receive keep-alive probes
4. **Cleanup:** Dead or disconnected clients are automatically removed

12.47.4.1.5.2 Keep-Alive Process

1. **Idle Detection:** Server tracks last received data timestamp for each connection
2. **Probe Transmission:** After 5 seconds of inactivity, a TCP keep-alive packet is sent
3. **Response Monitoring:** Server waits up to 3 seconds for any TCP-level response
4. **Connection Termination:** Connections that don't respond are closed and removed

12.47.4.1.5.3 TCP Stack Integration

- Uses `TcpGetLastRxTime()` to track TCP-level packet reception
- Calls `TcpSendKeepAlive()` to send keep-alive probes
- Monitors both application data and TCP acknowledgments

12.47.4.1.6 Testing Scenarios

12.47.4.1.6.1 Normal Operation

```
# Connect and send data
telnet <device_ip> 23
> hello
Data received
> Q
Goodbye!
```

12.47.4.1.6.2 Keep-Alive Testing

1. Connect to the server but don't send any data
2. After 5 seconds, observe keep-alive probe message in server console
3. Continue to stay idle for 3+ more seconds
4. Connection will be automatically closed

12.47.4.1.6.3 Dead Connection Testing

1. Connect to the server
2. Physically disconnect the network cable or crash the client
3. After 8 seconds total (5 + 3), the server will detect and close the dead connection

12.47.4.1.7 Error Handling

- **Socket Errors:** Automatically detected and connections are closed
- **Accept Failures:** Gracefully handled with appropriate error messages
- **Buffer Overflows:** Protected by fixed buffer sizes
- **Resource Limits:** New connections rejected when server is full

12.47.4.1.8 Use Cases

12.47.4.1.8.1 Production Applications

- **Long-lived Connections:** Services that maintain persistent client connections
- **Resource Management:** Systems with limited connection resources
- **Network Monitoring:** Applications requiring dead connection detection
- **Embedded Systems:** Resource-constrained environments needing efficient connection handling

12.47.4.1.8.2 Network Scenarios

- **Unreliable Networks:** Environments with frequent network interruptions
- **Mobile Clients:** Connections that may drop without proper closure
- **Firewall Traversal:** Maintaining connections through NAT/firewall devices
- **Load Balancing:** Backend services requiring connection health monitoring

12.47.4.1.9 Performance Considerations

- **Memory Usage:** Each connection uses approximately 1KB for read buffers plus keep-alive tracking
- **CPU Overhead:** Minimal - keep-alive checks only run once per second
- **Network Traffic:** Keep-alive probes are small TCP packets sent only when needed
- **Scalability:** Can be extended to support more connections by increasing `maxConnections`

12.47.4.1.10 Dependencies

- NetBurner RTOS and TCP/IP stack
- Standard NetBurner includes: `init.h`, `iosys.h`, `tcp.h`, `ipshow.h`, `nbrtos.h`
- TCP keep-alive functions: `TcpGetLastRxTime()`, `TcpSendKeepAlive()`

12.47.4.1.11 Customization

12.47.4.1.11.1 Timeout Values

Adjust the keep-alive timing constants for your specific application requirements:

- Shorter timeouts for faster dead connection detection
- Longer timeouts for reduced network overhead

12.47.4.1.11.2 Connection Limits

Modify `maxConnections` based on available system resources and application needs.

12.47.4.1.12 Troubleshooting

12.47.4.1.12.1 Common Issues

- ****"Server Full" message**:** All connection slots are occupied
- **Immediate disconnections:** Check network connectivity and firewall settings
- **Keep-alive not working:** Verify TCP stack support and timing configuration

12.47.4.1.12.2 Debug Output

The application provides detailed console output for monitoring:

- Connection establishment and termination
- Keep-alive probe transmission and responses
- Error conditions and socket states

12.47.5 Multiple TCP Interface Test

12.47.5.1 TCP Multiple Interface Test

12.47.5.1.1 Overview

This NetBurner application demonstrates TCP client functionality with support for multiple network interfaces. The application provides a web-based interface for testing TCP connections to remote servers using various connection methods and network interfaces.

12.47.5.1.2 Application Description

The TCP Multiple Interface Test is a comprehensive example that showcases different ways to establish TCP connections from a NetBurner device. It consists of two main components:

1. **Web Interface** (`clientweb.cpp`)** - Provides an HTML form for configuring and initiating TCP connections
2. **Main Application** (`main.cpp`)** - Initializes the system and starts the HTTP server

12.47.5.1.3 Features

12.47.5.1.3.1 Connection Testing Methods

The application tests multiple TCP connection approaches:

- **Standard Connection** - Basic TCP connection using `connect ()`
- **Non-blocking Connection** - Asynchronous connection using `NoBlockConnect ()`
- **Interface-specific Connections** - Connections via specific network interfaces
- **IPv4 and IPv6 Support** - Dual-stack networking capabilities

12.47.5.1.3.2 Web Interface

The web interface allows users to:

- Specify destination IP address (auto-populated with client's IP)
- Set destination port number (defaults to 2000)
- Initiate connection tests through a web form
- View connection results in the console output

12.47.5.1.3.3 Key Functions

SendMsg(IPADDR destIp, int destPort)

Main function that performs comprehensive connection testing by:

- Testing standard and non-blocking connections
- Iterating through all available network interfaces
- Testing both IPv4 and IPv6 connections (when available)
- Attempting connections via interface number and interface IP

TestConnect(int fd, const char *msg)

Helper function that:

- Validates connection file descriptors
- Waits for connection establishment (2-second timeout)
- Reports connection status and local interface information
- Properly closes connections after testing

Web Callback Functions

- `WebDestPort()` - Displays current destination port in web form
- `WebDestIp()` - Displays current destination IP in web form
- `WebShowClientIp()` - Shows the client's IP address
- `PostCallBack()` - Processes form submissions and initiates connections

12.47.5.1.4 Usage

12.47.5.1.4.1 Setup

1. Compile and deploy the application to NetBurner hardware
2. Connect the device to your network
3. Note the device's IP address from the console output

12.47.5.1.4.2 Testing Connections

1. Open a web browser and navigate to the device's IP address
2. The form will auto-populate with your client IP as the destination
3. Modify the destination IP and port as needed
4. Click submit to initiate connection tests
5. Monitor console output (via MTTTY) for detailed connection results

12.47.5.1.4.3 Prerequisites

- A TCP server must be running at the specified destination IP and port
- NetBurner provides `TcpServerWin.exe` in the `\nburn\pctools` directory for testing

12.47.5.1.5 Network Interface Testing

The application systematically tests connections through:

12.47.5.1.5.1 IPv4 Interfaces

- Direct interface number specification
- Interface IP address specification
- Both standard and non-blocking variants

12.47.5.1.5.2 IPv6 Interfaces (when enabled)

- Interface number specification
- All valid IPv6 prefixes on each interface
- Standard and non-blocking connection methods

12.47.5.1.6 Console Output

Connection attempts generate detailed logging including:

- Connection method being tested
- Source IP address and interface number
- Success/failure status with error codes
- Timeout notifications for failed connections

12.47.5.1.7 Error Handling

The application includes error handling for:

- Invalid connection parameters
- Connection timeouts (2-second limit)
- Network interface enumeration failures
- Form processing errors

12.47.5.1.8 Configuration

12.47.5.1.8.1 Default Values

- Default destination port: 2000
- Connection timeout: 5 seconds
- Status check timeout: 2 seconds

12.47.5.1.8.2 Compilation Options

- IPv6 support can be enabled/disabled via `#ifdef IPV6`
- System diagnostics are enabled by default (consider disabling for production)

12.47.5.1.9 Development Notes

This example serves as a comprehensive reference for:

- NetBurner TCP client implementation
- Multi-interface network programming
- Web-based device configuration
- IPv4/IPv6 dual-stack networking
- Non-blocking socket operations

12.47.6 TCP Multi-Socket Server

12.47.6.1 TCP Multi-Socket Server

12.47.6.1.1 Overview

This application demonstrates a TCP server implementation that can handle multiple simultaneous client connections using the NetBurner embedded platform. The server uses the `select()` function to efficiently manage multiple TCP connections concurrently without blocking.

12.47.6.1.2 Features

- **Multi-client support:** Handles up to 10 simultaneous TCP connections
- **Non-blocking I/O:** Uses `select ()` to efficiently manage multiple file descriptors
- **Connection management:** Automatically accepts new connections and manages client disconnections
- **Simple protocol:** Clients can send 'Q' to quit their connection
- **Debug output:** Status messages displayed on debug serial port

12.47.6.1.3 Configuration

The application uses the following configurable parameters:

- **Listen Port:** 23 (Telnet port)
- **Maximum Connections:** 10 simultaneous clients
- **Read Buffer Size:** 1024 bytes per connection

These values can be modified by changing the corresponding `#define` statements in `main.cpp`.

12.47.6.1.4 How It Works

12.47.6.1.4.1 Server Initialization

1. Initializes the NetBurner system and waits for network connectivity
2. Creates a listening socket on port 23
3. Displays the device's IP addresses for client connection reference

12.47.6.1.4.2 Connection Management

The server uses a file descriptor array (`fdArray`) to track active client connections. The main loop:

1. **Sets up file descriptor sets** for `select ()` monitoring
2. **Calls `select()`** to block until activity occurs on any socket
3. **Processes new connections** when clients connect to the listening port
4. **Handles client data** when existing connections have data to read
5. **Manages errors** by closing problematic connections

12.47.6.1.4.3 Client Interaction

- **Welcome message:** New clients receive a greeting upon connection
- **Quit command:** Clients can send 'Q' to cleanly disconnect
- **Echo behavior:** The server reads client data and logs it to the debug port
- **Connection limits:** If the server is full (10 connections), new clients are rejected with an error message

12.47.6.1.5 Testing the Application

12.47.6.1.5.1 Using Telnet

The easiest way to test this application is using multiple Telnet sessions:

```
telnet <netburner_ip_address> 23
```

You can open up to 10 simultaneous Telnet connections to test the multi-client functionality.

12.47.6.1.5.2 Expected Behavior

1. **Successful connection:** You'll see "Welcome to the NetBurner Multi-Socket TCP Server! 'Q' to quit."
2. **Server full:** If you try to connect when 10 clients are already connected, you'll see "I am sorry, but the server is full"
3. **Quit command:** Type 'Q' to cleanly disconnect
4. **Debug output:** Connection status, received data, and errors are logged to the serial debug port

12.47.6.1.6 Code Structure

12.47.6.1.6.1 Key Functions

- `**UserMain()` **: Main application entry point and event loop
- `**select()` **: Monitors multiple file descriptors for activity
- `**accept()` **: Accepts new client connections
- `**read()` **: Reads data from client connections
- `**close()` **: Closes file descriptors and cleans up connections

12.47.6.1.6.2 File Descriptor Management

The application maintains an array of file descriptors (`fdArray`) where:

- Index 0-9: Active client connections (0 = unused slot)
- Separate `fdListen`: The listening socket for new connections

12.47.6.1.6.3 Error Handling

- **Connection errors:** Automatically detected and connections are closed
- **Read errors:** Treated as connection problems and handled gracefully
- **Listen socket errors:** The listening socket is closed and reopened

12.47.6.1.7 Troubleshooting

- **Cannot connect:** Verify the NetBurner device has network connectivity and check IP address
- **Connection refused:** Ensure no other service is using port 23
- **Unexpected disconnections:** Check debug serial output for error messages
- **Server appears full:** Existing connections may not have closed properly; restart the application

12.47.7 TCP Resource Information

12.47.7.1 TCP Resource Information

12.47.7.1.1 Overview

This NetBurner embedded application demonstrates system resource monitoring capabilities by providing real-time information about TCP buffers, network sockets, and file descriptors. The application is designed to help developers understand and monitor network resource usage in embedded systems.

12.47.7.1.2 Features

The application provides three main monitoring functions:

- **System Buffer Monitoring:** Display the number of free system buffers
- **Socket Resource Tracking:** Show available network sockets and file descriptors
- **Dynamic Resource Testing:** Create and close listening sockets to demonstrate resource consumption

12.47.7.1.3 System Requirements

- NetBurner Hardware
- NetBurner SDK (tested with revision 3.5.4)
- Active network connection

12.47.7.1.4 Application Structure

12.47.7.1.4.1 Core Functions

The application utilizes three key system functions for resource monitoring:

- `GetFreeCount ()` - Returns the number of free system buffers
- `GetFreeSocketCount ()` - Returns the number of free sockets/file descriptors (excluding extra FDs)
- `GetFreeExtraFDCount ()` - Returns the number of extra file descriptors

12.47.7.1.4.2 Menu System

The application provides an interactive serial menu with the following options:

1. **Show free buffers and sockets** - Displays current resource availability
2. **Add a listen socket, display info, and close the socket** - Demonstrates resource consumption by creating a temporary listening socket on port 10000
3. **Get extra fd count** - Shows the count of extra file descriptors

12.47.7.1.5 Usage

1. Compile and deploy the application to your NetBurner device
2. Connect to the device via serial terminal
3. The application will automatically initialize the network and display the menu
4. Enter the number corresponding to your desired action:
 - Press '1' to view current resource status
 - Press '2' to test socket creation and see resource impact
 - Press '3' to check extra file descriptor count

12.47.7.1.6 Important Notes

12.47.7.1.6.1 Resource Management

- Each active socket (listening or connected) consumes system resources
- When the system runs out of buffers or sockets, it cannot accept incoming connections or make outgoing connections
- The listening socket test (option 2) temporarily uses port 10000 with a backlog of 5 connections

12.47.7.1.6.2 Network Initialization

- The application waits up to 10 seconds for an active network connection during startup
- System diagnostics are enabled by default (consider disabling for production builds)

12.47.7.1.7 Code Structure

```
main.cpp
  UserMain()           - Main application entry point
  showMenu()           - Display menu options
  processCommand()     - Handle user input
  System Functions     - Network resource monitoring
```

12.47.7.1.8 Development Notes

- System diagnostics are enabled for development purposes
- The code is designed for continuous operation with a command loop
- Error handling is implemented for socket operations

12.47.8 TCP Server

12.47.8.1 Simple TCP Server Example

A NetBurner embedded TCP server application that creates a listening socket on port 23 (Telnet) and accepts client connections for communication.

12.47.8.1.1 Overview

This application demonstrates how to implement a basic TCP server on NetBurner hardware. The server listens for incoming connections and provides a simple echo-style communication interface where it receives data from clients and displays it locally.

12.47.8.1.2 Features

- **TCP Server:** Creates a listening socket on port 23 (standard Telnet port)
- **Client Connection Handling:** Accepts incoming TCP connections from clients
- **Data Reception:** Reads data from connected clients using a simple `read()` function
- **Connection Management:** Handles client disconnections gracefully
- **Multi-tasking:** Uses RTOS tasks for concurrent operation

12.47.8.1.3 Technical Details

12.47.8.1.3.1 Configuration

- **Listen Port:** 23 (Telnet)
- **Receive Buffer Size:** 4096 bytes
- **Maximum Pending Connections:** 5
- **Task Priority:** MAIN_PRIO - 1 (higher than UserMain)

12.47.8.1.3.2 Key Components

TcpServerTask

The main server task that:

1. Creates a listening socket bound to INADDR_ANY on the specified port
2. Waits for client connections using `accept()`
3. Sends welcome messages to connected clients
4. Continuously reads data from clients until disconnection
5. Closes connections and waits for new ones

Buffer Management

- Uses a global receive buffer (`RXBuffer`) of 4096 bytes
- Null-terminates received data for string processing
- Displays received data length and content via `iprintf()`

12.47.8.1.4 Usage

12.47.8.1.4.1 Testing with Telnet

To test the TCP server, use a Telnet client from any system on the same network:

```
telnet <netburner_ip_address>
```

Where `<netburner_ip_address>` is the IP address of your NetBurner device.

12.47.8.1.4.2 Expected Behavior

1. The server displays "Waiting for connection on port 23..."
2. When a client connects, it shows the client's IP address
3. The client receives welcome messages:
 - "Welcome to the NetBurner TCP Server"
 - Connection details with server IP and port
4. Any data sent by the client is echoed to the server console
5. When the client disconnects, the server closes the connection and waits for new ones

12.47.8.1.5 Implementation Notes

12.47.8.1.5.1 Connection Lifecycle

- The server uses a blocking `accept()` call to wait for connections
- Each client connection is handled sequentially (single client at a time)
- The `read()` function blocks until data is received or connection closes
- Connection closure is detected when `read()` returns 0 or negative value

12.47.8.1.5.2 Limitations

- **Single Client:** This example handles one client connection at a time
- **Sequential Processing:** New connections must wait until the current client disconnects
- **No Timeout:** Connections remain open indefinitely until client closes

12.47.8.1.5.3 For Multiple Clients

This example uses a simple `read()` approach for single client connections. For handling multiple simultaneous client connections, refer to the "TCP Multi Socket Server" example.

12.47.8.1.6 System Requirements

- NetBurner hardware platform
- NetBurner RTOS (NBRRTOS)
- Network connectivity (Ethernet)
- Telnet client for testing

12.47.8.1.7 Troubleshooting

12.47.8.1.7.1 Common Issues

1. **Connection Refused:** Verify the NetBurner device IP address and network connectivity
2. **Port Already in Use:** Ensure no other service is using port 23
3. **No Response:** Check if the TCP server task started successfully

12.47.8.1.7.2 Debug Output

The application provides console output for:

- Server startup and port binding
- Client connection events with IP addresses
- Received data content and byte counts
- Connection closure events

12.47.8.1.8 Code Structure

```
main.cpp
TcpServerTask()    - Main TCP server loop
  listen()         - Create listening socket
  accept()         - Accept client connections
  read()           - Receive client data
  close()          - Close connections
UserMain()         - Application initialization
  init()           - System initialization
  OSTaskCreate()   - Create server task
```

12.47.8.1.9 Related Examples

For more advanced TCP server implementations, see:

- TCP Multi Socket Server - Handles multiple simultaneous client connections

12.47.9 TCP Server Using Via

12.47.9.1 TCP Server Via Example

12.47.9.1.1 Overview

This application demonstrates a TCP server implementation for multi-interface NetBurner devices using the `listenvia()` function. The server creates multiple listening sockets - one that listens on all network interfaces and additional sockets that listen on specific network interfaces.

12.47.9.1.2 Features

- **Multi-Interface Support:** Creates listening sockets for each available network interface
- **Multiple Concurrent Connections:** Supports up to 10 simultaneous TCP connections
- **Port Management:** Uses incrementing port numbers starting from port 23
- **Single-Task Architecture:** Uses `select()` to handle multiple sockets within one task
- **Connection Monitoring:** Provides detailed logging of connections and interface information

12.47.9.1.3 Technical Details

12.47.9.1.3.1 Configuration

- **Base Port:** 23 (Telnet)
- **Maximum Connections:** 10 concurrent connections
- **Read Buffer Size:** 1,024 bytes
- **Connection Queue:** Up to 5 pending connection requests per listening socket

12.47.9.1.3.2 Socket Architecture

The application creates the following listening sockets:

1. One socket listening on all interfaces (port 23)
2. For each available network interface:
 - One socket on port 24 (`listenPort + 1`)
 - One socket on port 24 + `interface_number`

12.47.9.1.3.3 Network Interface Handling

- Automatically detects available network interfaces using `GetInterfaceBlock()`
- Creates dedicated listening sockets for each interface using `listenvia()`
- Provides interface-specific connection information to clients

12.47.9.1.4 Usage

12.47.9.1.4.1 Building and Running

1. Compile the application for your NetBurner device
2. Deploy to the target hardware
3. The application will automatically start listening on configured ports

12.47.9.1.4.2 Testing Connections

Use Telnet to test connections from the command line:

```
telnet <ip_address> <port>
```

Examples:

- `telnet 192.168.1.100 23` - Connect to the all-interfaces socket
- `telnet 192.168.1.100 24` - Connect to interface 1, port 24
- `telnet 192.168.1.100 25` - Connect to interface 1, port 25

12.47.9.1.4.3 Client Interaction

- Upon connection, clients receive a welcome message
- Interface information is displayed showing which interface accepted the connection
- Type 'Q' to quit and close the connection
- Server echoes received data and provides connection status

12.47.9.1.5 Code Structure

12.47.9.1.5.1 Main Components

Initialization**

- System diagnostics enabled
- Waits for active network (10-second timeout)

- Displays IP addresses for all interfaces

Socket Management**

- `listeners[]` array stores all listening socket file descriptors
- `fdArray[]` manages active client connections
- Automatic socket recovery on errors

Event Loop**

- Uses `select()` for efficient I/O multiplexing
- Processes new connections, data reception, and errors
- Automatic cleanup of closed connections

12.47.9.1.5.2 Key Functions Used

- `listen()` - Creates socket listening on all interfaces
- `listenvia()` - Creates socket listening on specific interface
- `accept()` - Accepts incoming connections
- `select()` - Monitors multiple sockets for I/O events
- `TcpGetSocketInterface()` - Retrieves interface information

12.47.9.1.6 Error Handling

- **Connection Limit:** When server is full, new connections are rejected with a message
- **Socket Errors:** Automatic detection and recovery of failed listening sockets
- **Read Errors:** Proper cleanup of corrupted client connections
- **Interface Failures:** Continues operation even if some interfaces fail

12.47.9.1.7 Application Output

The application provides detailed console output including:

- Network interface discovery and socket creation status
- Connection acceptance with client IP and port information
- Interface identification for each connection
- Data reception logging
- Connection closure notifications
- Error condition reporting

12.47.9.1.8 Requirements

- NetBurner hardware platform
- NetBurner SDK with TCP networking support
- Multi-interface network configuration (optional for full functionality)

12.47.9.1.9 Notes

- System diagnostics are enabled by default (should be disabled for production)
- Socket file descriptors are automatically managed and cleaned up
- The server continues running indefinitely until manually stopped

12.47.10 TCP Speed Test

12.47.10.1 TCP Speed Test Application

12.47.10.1.1 Overview

This is a TCP speed test application designed for NetBurner embedded devices. The application creates a TCP server that measures network transmission performance by sending data to connected clients. It's designed to test the send speed capabilities of NetBurner hardware.

12.47.10.1.2 Features

- TCP server listening on configurable port (default: 1234)
- Automatic data buffer initialization with test patterns
- Network interface detection and IP address display
- Optional WiFi interface support
- Socket optimization for performance testing
- Real-time throughput measurement and reporting
- Client connection management with timeout handling

12.47.10.1.3 Hardware Requirements

- NetBurner embedded device
- Ethernet or WiFi network connectivity
- Debug serial port for monitoring

12.47.10.1.4 Software Components

12.47.10.1.4.1 Key Functions

Network Initialization

- Waits for active network connection
- Displays IP addresses of available interfaces
- Optional WiFi interface initialization (when ADD_WIFITEST is defined)

TCP Server Operation

- Creates listening socket on specified port
- Accepts incoming client connections
- Configures socket options for optimal performance:
 - Enables SO_NOPUSH option
 - Sets ACK buffers to 20
 - Sets TX buffers to 20

Speed Test Protocol

1. Server waits for client start signal (single byte)
2. Continuously sends 1460-byte data packets
3. Measures transmission time and byte count
4. Reports results when client disconnects

Data Pattern

- Test data consists of repeating ASCII characters ('A' + (index % 64))
- Creates predictable pattern for data integrity verification

12.47.10.1.5 Usage Instructions

12.47.10.1.5.1 Device Setup

1. Connect NetBurner device to network
2. Connect to debug serial port using terminal (MTTTY recommended)
3. Build and download the application to the device
4. Device will display its IP address and listening status

12.47.10.1.5.2 Running Speed Test

1. Note the IP address displayed by the device
2. Use the provided TcpSpeedTest.exe PC application:
`./TcpSpeedTest <ip_address>`
3. Add `-r` flag for continuous testing:
`./TcpSpeedTest <ip_address> -r`

12.47.10.1.5.3 Test Results

- Results displayed on both device serial output and PC command prompt
- Metrics include:
 - Total bytes sent
 - Time duration (in system ticks and seconds)
 - Calculated throughput

12.47.10.1.6 Configuration Options

12.47.10.1.6.1 WiFi Support

Uncomment `#define ADD_WIFITEST` to enable WiFi interface:

- Scans and displays available networks
- Attempts connection to configured network
- Note: Enable `FAST_WIFI_STACK` in [constants.h](#) for improved performance

12.47.10.1.6.2 Performance Optimization

- FlexSPI overclock option available (commented by default)
- Socket buffer optimization enabled
- System diagnostics available for debugging

12.47.10.1.7 Error Handling

- Connection timeout handling (10-second timeout for start signal)
- Socket operation error detection
- Graceful connection termination
- Automatic server restart after client disconnect

12.47.10.1.8 Performance Notes

- Uses aligned memory buffers for optimal performance
- Implements socket options for reduced latency
- Sends data in optimal 1460-byte chunks (standard Ethernet MTU)
- Continuous operation until manual reset

12.47.10.1.9 Troubleshooting

12.47.10.1.9.1 Common Issues

- **No network connection:** Check cable/WiFi configuration
- **Port binding failure:** Verify port 1234 is available
- **Client connection timeout:** Ensure PC application can reach device IP
- **Performance issues:** Consider enabling `FAST_WIFI_STACK` for WiFi

12.47.10.1.9.2 Debug Information

- Verbose mode available (set bVerbose = TRUE)
- System diagnostics enabled by default
- Serial output provides detailed connection status

12.47.11 TCP Stress Test

12.47.11.1 TCP Stress Test Application

12.47.11.1.1 Overview

This is a NetBurner embedded TCP stress testing application designed to measure various network performance metrics and response times. The application runs a series of sequential tests to evaluate TCP connection handling, data transfer speeds, and `select()` function performance.

12.47.11.1.2 Description

The TCP Stress Test application is designed to work in conjunction with a Windows program to exercise various test cases and measure network performance. It provides comprehensive testing of TCP socket operations including connection establishment, data exchange, and bulk data transfer capabilities.

12.47.11.1.3 Features

The application performs five sequential stress tests:

12.47.11.1.3.1 1. Listening Test

- Listens for incoming TCP connections on port 10000
- Exchanges single characters with connecting clients
- Measures the time and number of connections processed
- Terminates when it receives an 'X' character from a client

12.47.11.1.3.2 2. Connecting Test

- First accepts one incoming connection to determine the client's IP address
- Then repeatedly makes outgoing connections back to that client
- Exchanges single characters with each connection
- Measures connection establishment performance
- Terminates when it receives an 'X' character

12.47.11.1.3.3 3. Response Test

- Tests worst-case TCP performance by sending one byte at a time
- Listens for incoming bytes and echoes them back immediately
- Measures response time for single-byte exchanges
- Demonstrates TCP overhead for minimal data transfers

12.47.11.1.3.4 4. Select Response Test

- Tests the `select()` function performance for monitoring multiple file descriptors
- Uses `select()` to wait for data availability before reading
- Measures the response time of the `select()` system call
- Includes error handling for `select()` failures

12.47.11.1.3.5 5. Bulk Speed Test

- Tests maximum data transfer throughput
- Sends large buffers (`TCP_BUFFER_SEGMENTS * 1460` bytes per write)
- Uses socket options for optimized bulk transfer (`SO_NOPUSH`)
- Measures total bytes transferred until connection closes

12.47.11.1.4 Configuration

- **Listen Port:** 10000 (default)
- **Buffer Size:** 20,000 bytes for bulk data test
- **Time Resolution:** 20 ticks per second (`TICKS_PER_SECOND`)
- **Stack Size:** `USER_TASK_STK_SIZE` for the test task
- **Priority:** `MAIN_PRIO - 1` for the test task

12.47.11.1.5 Technical Details

12.47.11.1.5.1 Key Components

- **TestTask:** Main task that runs all five tests sequentially in a loop
- **UserMain:** Initialization function that sets up networking and creates the test task
- **DataBuffer:** 20KB aligned buffer used for bulk data transfers
- **Socket Options:** Uses `SO_NOPUSH` and custom ACK buffer settings for optimization

12.47.11.1.5.2 Network Requirements

- Active network connection required
- Compatible with NetBurner hardware platforms
- Requires a Windows client program (not included in NetBurner project)
- Uses standard TCP/IP sockets

12.47.11.1.5.3 Timing and Measurements

All timing measurements are reported in "ticks" where the default resolution is 20 ticks per second. The application measures:

- Connection establishment time
- Single-byte response time
- `Select()` function response time
- Bulk data transfer rates
- Total connections processed

12.47.11.1.6 Usage

1. Compile and deploy to NetBurner hardware
2. Ensure network connectivity is established
3. Run the corresponding Windows client program
4. The application will automatically cycle through all five tests
5. Press any key during execution to display network counters
6. Monitor serial output for test results and timing information

12.47.11.1.7 Project Structure

```
tcp-stress-test/
  main.cpp      # Main application source code
  ReadMe.txt    # Original documentation
  pc/           # Windows client program (exclude from NetBurner build)
```

Warning

Do not include the `pc` folder in your NetBurner project build as it contains Windows-specific client code.

12.47.11.1.8 Output Format

Each test provides timing results in the following format: Output Format Each test provides timing results in the following format:

"Accepted X times in Y ticks" (Listening Test) "Connected X times in Y ticks" (Connecting Test) "Responded X times in Y ticks" (Response Test) "Select Responded X times in Y ticks" (Select Response Test) "Nsent = X" (Bulk Speed Test)

12.48 Telnet Command**12.48.1 NetBurner Telnet Command Example**

A demonstration application that implements a multi-interface command processor using the NetBurner Command Processor Library. This application provides remote command access through serial port, TCP/Telnet, and optionally SSH connections.

12.48.1.1 Overview

The TelnetCmd example showcases how to create a command-line interface that can accept connections and process user commands from multiple sources simultaneously:

- Serial port (UART) connection
- TCP/Telnet connection on port 23
- SSH connection on port 22 (if SSH support is enabled)

12.48.1.2 Features**12.48.1.2.1 Multi-Interface Support**

- **Serial Port:** Direct UART connection at 115,200 baud, 8 data bits, 1 stop bit, no parity
- **Telnet:** Standard TCP connection on port 23 with up to 5 concurrent connections
- **SSH:** Encrypted connection on port 22 with up to 5 concurrent connections (optional)

12.48.1.2.2 Authentication System

- Simple username/password authentication
- Rejects credentials where username matches password (for testing)
- Separate SSH authentication handler with support for key-based authentication
- Configurable login prompts and timeouts

12.48.1.2.3 Session Management

- Session tracking with unique identifiers
- Connection counters and session state management
- Customizable prompts showing connection type (Serial0, Serial1, or Remote)
- Automatic timeout handling (60-second idle timeout)

12.48.1.2.4 Command Processing

- Extensible command framework
- Built-in "logout" command to close connections
- Command echo and response system
- Graceful connection handling and cleanup

12.48.1.3 Configuration

12.48.1.3.1 Serial Port Settings

```
#define SERIALPORT_TO_USE (0)      // Serial port 0
#define BAUDRATE_TO_USE (115200)  // 115.2k baud
#define STOP_BITS (1)             // 1 stop bit
#define DATA_BITS (8)            // 8 data bits
```

12.48.1.3.2 Network Settings

```
#define TCP_PORT_TO_USE (23)       // Standard Telnet port
```

12.48.1.4 Callback Functions

The application implements five main callback functions:

12.48.1.4.1 Authentication Callbacks

- `ProcessTcpAndSerialAuth()`: Handles TCP and serial authentication
- `ProcessSshAuth()`: Handles SSH authentication with key/password support

12.48.1.4.2 Connection Management

- `ProcessConnect()`: Called when new connections are established
- `ProcessDisconnect()`: Handles connection cleanup and goodbye messages
- `ProcessPrompt()`: Generates custom prompts for each session

12.48.1.4.3 Command Processing

- `ProcessCommand()`: Processes all user commands and responses

12.48.1.5 Connection Types and Prompts

The application identifies different connection types and provides appropriate prompts:

- **Serial0:** Direct serial connection on UART 0
- **Serial1:** Direct serial connection on UART 1
- **Remote:** TCP/Telnet or SSH network connections

Example session prompt: `NB:Remote>`

12.48.1.6 Disconnect Handling

The application handles various disconnect scenarios:

- **Timeout:** 60-second idle timeout
- **Socket Closed:** Network connection terminated
- **Authentication Failed:** Invalid credentials provided
- **Manual Logout:** User-initiated logout command
- **System Closed:** Application-initiated disconnect

12.48.1.7 SSH Support (Optional)

When SSH support is enabled (`NB_SSH_SUPPORTED`):

- Automatic certificate and key generation
- Secure encrypted connections
- Key-based authentication support
- Automatic retry logic for SSH listener startup

12.48.1.8 Usage

1. **Serial Connection:** Connect to the configured serial port at 115,200 baud
2. **Telnet Connection:** `telnet <device_ip> 23`
3. **SSH Connection:** `ssh <username>@<device_ip>` (if SSH enabled)

12.48.1.8.1 Authentication

- Enter any username and password combination
- Username and password must be different (test restriction)
- Type "logout" to disconnect

12.48.1.8.2 Sample Session

```
Hello!
Welcome to the NetBurner Test Command Program
Number of connections since boot since boot: 1
Please Log in
Username: admin
Password: password123

NB:Remote>help
# Remote Sent Cmd[help]

NB:Remote>logout
GoodBye
```

12.48.1.9 Build Requirements

- NetBurner development environment
- Command Processor Library
- Optional: SSH support libraries (NB_SSH_SUPPORTED)

12.48.1.10 System Integration

The application integrates with NetBurner's RTOS:

- Runs command processor at `MAIN_PRIO + 1` priority
- Uses system diagnostics (should be disabled for production)
- Waits for active network before starting SSH services
- Replaces standard I/O streams with serial port

12.48.1.11 Customization

To extend this application:

1. **Add Commands:** Modify `ProcessCommand()` to handle new command strings
2. **Change Authentication:** Update authentication callbacks for different credential checking
3. **Modify Prompts:** Customize `ProcessPrompt()` for different prompt formats
4. **Add Connection Types:** Extend connection handling for additional interfaces
5. **Configure Timeouts:** Adjust `CmdIdleTimeout` for different session timeouts

12.48.1.12 Security Notes

- This is a demonstration application with simplified authentication
- Production deployments should implement secure credential storage
- Consider disabling system diagnostics for production use
- SSH provides encrypted communication when available

12.48.1.13 Related Documentation

See NetBurner Command Processor Library documentation for additional API details and advanced configuration options.

12.49 TFTP Client

12.49.1 TFTP Client Application

12.49.1.1 Overview

This NetBurner TFTP Client example demonstrates how to use a NetBurner device to communicate with a TFTP (Trivial File Transfer Protocol) server for both file upload and download operations.

12.49.1.2 Features

- **File Download:** Retrieve files from a TFTP server using the `GetTFTP ()` function
- **File Upload:** Send files to a TFTP server using the `SendTFTP ()` function
- **Configurable Server:** Support for specifying TFTP server IP address
- **Error Handling:** Comprehensive error checking and timeout management
- **Binary and Text Modes:** Support for both binary ("b") and text ("t") transfer modes

12.49.1.3 Requirements

12.49.1.3.1 TFTP Server Configuration

1. The TFTP server must be configured to handle both read and write operations
2. Ensure the server allows file transfers in the directory where test files will be stored

12.49.1.3.2 NetBurner Device Configuration

The TFTP server address can be configured through multiple methods:

- NetBurner device's Configuration Server web interface
- Direct modification in the application code (currently set to 10.1.1.100)
- Serial Configuration Server of the device
- Application data flash memory using configuration parameters

12.49.1.4 Application Flow

12.49.1.4.1 Initialization

1. Initialize the network stack
2. Enable system diagnostics
3. Start HTTP web server on default port 80
4. Wait for DHCP address assignment (5-second timeout)

12.49.1.4.2 TFTP Operations

1. **File Upload Test:** Creates a sample text file with 10 lines and uploads it as "tftp.txt"
2. **File Download Test:** Downloads "tftp.txt" from the TFTP server
3. **Results Display:** Shows operation results and data content

12.49.1.5 Code Structure

12.49.1.5.1 Key Functions

12.49.1.5.1.1 SendTFTP_Test ()

- Creates sample data (10 lines of text)
- Uploads data to TFTP server as "tftp.txt"
- Handles upload results and error conditions

12.49.1.5.1.2 ReadTFTP_Test ()

- Downloads "tftp.txt" from TFTP server
- Displays received data and byte count
- Handles download results and error conditions

12.49.1.5.1.3 UserMain ()

- Main application entry point
- Performs initialization sequence
- Executes both send and receive tests
- Enters infinite loop after completion

12.49.1.5.2 Configuration Parameters

```
#define TFTP_SERVER_IP "10.1.1.100" // Default TFTP server IP address
const int bufferSize = 1024;      // Transfer buffer size
```

12.49.1.5.3 TFTP Function Parameters

Both `GetTFTP ()` and `SendTFTP ()` functions use similar parameter structures:

- **Filename:** Name of file to transfer ("tftp.txt")
- **Mode:** Transfer mode - "b" for binary, "t" for text
- **Buffer:** Data buffer for transfer operations
- **Length:** Data length for transfers
- **Timeouts:** Operation and packet-level timeout values
- **Server IP:** TFTP server IP address

12.49.1.6 Return Codes

The application handles three main TFTP return codes:

- **TFTP_OK:** Operation completed successfully
- **TFTP_TIMEOUT:** Operation timed out
- **TFTP_ERROR:** Error occurred during operation

12.49.1.7 Usage Instructions

1. **Setup TFTP Server:** Configure a TFTP server on your network with read/write permissions
2. **Configure IP Address:** Update `TFTP_SERVER_IP` to match your TFTP server's IP address
3. **Build and Deploy:** Compile and deploy the application to your NetBurner device
4. **Network Connection:** Ensure the NetBurner device is connected to the same network as the TFTP server
5. **Monitor Output:** Use a serial terminal to view application output and results
6. **Reset to Repeat:** Reset the device to run the test sequence again

12.49.1.8 Test Files

The application works with a test file named "tftp.txt":

- **Upload Test:** Creates and sends a file with 10 numbered lines
- **Download Test:** Retrieves the same file from the server
- **Content Verification:** Displays received data for verification

12.49.1.9 Network Requirements

- DHCP-enabled network for automatic IP assignment
- Network connectivity between NetBurner device and TFTP server
- TFTP server listening on standard port 69
- Firewall permissions for TFTP traffic

12.49.1.10 Troubleshooting

12.49.1.10.1 Common Issues

- **Timeout Errors:** Check network connectivity and TFTP server status
- **Permission Errors:** Verify TFTP server write permissions
- **File Not Found:** Ensure test file exists on server for download operations
- **Network Issues:** Confirm [DHCP](#) assignment and IP connectivity

12.49.1.10.2 Debug Information

- System diagnostics are enabled by default
- Serial output provides detailed operation status
- HTTP server runs on port 80 for additional configuration options

12.49.1.11 Customization Options

12.49.1.11.1 Server Configuration

Replace hardcoded IP with configurable parameters:

```
static config_string gTftpServerName(appdata, "DNS_or_IPAddress", "TFTP_SERVER");  
// or  
static config_IPADDR4 gTftpServerAddress(appdata, "10.1.1.123", "TFTP_SERVER");
```

12.49.1.11.2 Buffer Size

Adjust `bufferSize` constant to handle larger file transfers as needed.

12.49.1.11.3 Timeout Values

Modify timeout parameters in function calls:

- `TICKS_PER_SECOND * 10`: Overall operation timeout
- `TICKS_PER_SECOND * 5`: Individual packet timeout

12.49.1.12 Development Notes

- Remove `EnableSystemDiagnostics()` for production deployments
- Consider implementing error recovery mechanisms for production use
- The application demonstrates basic TFTP functionality suitable for embedded systems
- HTTP server provides additional configuration capabilities through web interface

12.50 TFTP Server

12.50.1 TFTP Server Application

12.50.1.1 Overview

This application implements a TFTP (Trivial File Transfer Protocol) Server as a servlet on an embedded device. The server allows clients to transfer files using the TFTP protocol, which is a simplified version of FTP designed for minimal overhead and ease of implementation.

12.50.1.2 Application Description

The TFTP Server application is built on the NetBurner NNDK (NetBurner Network Development Kit) platform and provides the following functionality:

- **TFTP Server Implementation:** Runs as a servlet on the device
- **File Serving:** Makes specific files available for TFTP transfer
- **Web Interface:** Includes an HTTP web server for device management
- **Network Integration:** Full network stack initialization and management

12.50.1.3 Key Components

12.50.1.3.1 Main Application (`main.cpp`)

The main application file contains:

- **Application Name:** "TFTP_Server"
- **TFTP Server Object:** `MyServer` - the main TFTP server instance
- **File Source:** `file1` - makes test data available as "test.dat"
- **Test Data:** References embedded test data array (5500 bytes)

12.50.1.3.2 Test Data (`data.cpp`)

Contains auto-generated test data:

- **Array Name:** `testdata_array`
- **Size:** 5500 bytes
- **Content:** Formatted hexadecimal test data representing numbered lines
- **Format:** Each line contains line numbers and repeating digit patterns

12.50.1.3.3 Sample File (`filetosend.txt`)

Provides a human-readable version of the test data:

- 50 lines of formatted text
- Each line prefixed with a 4-digit line number (0000-0059)
- Contains repeating digit patterns for testing transfer integrity

12.50.1.4 Functionality

12.50.1.4.1 Network Services

1. **TFTP Server:**

- Serves files via TFTP protocol
- Makes "test.dat" available for download
- Uses embedded test data array

2. **HTTP Web Server:**

- Starts on default port 80
- Provides web-based interface
- Enables system diagnostics

3. **Network Stack:**

- Full network initialization
- Active network detection with 5-second timeout
- Continuous operation loop

12.50.1.4.2 File Serving

The application demonstrates TFTP file serving by:

- Creating a TFTP server instance
- Registering a file source named "test.dat"
- Mapping the file to the embedded test data array
- Making the file available for TFTP client downloads

12.50.1.5 System Requirements

- NetBurner NNDK platform
- Network connectivity
- Sufficient memory for test data (5500+ bytes)

12.50.1.6 Usage

1. **Startup Sequence:**

- Network stack initialization
- System diagnostics enablement
- HTTP server startup
- Network connection wait (up to 5 seconds)
- Status reporting

2. **Operation:**

- Continuous operation with 1-second intervals
- TFTP server runs in background
- Web server provides management interface

3. **File Access:**

- TFTP clients can connect to the device
- Request file "test.dat" for download
- Receive the 5500-byte test data array

12.50.1.7 Development Notes

- Built using NetBurner NNDK framework
- Includes system diagnostics (recommended for development only)
- Uses standard TFTP protocol implementation
- Embedded test data for demonstration purposes

12.50.1.8 File Structure

```
main.cpp          # Main application entry point
data.cpp          # Auto-generated test data array
filetosend.txt    # Human-readable test data sample
ReadMe.txt        # Basic application description
```

12.50.1.9 Output Information

Upon startup, the application displays:

- Application name and version
- NNDK revision information
- Network status updates

12.50.1.10 Protocol Support

- **TFTP**: File transfer protocol implementation
- **HTTP**: Web server for device management
- **TCP/IP**: Full network stack support

12.51 Time Functions

12.51.1 Time Functions Application

A comprehensive embedded C++ application demonstrating system time management, time zone handling, and various time-related operations on NetBurner embedded systems.

12.51.1.1 Overview

This application provides a complete example of how to work with system time functions in an embedded environment. It includes capabilities for setting system time via NTP or manual entry, managing time zones, and displaying time information in various formats.

12.51.1.2 Features

12.51.1.2.1 Core Time Operations

- **System Time Display**: Shows current time in multiple formats (GMT, UTC, Local)
- **NTP Time Synchronization**: Automatically sets system time using Network Time Protocol
- **Manual Time Setting**: Interactive prompts for setting system time manually
- **Time Zone Management**: Comprehensive time zone selection for US and international locations
- **Boot Time Tracking**: Displays elapsed time since system boot with millisecond precision

12.51.1.2.2 Time Formats Supported

- **time_t**: Calendar time since epoch
- **struct tm**: Calendar time structure with individual components
- **GMT/UTC**: Greenwich Mean Time and Coordinated Universal Time
- **Local Time**: Time adjusted for current time zone and daylight savings

12.51.1.2.3 Interactive Menu System

The application provides a user-friendly serial interface with the following options:

1. Display system time information
2. Set system time using NTP
3. Set local time zone
4. Set system time manually
5. Test time zone functionality
6. Display time since boot with millisecond precision 0. Reboot system

12.51.1.3 File Structure

12.51.1.3.1 main.cpp

The primary application file containing:

- **displaySystemTime()**: Comprehensive time display function showing various time formats
- **setSystemTimeNTP()**: NTP-based time synchronization
- **setSystemTimeManual()**: Manual time setting with parameter input
- **setSystemTimeManualPrompt()**: Interactive time setting via user prompts
- **setSystemTimeZone()**: Time zone selection interface
- **displayTimeSinceBootInMsec()**: Boot time calculation and display
- **UserMain()**: Main application loop with menu-driven interface

12.51.1.3.2 TimezoneTest.cpp

Specialized testing module for time zone functionality:

- **testTimeZone()**: Validates time zone settings and displays daylight savings transitions
- **showHumanTimeGMT()**: GMT time formatting and display
- **showHumanTimeLocal()**: Local time formatting with timezone offset
- **getOffsetAt()**: Calculates timezone offset for specific timestamps

12.51.1.4 Key Time Functions Demonstrated

12.51.1.4.1 Standard C Time Functions

- `time()`: Current calendar time retrieval
- `gmtime_r()`: Convert `time_t` to UTC struct `tm` (thread-safe)
- `localtime_r()`: Convert `time_t` to local struct `tm` (thread-safe)
- `mktime()`: Convert struct `tm` to `time_t`
- `timegm()`: Convert GMT struct `tm` to `time_t`
- `strftime()`: Format time strings for display
- `difftime()`: Calculate time differences

12.51.1.4.2 NetBurner-Specific Functions

- `SetTimeNTPFromPool()`: NTP synchronization
- `set_time()`: System time setting
- `tzsetchar()`: Time zone configuration
- `Get_msec()`: Millisecond-precision system uptime

12.51.1.5 Time Zone Support

The application includes comprehensive time zone support:

- **US/Canada Time Zones**: Major North American time zones
- **International Time Zones**: Global time zone coverage
- **Daylight Savings Time**: Automatic DST transitions
- **POSIX Time Zone Strings**: Standard timezone specification format

12.51.1.5.1 Time Zone Testing

The timezone test functionality provides:

- Current timezone display with offset
- Start and end of year boundary analysis
- Automatic daylight savings time transition detection
- Precise timing of DST changes

12.51.1.6 Usage Requirements

12.51.1.6.1 Prerequisites

Before using the time functions:

1. **Network Connectivity**: Required for NTP synchronization
2. **System Initialization**: Proper embedded system startup
3. **Time Zone Configuration**: Must be set for accurate local time display

12.51.1.6.2 Compilation Dependencies

- NetBurner SDK headers: `init.h`, `hal.h`, `nbttime.h`, `iosys.h`, `timezones.h`
- Standard C library time functions
- Network stack for NTP functionality

12.51.1.7 Time Precision and Accuracy

12.51.1.7.1 Precision Levels

- **Second Precision:** Standard `time_t` and struct tm operations
- **Millisecond Precision:** Boot time tracking via `Get_msec()`
- **NTP Accuracy:** Network-synchronized time with sub-second accuracy

12.51.1.7.2 Boot Time Calculations

The application calculates and displays:

- Days, hours, minutes, seconds since boot
- Remaining milliseconds for complete precision
- Multiple format options for different use cases

12.51.1.8 Error Handling

The application includes robust error handling for:

- **NTP Failures:** Graceful fallback when network time is unavailable
- **Invalid Input:** Input validation for manual time entry
- **Time Zone Errors:** Validation of timezone selection
- **System Time Issues:** Error reporting for time setting operations

12.51.1.9 Example Output

```
System Time Information ***
Seconds obtained from time(): time_t = 1609459200
Seconds since program start = 1234

Time obtained from gmtime_r():
UTC Time   = Fri, 01 Jan 2021 00:00:00

Time obtained from localtime_r():
Local Time = Thu, 31 Dec 2020 16:00:00 PST [-0800 offset UTC]

Current Time Zone: PST (-0800)
Daylight Savings Time Changes:
At Time: Sun, 08 Mar 2020 01:59:59 (-0800)
Offset change is: -28800 to -25200
```

12.51.1.10 Development Notes

12.51.1.10.1 Thread Safety

- Uses thread-safe time functions (`gmtime_r`, `localtime_r`)
- Suitable for multi-threaded embedded environments
- Proper buffer management for string operations

12.51.1.10.2 Memory Management

- Static buffer allocation for time string formatting
- Minimal dynamic memory usage
- Embedded-friendly memory patterns

12.51.1.10.3 Performance Considerations

- Efficient boot time calculations using integer arithmetic
- Minimal system calls for time operations
- Optimized timezone transition detection algorithms

12.51.1.11 Integration

This application serves as a reference implementation for:

- Real-time clock (RTC) integration
- Network time synchronization systems
- Time-aware embedded applications
- Logging systems requiring accurate timestamps
- Scheduling and timing-critical applications

The modular design allows individual functions to be extracted and integrated into larger embedded systems requiring time management capabilities.

12.52 Timers

System timer examples:

- [Timer](#)
- [Interval Timer](#)
- [Stopwatch Timer](#)

12.52.1 Timer

12.52.1.1 HiRes Delay Timer Example

12.52.1.1.1 Overview

The HiRes Delay Timer Example is a test application that demonstrates and validates the precision of microsecond-level delays using hardware timers. This application provides benchmarking capabilities for the HiResDelay class, which uses hardware timers to achieve high-precision timing delays.

12.52.1.1.2 Features

- **High-Resolution Timing:** Uses hardware timers for microsecond-precision delays
- **Precision Measurement:** Accurately measures actual delay times vs. requested delays
- **Error Analysis:** Calculates and displays timing errors for performance evaluation
- **Comprehensive Testing:** Tests delays ranging from 1 microsecond to 100+ milliseconds
- **Priority Control:** Runs tests at high priority to minimize interference from other tasks

12.52.1.1.3 How It Works

The application uses the `HiResDelay` class in conjunction with a `StopWatch` to:

1. Request a specific delay duration in microseconds
2. Measure the actual time taken for the delay
3. Calculate the error between requested and actual delay times
4. Display results for analysis

12.52.1.1.3.1 Test Sequence

The application runs a comprehensive test suite with the following delay values (each tested twice):

- Ultra-short delays: 1, 2 microseconds
- Short delays: 5, 10, 15, 20, 21 microseconds
- Medium delays: 50, 100 microseconds
- Long delays: 1ms, 2ms, 10ms, 20ms
- Very long delays: 100ms, 100.001ms

12.52.1.1.4 Usage

1. **Start the Application:** The program initializes and displays "Application Started"
2. **Run Tests:** Press any key when prompted to execute the delay test sequence
3. **View Results:** Each test displays:
 - Requested delay time (microseconds)
 - Actual measured delay time
 - Error between requested and actual time

12.52.1.1.4.1 Sample Output Format

```
1:      1.05  err:   0.05
2:      2.12  err:   0.12
5:      4.98  err:  -0.02
```

12.52.1.1.5 Key Components

12.52.1.1.5.1 DelayObject Class

- Provides the `DelayUsec()` method for microsecond delays
- Utilizes hardware timers for precision timing

12.52.1.1.5.2 StopWatch Class

- High-precision timer for measuring elapsed time
- Converts timer ticks to microseconds for analysis
- **Methods:** `Start()`, `Stop()`, `Clear()`, `Convert()`

12.52.1.1.5.3 DOTest Function

Parameters:

- `DelayObject &dob`: Reference to the delay object
- `int usec`: Requested delay in microseconds

Functionality:

- Starts precision timing measurement
- Executes the requested delay
- Measures actual delay duration
- Calculates and displays timing error

12.52.1.1.6 System Requirements

- Compatible with embedded systems supporting:
 - Hardware timers
 - Real-time operating system with priority control
 - Standard I/O operations (`printf`, `getchar`)

12.52.1.1.7 Technical Notes

- **Priority Handling**: Tests run at priority level 2 to minimize task interference
- **System Diagnostics**: Enabled for development (should be disabled in production)
- **Precision**: Capable of measuring sub-microsecond timing differences
- **Error Reporting**: Displays both positive and negative timing errors

12.52.1.1.8 Build Dependencies

Required header files:

- `init.h` - System initialization
- `iosys.h` - I/O system functions
- `stopwatch.h` - High-precision timing
- `HiResDelay.h` - Hardware timer delay functionality

12.52.1.1.9 Purpose

This application serves as:

- **Validation Tool**: Verifies HiResDelay class accuracy
- **Benchmarking Utility**: Measures system timing performance
- **Development Aid**: Helps optimize timing-critical applications
- **Quality Assurance**: Ensures timing requirements are met in embedded systems

The comprehensive test suite helps developers understand the timing characteristics and limitations of their hardware platform, enabling them to make informed decisions about timing-critical code implementation.

12.52.2 Interval Timer

12.52.2.1 Interval Timer Example Application

12.52.2.1.1 Overview

The Interval Timer Example demonstrates how to use hardware timers on NetBurner platforms to schedule periodic interrupts for time-critical applications. This example showcases three different methods of utilizing interval timers for reliable timing operations.

The default for the timer constructor is to use the next available system timer, but you can also specify a specific timer as an optional parameter. Depending on the application requirements, the system may also use one or more of the available timers.

Warning

Always check the return value when creating a timer object. A value of less than 0 means a timer could not be allocated.

12.52.2.1.2 Features

The application provides three distinct timing mechanisms:

1. **Semaphore Posting** - Posts to an OS semaphore at regular intervals
2. **Flag Posting** - Posts to OS flags at specified frequencies
3. **Interrupt Callback** - Triggers custom interrupt service routines

12.52.2.1.3 Hardware Timer Availability

The number of available timers varies by NetBurner platform:

- **MODM7AE70**: Timers 0-11 (12 timers)
- **MOD54415**, **MOD54417**, **NANO54415**, **SB800EX**: Timers 1-3 (3 timers)
- **MODRT1061**: Timers 0-20 (21 timers)

12.52.2.1.4 Key Components

12.52.2.1.4.1 Timer Management

- Automatic timer allocation using next available system timer
- Manual timer specification supported via optional constructor parameter
- Error handling for timer allocation failures (return value < 0)

12.52.2.1.4.2 Synchronization Objects

- `OS_SEM timerSem` - Semaphore for timer-based synchronization
- `OS_FLAGS timerFlag` - Flag group for event signaling
- Global interrupt counter for callback demonstration

12.52.2.1.5 Application Menu Options

12.52.2.1.5.1 Option 1: Semaphore Timer

- Prompts user for desired frequency (minimum 20 Hz)
- Creates interval timer that posts to semaphore
- Demonstrates semaphore counting mechanism
- Reports total semaphore posts after completion

12.52.2.1.5.2 Option 2: Flag Timer

- User-configurable flag posting frequency (minimum 20 Hz)
- Interactive mode with manual termination
- Shows flag pending and clearing operations
- Counts and reports flag post events

12.52.2.1.5.3 Option 3: Interrupt Callback

- Configurable interrupt frequency (minimum 20 Hz)
- Demonstrates custom ISR callback implementation
- Interactive termination via keypress
- Tracks and reports interrupt occurrences

12.52.2.1.6 Important Implementation Notes

12.52.2.1.6.1 Timer Allocation

Always verify timer allocation success:

```
int timerNumber = IntervalOSSem(&timerSem, freq);
if (timerNumber < 0) {
    printf("Error %d\r\n", timerNumber);
    // Handle allocation failure
}
```

12.52.2.1.6.2 Interrupt Service Routine Guidelines

The timer callback function (`timerInterruptCallbackFunc`) operates in interrupt context:

- Keep execution time minimal
- Avoid blocking operations
- Use interrupt-safe functions only

12.52.2.1.6.3 Frequency Limitations

- Minimum frequency enforced at 20 Hz for all timer modes
- Higher frequencies possible but dependent on system capabilities
- Consider system load when selecting frequencies

12.52.2.1.7 Code Structure

12.52.2.1.7.1 Main Components

- `UserMain()` - Application entry point and main loop
- `displayMenu()` - User interface display
- `processCommand()` - Command processing and timer setup
- `timerInterruptCallbackFunc()` - ISR callback demonstration

12.52.2.1.7.2 Dependencies

- `<IntervalTimer.h>` - Timer functionality
- `<nbstring.h>` - String processing utilities
- `<init.h>` - System initialization

12.52.2.1.8 Usage Instructions

1. Compile and flash the application to your NetBurner device
2. Connect via serial terminal
3. Wait for network initialization to complete
4. Select desired timer mode (1, 2, or 3) from menu
5. Enter frequency when prompted (minimum 20 Hz)
6. Observe timer operation and results
7. For interrupt mode, press any key to stop timing

12.52.2.1.9 System Requirements

- NetBurner development environment
- Compatible NetBurner hardware platform
- Serial terminal for user interaction
- Active network connection (for system diagnostics)

12.52.2.1.10 Error Handling

The application includes comprehensive error checking:

- Timer allocation validation
- Frequency boundary enforcement
- User input validation
- System timer availability verification

12.52.2.1.11 Performance Considerations

- Timer precision depends on system clock accuracy
- High frequencies may impact overall system performance
- Multiple simultaneous timers share system resources
- ISR execution time affects maximum achievable frequencies

12.52.2.1.12 Development Notes

- System diagnostics enabled for development (consider removing for production)
- Application runs in continuous loop for repeated testing
- Memory cleanup handled automatically by RTOS
- Thread-safe operations using NetBurner RTOS primitives

12.52.3 Stopwatch Timer

12.52.3.1 Stopwatch Example Application

12.52.3.1.1 Overview

This application demonstrates the usage of a [StopWatch](#) timer object for measuring execution time of code segments. The [StopWatch](#) class provides precise timing capabilities for performance measurement and profiling.

12.52.3.1.2 Application Description

The Stopwatch Example application is a NetBurner RTOS-based program that showcases how to use the [StopWatch](#) class to measure the execution time of a simple counting loop.

12.52.3.1.2.1 Key Features

- Demonstrates [StopWatch](#) object creation and usage
- Measures execution time of a 10,000 iteration loop
- Displays timing results with high precision
- Shows timer resolution capabilities
- Includes system diagnostics and network initialization

12.52.3.1.3 Code Structure

12.52.3.1.3.1 Main Components

- **StopWatch Object:** Core timing functionality
- **Performance Test:** Simple counting loop for demonstration
- **System Initialization:** Network and diagnostic setup
- **Continuous Operation:** Maintains application state

12.52.3.1.3.2 Execution Flow

1. Initialize system and enable diagnostics
2. Wait for active network connection
3. Create [StopWatch](#) instance
4. Start timing measurement
5. Execute 10,000 iteration counting loop
6. Stop timing measurement
7. Display elapsed time and timer resolution
8. Enter continuous operation mode

12.52.3.1.4 Usage

The application automatically runs the timing demonstration upon startup. The output includes:

- Elapsed time in timer ticks
- Elapsed time converted to seconds (scientific notation)
- Timer resolution in seconds

12.52.3.1.4.1 Sample Output

```
Starting stopwatch on for loop of 10000 counts...complete
Elapsed time: [ticks], [seconds] seconds
Resolution: [resolution] seconds
```

12.52.3.1.5 Stopwatch Class Interface

Based on the documentation, the [StopWatch](#) class provides:

- **Start()**: Begin timing measurement
- **Stop()**: End timing measurement
- **GetTime()**: Retrieve elapsed time in timer ticks
- **Convert()**: Convert timer ticks to seconds
- **CountResolution()**: Get timer resolution

12.52.3.1.6 System Requirements

- NetBurner RTOS environment
- Network connectivity (application waits for active network)
- Compatible NetBurner hardware platform

12.52.3.1.7 Build Dependencies

- [init.h](#) - System initialization
- [nbrtos.h](#) - NetBurner RTOS functions
- [string.h](#) - Standard C string operations
- [stopwatch.h](#) - [StopWatch](#) class definition

12.52.3.1.8 Notes

- System diagnostics are enabled for development/debugging
- Application enters infinite loop after demonstration
- Uses `OSTimeDly(TICKS_PER_SECOND)` for timing delays
- Designed for embedded NetBurner platforms

12.53 UDP

The UDP API includes both socket based and object based (UDP Packet) implementations.
UDP Examples:

- [UDP to Serial](#)
- [UDP Notify Callback Function](#)
- [UDP C++ Packet Class](#)
- [UDP Echo](#)
- [UDP Send/Receive](#)
- [UDP Sockets](#)
- [UDP Sockets Via](#)

12.53.1 UDP to Serial

12.53.1.1 NetBurner UDP to Serial Bridge

A NetBurner embedded application that provides bidirectional communication between UDP network traffic and a serial port interface.

12.53.1.1.1 Overview

This application acts as a bridge between UDP network packets and serial communication, enabling remote devices to communicate with serial-connected equipment over a network. It's particularly useful for:

- Remote monitoring and control of serial devices
- Network-enabling legacy serial equipment
- Creating distributed serial communication systems
- Debugging and testing serial protocols over network

12.53.1.1.2 Features

- **Bidirectional Communication:** Data flows seamlessly between UDP and serial interfaces
- **Web-based Configuration:** Easy setup through browser interface
- **Configurable Parameters:** Customizable baud rates, ports, and IP addresses
- **Real-time Operation:** Low-latency data transfer with `select()` based I/O
- **Remote Console:** Web-based console access for monitoring and debugging
- **Persistent Settings:** Configuration stored in non-volatile memory

12.53.1.1.3 Architecture

The application consists of two main communication paths:

12.53.1.1.3.1 UDP to Serial (UdpReaderMain Task)

- Listens for incoming UDP packets on configured port
- Forwards received data to the serial port
- Displays packet information for debugging

12.53.1.1.3.2 Serial to UDP (UserMain Task)

- Monitors serial port for incoming data using `select()`
- Packages serial data into UDP packets
- Sends packets to configured destination IP and port

12.53.1.1.4 Configuration Parameters

Parameter	Default Value	Description
UDP Listen Port	24	Local port for receiving UDP packets
UDP Destination Port	8888	Remote port for sending serial data
UDP Destination IP	192.168.1.188	Remote IP address for outgoing packets
Serial Baud Rate	115200	Serial communication speed
Serial Port	1	Physical serial port number

12.53.1.1.4.1 Available Baud Rates

- 9600
- 19200
- 38400

- 57600
- 115200
- 234000

12.53.1.1.5 File Structure

```
main.cpp           # Main application logic and UDP/Serial handlers
nvsettings.h       # Configuration class definitions
htmlvars.h         # Web interface variable declarations
index.html         # Web configuration interface
style.css          # Web interface styling
```

12.53.1.1.6 Hardware Configuration

- **Data Serial Port:** Port 1 (configurable baud rate)
- **Debug Serial Port:** Port 0 (115200 baud, fixed).
- **Serial Parameters:** 8 data bits, 1 stop bit, no parity

Note

Debug serial port can be viewed on the console.html web page.

12.53.1.1.7 Network Configuration

The application automatically:

- Waits for [DHCP](#) network configuration
- Starts HTTP server on port 80
- Enables remote console access via web browser

12.53.1.1.8 Web Interface

Access the configuration interface by navigating to [http://\[device-ip\]/](http://[device-ip]/) in a web browser. The interface allows real-time modification of:

- UDP listen port
- Destination IP address and port
- Serial baud rate

Visit [http://\[device-ip\]/console.html](http://[device-ip]/console.html) for real-time console output and debugging information.

12.53.1.1.9 Usage Examples

12.53.1.1.9.1 Example 1: Serial Device Monitoring

Configure the bridge to receive data from a serial sensor and forward it to a network monitoring system:

- Set destination IP to your monitoring server
- Configure appropriate baud rate for your sensor
- Monitor data flow through the web console

12.53.1.1.9.2 Example 2: Remote Serial Control

Send commands to a serial device from a network application:

- Configure UDP listen port (e.g., 24)
- Send UDP packets containing serial commands to the bridge
- Commands are automatically forwarded to the serial device

12.53.1.1.10 Protocol Details

12.53.1.1.10.1 UDP Packet Structure

- **Source Port:** 8889 (fixed for outgoing packets)
- **Destination Port:** Configurable
- **Data:** Serial data plus null terminator and padding bytes
- **Padding:** 4 additional zero bytes added to prevent packet filtering

12.53.1.1.10.2 Serial Configuration

- **Data Bits:** 8
- **Stop Bits:** 1
- **Parity:** None
- **Flow Control:** None

12.53.1.1.11 Debugging and Monitoring

The application provides debugging output:

- UDP packet reception notifications with source information
- Serial data transmission confirmations
- Configuration parameter display on startup
- Real-time data flow visualization

Note

A python program is included for UDP packet receive on a host computer: ReceiveUDP.py.

All debug information is available through:

- Serial console (UART0 at 115200 baud)
- Web console interface ([http://\[device-ip\]/console.html](http://[device-ip]/console.html))

12.53.1.1.12 Building and Deployment

This application is designed for the NetBurner embedded platform and requires:

- NetBurner NNDK (NetBurner Network Development Kit)
- Compatible NetBurner hardware module
- Network connectivity (Ethernet)
- Serial interface hardware

12.53.1.1.13 Technical Notes

- Uses NetBurner RTOS for multitasking
- Implements I/O with `select()` for optimal performance
- Configuration changes require restart
- UDP packets with minimal data include padding to prevent firewall filtering
- Thread-safe operation with separate tasks for each communication direction

12.53.1.1.14 Troubleshooting

12.53.1.1.14.1 Common Issues

1. **No UDP Traffic:** Check firewall settings and network connectivity
2. **Serial Communication Problems:** Verify baud rate and wiring
3. **Configuration Not Saved:** Ensure proper form submission through web interface

12.53.1.1.14.2 Debug Steps

1. Monitor web console for real-time status
2. Verify network configuration ([DHCP](#) assignment)
3. Test serial port with external terminal program
4. Use network packet capture tools to verify UDP traffic

12.53.2 UDP Notify Callback Function

12.53.2.1 UDP Notify Example

12.53.2.1.1 Overview

This application demonstrates how to use callback functions to handle incoming UDP packets in a NetBurner NNDK (NetBurner Network Development Kit) environment. The example implements an asynchronous UDP packet receiver that automatically processes incoming data through a registered callback function.

12.53.2.1.2 Application Details

Application Name:** UDP Notify Example

Listen Port:** 2121

Protocol:** UDP (User Datagram Protocol)

12.53.2.1.3 Key Features

- **Callback-based UDP handling:** Uses a notification callback function instead of polling
- **Automatic packet processing:** Incoming UDP packets trigger the callback function automatically
- **Network stack initialization:** Includes proper [DHCP](#) client setup and network interface management
- **Packet validation:** Validates received packets before processing
- **Diagnostic support:** Includes system diagnostics for debugging

12.53.2.1.4 How It Works

12.53.2.1.4.1 Network Initialization

1. Initializes the network stack
2. Enables system diagnostics
3. Waits for active network connection (5 second timeout)
4. Waits for [DHCP](#) address assignment to complete
5. Displays the listening IP address and port

12.53.2.1.4.2 UDP Packet Handling

1. Creates a FIFO (First-In-First-Out) buffer for incoming packets
2. Registers the UDP port with a callback function using `RegisterUDPFifoWithNotify()`
3. When packets arrive, the callback function `UdpListenCallback()` is automatically invoked
4. The callback processes the packet and displays information about the sender and data

12.53.2.1.4.3 Callback Function Behavior

The `UdpListenCallback()` function:

- Creates a `UDPPacket` object from the FIFO
- Validates the received packet
- Extracts packet size and source IP address
- Displays the packet data using `ShowData()`
- Handles invalid packets gracefully

12.53.2.1.5 Technical Components

12.53.2.1.5.1 Key Functions

- `RegisterUDPFifoWithNotify()`: Registers UDP port with callback notification
- `UDPPacket`: Class for handling UDP packet data
- `OS_FIFO`: Operating system FIFO for packet queuing
- `WaitForActiveNetwork()`: Waits for network connectivity
- `GetInterfaceBlock()`: Retrieves network interface information

12.53.2.1.5.2 Callback Function Signature

```
void UdpListenCallback(OS_FIFO *pfifo, uint16_t port)
```

The callback function must match the typedef:

```
typedef void (udp_data_notify)(OS_FIFO *pfifo, uint16_t port);
```

12.53.2.1.6 Usage

1. **Compile and deploy** the application to a NetBurner device
2. **Connect the device** to your network (`DHCP` will assign an IP address)
3. **Monitor the console** output to see the assigned IP address
4. **Send UDP packets** to the displayed IP address on port 2121
5. **View the output** as the callback function processes and displays received packets

12.53.2.1.7 Example Output

```
Application: UDP Notify Example
```

```
NNDK Revision: [version]
```

```
Listening for UDP packets on 192.168.1.100: 2121
```

```
UdpListenCallback()
```

```
Received a UDP packet with 15 bytes from : 192.168.1.50
```

```
[Packet data display]
```

12.53.2.1.8 Differences from Standard UDP Handling

This example uses **callback-based notification** rather than polling:

- **Standard approach:** Use `RegisterUDPFifo()` and manually check for data
- **Notification approach:** Use `RegisterUDPFifoWithNotify()` with automatic callback execution

12.53.2.1.9 Requirements

- NetBurner NNDK development environment
- NetBurner-compatible hardware device
- Network connection with **DHCP** server
- UDP client application for testing (can send packets to port 2121)

12.53.2.1.10 Notes

- The application runs in an infinite loop, continuously listening for UDP packets
- System diagnostics are enabled for development/debugging purposes
- Remove diagnostic features for production deployment
- The callback function executes in the context of the network stack
- Packet timeout can be configured by replacing the 0 parameter in `UDPPacket` constructor

12.53.3 UDP C++ Packet Class

This application will send/receive UDP packets with another host on a network, such as a PC. Use the MTTTY serial port program to access the menu and prompts to specify the destination IP address and port number. NetBurner supplies an API for handling UDP as a C++ Class using `UDPPacket`, or you can use a UDP sockets API (see UDP socket example).

For an external UDP host you can use the NetBurner Java example, or the NetBurner UDP terminal program.

12.53.4 UDP Echo

12.53.4.1 UDP C++ Packet Class Example

12.53.4.1.1 Overview

This application demonstrates how to send and receive UDP packets using NetBurner's C++ `UDPPacket` class. It provides a simple interactive interface for UDP communication between a NetBurner device and another host on the network.

12.53.4.1.2 Description

This example program will receive a UDP packet from another device or host computer and send a response. To run the example, connect a serial port from your PC to the debug serial port on your NetBurner device and run a terminal program such as MTTTY. On the PC, run the NetBurner UDP Terminal (be sure to set the IP address and port numbers to match). Send data from the NetBurner UDP Terminal, and observe that the packet is received in MTTTY. Then observe the response from the device in the NetBurner UDP Terminal.

The application creates a bidirectional UDP communication system that:

- Accepts user input for destination IP address and port configuration
- Sends UDP packets containing user-entered text data
- Receives and displays incoming UDP packets from remote hosts
- Uses NetBurner's `UDPPacket` C++ class for simplified UDP handling

12.53.4.1.3 Features

- **Interactive Configuration:** Prompts user to specify destination IP address and port number via serial console
- **Dual Operation:** Simultaneously sends and receives UDP packets on the same port
- **Real-time Display:** Shows received packet data with source information
- **Multi-tasking:** Uses separate tasks for sending and receiving operations
- **Network Ready:** Waits for active network connection ([DHCP](#)) before starting

12.53.4.1.4 Architecture

12.53.4.1.4.1 Tasks

1. UserMain Task:

- Initializes network stack
- Handles user input and configuration
- Sends outgoing UDP packets
- Creates the UDP reader task

2. UdpReaderTask:

- Monitors specified port for incoming packets
- Processes and displays received data
- Uses FIFO queue for packet management

12.53.4.1.4.2 Key Components

- **UDPPacket Class:** NetBurner's C++ wrapper for UDP operations
- **OS_FIFO:** Queue system for managing incoming packets
- **RegisterUDPFifo():** Registers port for UDP packet reception
- **Serial Interface:** MTTTY terminal for user interaction

12.53.4.1.5 Usage

1. **Setup:** Connect to the device using MTTTY serial port program
2. **Network:** Ensure device has active network connection ([DHCP](#))
3. **Configuration:**
 - Enter destination IP address when prompted
 - Enter port number for both sending and receiving
4. **Operation:**
 - Type messages and press Enter to send UDP packets
 - Incoming packets are automatically displayed with source information

12.53.4.1.6 Network Requirements

- Active Ethernet connection
- [DHCP](#) server for automatic IP configuration (or manual IP setup)
- Network accessibility to destination host
- Open UDP port for bidirectional communication

12.53.4.1.7 Compatible Hosts

The application can communicate with:

- NetBurner Java UDP examples
- NetBurner UDP terminal program
- Any standard UDP client/server application
- Custom UDP applications on PC or other devices

12.53.4.1.8 Technical Details

- **API:** Uses NetBurner's [UDPPacket](#) C++ class (alternative to UDP sockets API)
- **Port Configuration:** Same port used for both transmission and reception
- **Data Format:** Text-based with null termination
- **Task Priority:** Reader task runs at higher priority than main task
- **Timeout Support:** Configurable timeout for packet reception (currently set to blocking)

12.53.4.1.9 Development Notes

- System diagnostics are enabled (consider removing for production)
- Application waits up to 5 seconds for network activation
- Uses standard NetBurner NNDK (NetBurner Network Development Kit)
- Includes proper error handling and packet validation

12.53.5 UDP Send/Receive

12.53.5.1 UDP Send/Receive Example

A NetBurner embedded application that demonstrates bidirectional UDP communication between a NetBurner device and a host computer or another networked device.

12.53.5.1.1 Overview

This application creates a UDP server that can simultaneously send and receive UDP packets on the same port. It features a multi-threaded design with one thread dedicated to receiving packets and displaying them via the debug serial port, while the main thread handles user input and sends UDP packets to a specified destination.

12.53.5.1.2 Features

- **Bidirectional UDP Communication:** Send and receive UDP packets on the same port
- **Multi-threaded Design:** Separate receive thread for non-blocking packet handling
- **Interactive Configuration:** Runtime configuration of port number and destination IP
- **Debug Output:** Received packets displayed with hex dump formatting
- **DHCP Support:** Automatic network configuration with [DHCP](#)

12.53.5.1.3 Hardware Requirements

- NetBurner embedded device with network capability
- Serial connection to PC for debugging and user interaction
- Network connection (Ethernet)

12.53.5.1.4 Software Requirements

- NetBurner NNDK (NetBurner Network Development Kit)
- Terminal program (such as MTTTY) for serial communication
- UDP client application on PC (such as NetBurner UDP Terminal)

12.53.5.1.5 How It Works

12.53.5.1.5.1 Application Flow

1. **Initialization:** Network stack initializes and waits for **DHCP** address assignment
2. **Configuration:** User prompted to enter:
 - UDP port number (used for both sending and receiving)
 - Destination IP address for outgoing packets
3. **Task Creation:** Separate receive task created with higher priority than main task
4. **Operation:**
 - Main thread waits for user input and sends UDP packets
 - Receive thread continuously monitors for incoming packets

12.53.5.1.5.2 Threading Architecture

- **Main Thread:** Handles user input from serial terminal and sends UDP packets
- **Receive Thread:** Listens for incoming UDP packets using OS_FIFO mechanism
- **Priority:** Receive thread runs at higher priority than main thread

12.53.5.1.6 Usage

12.53.5.1.6.1 Setup

1. Connect NetBurner device to network
2. Connect serial cable between PC and NetBurner debug port
3. Open terminal program (MTTTY) and connect to serial port
4. Compile and flash the application to the NetBurner device

12.53.5.1.6.2 Operation

1. **Start Application:** Reset or power on the NetBurner device
2. **Wait for Network:** Application waits up to 5 seconds for **DHCP** address
3. **Configure Port:** Enter the UDP port number when prompted
4. **Configure Destination:** Enter the destination IP address
5. **Send Data:** Type messages and press Enter to send UDP packets
6. **Receive Data:** Incoming packets automatically displayed with source IP and hex dump

12.53.5.1.6.3 Example Session

```

Application: UDP Send/Receive Example
NNDK Revision: [Version]
Enter the UDP port number (will be used for send & receive): 1234
Enter the destination IP Address: 192.168.1.100
Listening/Sending on UDP Port 1234, Sending to IP address: 192.168.1.100
Enter data and hit return to send.
Hello World
Sent "Hello World" to 192.168.1.100:1234

Received UDP packet with 12 bytes from: 192.168.1.100
48 65 6C 6C 6F 20 57 6F 72 6C 64 00   Hello World.

```

12.53.5.1.7 Key Components

12.53.5.1.7.1 UDP Packet Handling

- Uses NetBurner UDP C++ class for packet management
- OS_FIFO mechanism for packet queuing
- Automatic packet validation and parsing

12.53.5.1.7.2 Network Functions

- `RegisterUDPFifo()`: Associates FIFO with UDP port for packet reception
- `UDPPacket`: Class for constructing and sending UDP packets
- `AsciiToIp()`: Converts string IP address to IPADDR format

12.53.5.1.7.3 System Functions

- `OSTaskCreatewName()`: Creates named threads with specified stack and priority
- `WaitForActiveNetwork()`: Blocks until network interface is ready
- `ShowData()`: Utility function for hex dump display

12.53.5.1.8 Configuration Options

- **Port Number**: Any valid UDP port (1-65535)
- **Destination IP**: Any reachable IP address on the network
- **Timeout**: Receive timeout (currently set to wait forever)
- **Stack Size**: Configurable via USER_TASK_STK_SIZE

12.53.5.1.9 Testing

Use the NetBurner UDP Terminal application on your PC:

1. Set the IP address to match your NetBurner device
2. Set the port number to match your configuration
3. Send messages from both PC and NetBurner device
4. Observe bidirectional communication

12.53.5.1.10 Notes

- Application uses the same port for both sending and receiving
- Received packets include source IP address identification
- System diagnostics are enabled (consider removing for production)
- **DHCP** timeout is set to 5 seconds
- Input buffer limited to 80 characters

12.53.6 UDP Sockets

12.53.6.1 UDP Sockets Example Application

12.53.6.1.1 Overview

This NetBurner application demonstrates bidirectional UDP communication using the UDP Sockets API. The application can send and receive UDP packets with another host on a network, making it useful for network communication testing and development.

12.53.6.1.2 Features

- **Bidirectional Communication:** Send and receive UDP packets simultaneously
- **Interactive Configuration:** Runtime configuration of destination IP and port
- **Multi-threaded:** Separate task for receiving packets to ensure real-time processing
- **Serial Interface:** Uses MTTTY serial port program for user interaction
- **Network Stack Integration:** Built on NetBurner's networking framework

12.53.6.1.3 Architecture

The application consists of two main components:

12.53.6.1.3.1 Main Task (UserMain)

- Initializes the network stack and waits for [DHCP](#)
- Prompts user for destination IP address and port number
- Creates a UDP socket for sending packets
- Handles user input and sends UDP packets
- Runs at standard priority

12.53.6.1.3.2 Receive Task (UdpReceiveTask)

- Monitors specified port for incoming UDP packets
- Creates a separate UDP socket for receiving
- Processes incoming packets and displays their contents
- Runs at higher priority than main task for responsive packet handling

12.53.6.1.4 Usage

12.53.6.1.4.1 Setup

1. Connect to the device using MTTTY serial port program
2. Ensure the device is connected to a network with [DHCP](#)

12.53.6.1.4.2 Runtime Configuration

1. When prompted, enter the destination IP address of the UDP server
2. Enter the port number (used for both sending and receiving)
3. The application will create sockets and start the receive task

12.53.6.1.4.3 Operation

- Type messages and press Enter to send UDP packets
- Incoming packets are automatically displayed with source information
- The application runs continuously until reset

12.53.6.1.4.4 Example Session

Application: UDP Sockets Example
NNDK Revision: [version]

Enter the UDP Server destination IP address: 192.168.1.100
Enter the source/destination port number: 12345

Sending/Receiving with host 192.168.1.100: 12345
UdpReceiveTask monitoring port 12345
Listening for UDP packets on port 12345

Enter data and hit enter to send.
Hello World

Sending "Hello World" using UDP to 192.168.1.100 : 12345

Received a UDP packet with 11 bytes from : 192.168.1.100
Response message

12.53.6.1.5 Technical Details

12.53.6.1.5.1 Dependencies

- NetBurner NNDK (NetBurner Network Development Kit)
- Standard C libraries: `stdlib.h`, `string.h`, `stdio.h`
- NetBurner specific: `init.h`, `system.h`, `udp.h`, `utils.h`

12.53.6.1.5.2 Socket Management

- **Transmit Socket:** Created using `CreateTxUdpSocket()`
- **Receive Socket:** Created using `CreateRxUdpSocket()`
- Uses standard BSD socket functions: `sendto()` and `recvfrom()`

12.53.6.1.5.3 Task Management

- Main task priority: `MAIN_PRIO`
- Receive task priority: `MAIN_PRIO - 1` (higher priority)
- Stack allocation: `USER_TASK_STK_SIZE` for receive task

12.53.6.1.5.4 Buffer Limitations

- Maximum message size: 80 characters
- Input buffer: 80 bytes for user input
- Receive buffer: 80 bytes for incoming packets

12.53.6.1.6 Testing with External Applications

The application can be tested with:

- **NetBurner Java Example:** Companion Java application for UDP testing
- **NetBurner UDP Terminal Program:** Specialized terminal for UDP communication
- **Standard UDP Tools:** Any application capable of sending/receiving UDP packets

12.53.6.1.7 Alternative API

Note: NetBurner also provides a C++ Class-based API using [UDPPacket](#) for object-oriented UDP handling.

12.53.6.1.8 Error Handling

The application includes error checking for:

- UDP socket creation failures
- Network initialization issues
- [DHCP](#) timeout scenarios

Failed socket creation will cause the application to enter an infinite loop, requiring a system reset.

12.53.6.1.9 Development Notes

- System diagnostics are enabled for debugging (should be removed for production)
- 5-second timeout for network initialization
- Null-termination is manually added to received packets
- The application uses NetBurner's proprietary IP address formatting ([I](#))

12.53.7 UDP Sockets Via

12.53.7.1 UDP Sockets Via Example

12.53.7.1.1 Overview

This NetBurner application demonstrates UDP socket communication using the NetBurner Development Kit (NNDK). The application creates multiple UDP sockets across different network interfaces and allows bidirectional communication with other hosts on the network.

12.53.7.1.2 Features

- **Multiple Socket Types:** Creates receive-only (RX), transmit-only (TX), and bidirectional (RX/TX) UDP sockets
- **Multi-Interface Support:** Automatically creates sockets for all available network interfaces
- **Port Management:** Uses sequential port numbering starting from port 5000
- **Select-based Receiving:** Demonstrates non-blocking UDP packet reception using the [select\(\)](#) system call
- **Interactive Menu:** Provides a simple command interface via serial terminal

12.53.7.1.3 Application Functionality

12.53.7.1.3.1 Socket Creation

The application creates three types of UDP sockets for each interface:

1. **RX Socket:** Receive-only socket for incoming UDP packets
2. **RX/TX Socket:** Bidirectional socket for both sending and receiving
3. **TX Socket:** Transmit-only socket for outgoing UDP packets

12.53.7.1.3.2 Interface Detection

The program automatically detects all available network interfaces and creates sockets for:

- All interfaces (general sockets)
- Specific interface numbers
- Specific IP addresses

12.53.7.1.3.3 Port Assignment

Starting from port 5000, the application assigns ports sequentially:

- Port N: RX socket
- Port N+1: RX/TX socket
- Port N+2: TX socket

12.53.7.1.4 Usage

12.53.7.1.4.1 Setup

1. Compile and flash the application to your NetBurner device
2. Connect to the device using MTTTY serial port program
3. Wait for the network to initialize and obtain a [DHCP](#) address

12.53.7.1.4.2 Operation

1. **Enter Destination IP:** When prompted, enter the IP address of the UDP host you want to communicate with
2. **View Socket Information:** The application displays a table showing:


```

RX  | RXTX| TX  | On
5000| 5001| 5002|All
5003| 5004| 5005|ifn1 on 192.168.1.100
5006| 5007| 5008|IP:192.168.1.100
```
3. **Send Test Packets:** Press 'T' or 't' to transmit test messages to all TX-capable sockets
4. **Receive Packets:** The application automatically displays received UDP packets with source information

12.53.7.1.4.3 Commands

- **T or t:** Transmit test packets to all transmit-capable sockets

12.53.7.1.5 Code Structure

12.53.7.1.5.1 Main Components

- `AddSocketsIfn()`: Creates sockets for a specific interface number
- `AddSocketsIP()`: Creates sockets for a specific IP address
- `TransmitAll()`: Sends test messages through all transmit sockets
- `UserMain()`: Main application loop handling initialization and packet processing

12.53.7.1.5.2 Key Variables

- `rx_fd[]`: Array of receive-capable socket file descriptors
- `tx_fd[]`: Array of transmit-capable socket file descriptors
- `using_port`: Current port number for socket creation
- `nrxfd, ntxfd`: Counters for RX and TX socket arrays

12.53.7.1.6 Testing

12.53.7.1.6.1 External UDP Hosts

You can test the application using:

- NetBurner Java UDP example application
- NetBurner UDP terminal program
- Any standard UDP client/server application

12.53.7.1.6.2 Packet Reception

When UDP packets are received, the application displays:

- Number of bytes received
- Source IP address
- Local port number
- Remote port number
- Hexadecimal dump of packet data

12.53.7.1.7 API Information

This example demonstrates the NetBurner UDP Sockets API functions:

- `CreateRxUdpSocket()`
- `CreateTxUdpSocket()`
- `CreateRxTxUdpSocket()`
- `CreateRxUdpSocketVia()`
- `CreateTxUdpSocketVia()`
- `CreateRxTxUdpSocketVia()`

For C++ applications, NetBurner also provides a `UDPPacket` class API for object-oriented UDP handling.

12.53.7.1.8 Notes

- The application enables system diagnostics for debugging (should be disabled in production)
- All sockets use UDP protocol for lightweight, connectionless communication
- The `select()` mechanism allows efficient handling of multiple socket file descriptors
- Socket creation spans all detected network interfaces automatically

12.54 VLAN

12.54.1 VLAN Example Application

12.54.1.1 Overview

This NetBurner application demonstrates the creation and management of multiple VLAN (Virtual Local Area Network) interfaces on a single Ethernet port. The application creates three different VLAN interfaces with different configurations and provides both web-based and serial console interfaces for monitoring network status.

To build this example MULTIHOME must be enabled in `predef.h`. Rather than change the file in `\nburn\nbrtos\include`, use the override feature and add `predef.h` to your project.

- Ethernet 0, [DHCP](#), VLAN ID 1
- Ethernet 0, [DHCP](#), VLAN ID 2
- Ethernet 0, Static IP, VLAN ID 10

The serial port interface provides an interactive menu to display interface and VLAN information, as well as other system parameters.

12.54.1.2 Features

- **Multiple VLAN Interface Creation:** Creates 3 VLAN interfaces on Ethernet 0
- **Mixed Configuration Support:** Supports both [DHCP](#) and static IP configurations
- **Interactive Serial Menu:** Command-line interface for network diagnostics
- **Web Interface:** HTTP server displaying network interface information
- **IPv4 and IPv6 Support:** Comprehensive IP address management and display
- **Real-time Network Monitoring:** Link status, [ARP](#) tables, and interface statistics

12.54.1.3 VLAN Configuration

The application automatically creates the following VLAN interfaces:

1. **VLAN ID 1:** Ethernet 0 with [DHCP](#) configuration
2. **VLAN ID 2:** Ethernet 0 with [DHCP](#) configuration
3. **VLAN ID 10:** Ethernet 0 with static IP (198.18.1.20/16)

12.54.1.4 Prerequisites

- **MULTIHOME Support:** Must be enabled in `predef.h`
- **NetBurner Platform:** Compatible NetBurner hardware platform
- **Network Infrastructure:** VLAN-capable switch or router

12.54.1.4.1 Enabling MULTIHOME

To enable MULTIHOME support:

1. Create a local `predef.h` file in your project directory
2. Add `#define MULTIHOME` to enable multiple interface support
3. Include any other required preprocessor definitions

This approach uses the NetBurner override feature instead of modifying the system files directly.

12.54.1.5 Serial Console Commands

The application provides an interactive serial console with the following commands:

Command	Description
A	Show ARP table
C	Show network counters
E	Dump Ethernet information
I	Show all network interfaces and IP addresses
W	Display system tick count and uptime
?	Show help menu
1-9 <vlan_id>	Set VLAN tag on interface (1-9)

12.54.1.5.1 Example Usage

```

I          # Show all interfaces
A          # Display ARP table
1 100     # Set interface 1 to VLAN 100
W          # Show system uptime

```

12.54.1.6 Network Interface Information

The application displays comprehensive network information including:

12.54.1.6.1 IPv4 Information

- IP Address
- Subnet Mask
- DNS Servers (Primary and Secondary)
- Gateway Address
- Auto-IP Address

12.54.1.6.2 IPv6 Information (if enabled)

- Link-local addresses
- Router-assigned addresses
- DHCP-assigned addresses
- Static addresses
- DNS servers with source information
- [DHCP](#) lease timing information

12.54.1.6.3 Physical Interface Details

- MAC Address
- Link Status (UP/DOWN)
- Link Speed (MB)
- Duplex Mode (Full/Half)

12.54.1.7 Web Interface

The application includes an HTTP server that provides:

- **Network Interface Overview:** All active interfaces and their configurations
- **Clickable IP Links:** Direct links to access the device via different IP addresses
- **Real-time Status:** Current network configuration and lease information
- **IPv6 Support:** Complete IPv6 address information with source tracking

Access the web interface by navigating to any of the device's IP addresses in a web browser.

12.54.1.8 File Structure

```
project/
  main.cpp           # Main application logic and VLAN setup
  ip_util.cpp        # IP address utility functions
  ip_util.h          # Header file for IP utilities
  index.html         # Web interface homepage
  predef.h           # Project-specific preprocessor definitions
```

12.54.1.9 Key Functions

12.54.1.9.1 VLAN Management

- `AddVlanInterface()`: Creates new VLAN interface
- `SetVlan()`: Configures VLAN tag on existing interface

12.54.1.9.2 Network Information Display

- `showIpAddressesSerial()`: Displays interface info on serial console
- `showIpAddressesWeb()`: Generates web page with interface information
- `showLinkStatus()`: Shows physical link status

12.54.1.9.3 Utility Functions

- `getTimeStringFromSeconds()`: Converts seconds to HH:MM:SS format
- `getLeaseTimeInfo()`: Formats DHCP lease timing information
- `SourceName()`: Returns human-readable IPv6 address source names

12.54.1.10 Building and Deployment

1. **Setup Project:** Create new NetBurner project
2. **Enable MULTIHOMES:** Add `predef.h` with `#define MULTIHOMES`
3. **Add Source Files:** Include all `.cpp` and `.h` files
4. **Build:** Compile using NetBurner development tools
5. **Deploy:** Flash to NetBurner hardware

12.54.1.11 Network Setup Requirements

- **VLAN-Capable Switch:** Network infrastructure must support VLAN tagging
- **Proper VLAN Configuration:** Switch ports must be configured for appropriate VLAN IDs
- **IP Address Management:** Ensure IP ranges don't conflict across VLANs

12.54.1.12 Troubleshooting

12.54.1.12.1 Common Issues

1. **MULTIHOME Not Enabled:** Verify `predef.h` contains `#define MULTIHOME`
2. **Link Status Down:** Check physical connections and cable
3. **DHCP Not Working:** Verify **DHCP** server availability on VLANs
4. **Web Interface Inaccessible:** Check IP address configuration and routing

12.54.1.12.2 Debug Commands

Use the serial console commands to diagnose issues:

- `E` - Check Ethernet hardware status
- `I` - Verify interface configuration
- `C` - Review network counters for errors
- `A` - Check **ARP** table for connectivity

12.54.1.13 Platform Compatibility

This application is designed for NetBurner embedded platforms with Ethernet capability. It supports both single and dual Ethernet configurations (commented code shows dual-port setup for platforms like the 54417).

12.54.1.14 IPv6 Support

IPv6 functionality is conditionally compiled based on the `IPV6` preprocessor definition. When enabled, the application provides:

- Multiple IPv6 address assignment methods
- **DHCP** v6 lease management
- Router advertisement processing
- Comprehensive IPv6 status reporting

12.55 Web Server

Web Server Examples:

- [Ajax Graph](#)
- [Bootstrap Hello World](#)
- [HTML File Post](#)
- [HTML File Post Task Handoff](#)
- [Flash Form](#)
- [GIF Canvas](#)
- [HTML Application File Post](#)
- [HTML Cookie](#)
- [HTML File Post Task Handoff](#)
- [HTML Form Post](#)

- [HTML Form Post 2.X Compatible](#)
- [HTML Password](#)
- [HtmlPostDateTime](#)
- [HTML Server GET Request](#)
- [HTMLVariables](#)
- [Ping From Web](#)
- [Raw Form Post 2pX](#)
- [Remote Web Console for Serial/stdout](#)
- [Serial Webserver](#)
- [Simple HTML](#)
- [TicTacToe](#)

12.55.1 Ajax Graph

12.55.1.1 NetBurner AJAX Graph Application

12.55.1.1.1 Overview

This application demonstrates how to create a dynamic web-based graph using a NetBurner embedded device. The system continuously logs sensor data from multiple sources and serves this data to a web interface through AJAX calls, allowing real-time visualization of sensor readings.

12.55.1.1.2 Features

- **Real-time Data Logging:** Continuously logs data from two simulated sensors with different sampling periods
- **Web-based Visualization:** Serves sensor data through HTTP for dynamic graph generation
- **AJAX Integration:** Provides JSON and CSV data formats for web consumption
- **Ring Buffer Implementation:** Efficient circular buffer for data storage with configurable size
- **Multiple Data Export Formats:** Supports both JSON (for AJAX) and CSV (for data export)

12.55.1.1.3 System Architecture

12.55.1.1.3.1 Core Components

1. **Data Generator** (`datagenerator.cpp/h`)
 - Simulates two sensors with different characteristics
 - Sensor 1: Period of 13 ticks, initial value 44.23
 - Sensor 2: Period of 11 ticks, initial value -10.0
 - Generates realistic sensor readings using sine wave variations with random noise
2. **Data Logger** (`datalog.cpp/h`)
 - Implements a thread-safe ring buffer for efficient data storage
 - Supports up to 1000 data points by default
 - Provides serialization methods for different output formats
 - Automatic overwriting of old data when buffer is full
3. **Web Server Interface** (`main.cpp`)
 - HTTP server integration for serving web pages and data
 - AJAX endpoints for real-time data retrieval
 - Support for both JSON and CSV data formats

12.55.1.1.3.2 Data Structure

Each logged data point contains:

- `sensorID`: Identifier for the sensor (1 or 2)
- `timeStamp`: System tick when the reading was taken
- `value`: The actual sensor reading (floating-point)

12.55.1.1.4 Web Interface Integration

The application provides several web endpoints:

12.55.1.1.4.1 AJAX Endpoints

- **Data Retrieval:** `getSensorData()` function serves sensor data in JSON format


```
{
  "entries": [
    {"id": 1, "timeStamp": 12345, "value": 44.2567},
    {"id": 2, "timeStamp": 12346, "value": -9.8734},
    ...
  ]
}
```
- **Time Synchronization:** `getHTMLTicks()` provides current system tick for time alignment

12.55.1.1.4.2 HTML Integration Points

The web interface uses the following integration methods:

- `<!--FUNCTIONCALL getHTMLTicks -->` - Embeds current system time
- `<!--CPPCALL getSensorData() -->` - Retrieves sensor data via AJAX
- JavaScript in `index.html` handles the dynamic graph generation

12.55.1.1.5 Configuration

12.55.1.1.5.1 Sensor Settings

```
#define LOG_SIZE 1000           // Maximum number of data points stored
#define SENSOR_1_PERIOD 13      // Sensor 1 sampling period (ticks)
#define SENSOR_2_PERIOD 11      // Sensor 2 sampling period (ticks)
#define SENSOR_1_INIT 44.23     // Sensor 1 initial value
#define SENSOR_2_INIT -10.0     // Sensor 2 initial value
```

12.55.1.1.5.2 System Requirements

- NetBurner embedded platform with NNDK
- Network connectivity for web server functionality
- Sufficient RAM for data logging buffer (configurable)

12.55.1.1.6 Application Flow

1. Initialization

- Network stack initialization
- HTTP server startup on port 80
- **DHCP** address acquisition
- Data log seeding with initial random data

2. Runtime Operation

- Continuous sensor data generation at different periods

- Real-time logging to ring buffer
- Web server handles AJAX requests for data retrieval
- Optional console output for debugging (press any key)

3. Data Management

- Automatic buffer management with oldest data overwritten
- Efficient serialization for web delivery
- Support for both real-time streaming and historical data export

12.55.1.1.7 Usage

12.55.1.1.7.1 Deployment

1. Compile and flash the application to a NetBurner device
2. Ensure network connectivity ([DHCP](#) or static IP)
3. Access the web interface through the device's IP address
4. The graph will automatically update with real-time sensor data

12.55.1.1.7.2 Development

The modular design allows for easy customization:

- **Adding Sensors:** Modify the data generator to support additional sensors
- **Changing Sampling Rates:** Adjust period constants for different update frequencies
- **Buffer Size:** Modify `LOG_SIZE` for different memory/history requirements
- **Data Formats:** Add new serializer functions for different export formats

12.55.1.1.8 Technical Notes

- The ring buffer implementation ensures $O(1)$ insertion and efficient memory usage
- Sensor data generation uses mathematical functions to create realistic, varying readings
- The web server integration is designed to handle multiple concurrent AJAX requests
- All timing is based on the NetBurner system tick counter for precision

12.55.1.1.9 File Structure

```
main.cpp           # Main application and web server endpoints
datalog.cpp/h      # Ring buffer implementation for data storage
datagenerator.cpp/h # Sensor simulation and data generation
readme.txt         # Original project documentation
```

12.55.2 Bootstrap Hello World

12.55.2.1 NetBurner Bootstrap HelloWorld Example

12.55.2.1.1 Overview

The Bootstrap HelloWorld Example demonstrates how to utilize Twitter Bootstrap (<http://twitter.github.io/bootstrap/>) on a NetBurner device. This example loads a simple "Hello World" web page on to the device, along with the Twitter Bootstrap framework (version 4.0).

12.55.2.1.2 Application Description

The main application (`main.cpp`) creates a minimal web server that:

- Initializes the NetBurner network stack
- Enables system diagnostics for debugging
- Starts an HTTP web server on the default port (80)
- Waits for **DHCP** network configuration
- Displays application information and NNDK revision via serial output
- Runs continuously with a 1-second delay loop

12.55.2.1.3 Key Features

12.55.2.1.3.1 Embedded Application (`main.cpp`)

- **Network Stack Initialization:** Automatically configures network connectivity
- **Web Server:** Built-in HTTP server for serving web content
- **DHCP Support:** Automatic IP address assignment
- **System Diagnostics:** Debug information output
- **Real-time Operation:** Uses NBRTOS for task scheduling

12.55.2.1.3.2 Web Interface (`index.html`)

- **Bootstrap Framework:** Modern, responsive web design
- **Image Carousel:** Interactive slideshow showcasing NetBurner products
- **Responsive Design:** Adapts to different screen sizes
- **Professional Styling:** Clean, marketing-focused layout

12.55.2.1.4 Hardware Compatibility

Based on the web interface images, this application supports NetBurner development boards including:

- NANO54415 series
- Serial2Ethernet modules
- Other NetBurner embedded modules

12.55.2.1.5 Development Environment

- **IDE:** NetBurner's customized Eclipse-based development environment
- **Platform Support:** Both Mac and PC development
- **Build System:** One-click compile, link, and load process
- **Web Assets:** HTML, CSS, and JavaScript files served directly from the embedded device

12.55.2.1.6 File Structure

```
project/
  main.cpp           # Main application source code
  index.html         # Web interface homepage
  assets/            # Web assets (referenced in HTML)
    css/             # Bootstrap CSS files
    js/              # JavaScript libraries
    img/             # Product images and screenshots
```

12.55.2.1.7 Getting Started

1. **Development Setup:** Install NetBurner's Eclipse-based IDE
2. **Hardware Connection:** Connect your NetBurner module to the network
3. **Project Build:** Use the IDE's one-click build process
4. **Deployment:** Load the application onto your NetBurner device
5. **Access:** Navigate to the device's IP address in a web browser

12.55.2.1.8 Application Flow

1. Device boots and initializes the network stack
2. HTTP server starts on port 80
3. Device acquires IP address via [DHCP](#)
4. Application information is displayed via serial output
5. Web interface becomes accessible at the device's IP address
6. Main application loop continues indefinitely

12.55.2.1.9 Web Interface Features

- **Welcome Message:** "Hello World. It's that easy."
- **Product Showcase:** Carousel displaying NetBurner development tools
- **Development Information:** Details about the Eclipse-based IDE
- **Cross-platform Support:** Information about Mac and PC compatibility

12.55.2.1.10 Technical Details

- **Operating System:** NetBurner RTOS (NBRTOS)
- **Network Protocol:** HTTP/TCP
- **Frontend Framework:** Bootstrap 4.x
- **JavaScript Libraries:** jQuery 3.3.1, Bootstrap Bundle
- **Styling:** Bootstrap grid system and components

12.55.2.1.11 Use Cases

This example serves as a foundation for:

- IoT device web interfaces
- Embedded system monitoring dashboards
- Remote device configuration portals

12.55.2.1.12 Next Steps

This basic example can be extended with:

- REST API endpoints
- Real-time data visualization
- Device configuration interfaces
- Sensor data collection and display
- Remote control capabilities

12.55.3 HTML File Post

12.55.3.1 File Post with Multipart Form

12.55.3.1.1 Overview

This NetBurner application provides a web-based file upload interface that allows users to upload files to the NetBurner device via HTTP POST requests. The uploaded files are processed and displayed back to the user in a web browser, with options for both text and binary data visualization.

12.55.3.1.2 Features

- **Web-based file upload:** Simple HTML form interface for selecting and uploading files
- **Multipart form data support:** Handles file uploads using the standard multipart/form-data encoding
- **File processing:** Reads and processes uploaded files up to 10KB in size
- **Dual display modes:**
 - Text mode: Shows readable characters with HTML entity encoding
 - Binary mode: Shows both hexadecimal and ASCII representation of file data
- **Real-time feedback:** Console output showing POST processing events

12.55.3.1.3 Architecture

12.55.3.1.3.1 Core Components

1. **main.cpp:** Application initialization and network setup
2. **formcode.cpp:** HTTP POST processing and file handling logic
3. **index.html:** Main web page with file upload form
4. **filepost.html:** Target page for form submission (handled by callback)

12.55.3.1.3.2 Key Functions

- `ProcessPostFile()`: Reads uploaded file data into memory buffer
- `ShowFileData()`: Displays file contents in web browser with formatting
- `PostCallBack()`: Handles POST events and coordinates file processing
- `UserMain()`: Initializes network stack and web server

12.55.3.1.4 Technical Details

12.55.3.1.4.1 Memory Management

- File buffer size: 10,000 bytes (10KB maximum file size)
- Uses static buffer allocation for file storage
- Automatic cleanup of file descriptors after processing

12.55.3.1.4.2 Security Features

- HTML entity encoding for special characters `<`, `>`, `&`, `"`
- Input validation and size limits
- Controlled file access through NetBurner's HTTP framework

12.55.3.1.4.3 Network Configuration

- Uses [DHCP](#) for automatic IP address assignment
- HTTP server runs on default port 80
- 5-second timeout for network initialization

12.55.3.1.5 Usage

12.55.3.1.5.1 Setup

1. Deploy the application to a NetBurner device
2. Connect the device to your network
3. Wait for [DHCP](#) address assignment
4. Note the device's IP address from console output

12.55.3.1.5.2 File Upload Process

1. Open web browser and navigate to the device's IP address
2. Select a file using the "File to process" field
3. Optionally check the binary display option (if available)
4. Click "Send File" to upload
5. View the processed file content on the results page

12.55.3.1.5.3 Supported File Types

- Text files (.txt, .html, .xml, etc.)
- Binary files (images, executables, etc.)
- Any file type up to 10KB in size

12.55.3.1.6 Display Modes

12.55.3.1.6.1 Text Mode

- Shows printable ASCII characters (0x20-0x80)
- Converts HTML special characters to entities
- Preserves line breaks with `<BR>` tags

12.55.3.1.6.2 Binary Mode

- Hexadecimal dump format (16 bytes per line)
- ASCII representation alongside hex values
- Non-printable characters shown as dots

12.55.3.1.7 Development Notes

12.55.3.1.7.1 Dependencies

- NetBurner NNDK (NetBurner Network Development Kit)
- HTTP server libraries ([http.h](#), [httppost.h](#))
- Standard I/O libraries ([iosys.h](#), [fdprintf.h](#))

12.55.3.1.7.2 Error Handling

- File size limits enforced
- Invalid file descriptors handled gracefully
- Network timeout protection during initialization

12.55.3.1.8 Console Output

The application provides diagnostic output showing:

- Application name and NNDK revision
- POST processing events (start, variable, file, end)
- Network initialization status
- System diagnostics (if enabled)

12.55.3.1.9 Limitations

- Maximum file size: 10KB
- Single file upload per request
- No persistent file storage
- Files are processed in memory only

12.55.3.1.10 Security Considerations

- File size limits prevent memory overflow
- HTML encoding prevents XSS attacks
- No file system access beyond temporary processing
- Network access controls should be implemented at infrastructure level

12.55.3.1.11 Troubleshooting

12.55.3.1.11.1 Common Issues

1. **File not uploading:** Check file size (<10KB) and network connectivity
2. **Display issues:** Verify HTML encoding and browser compatibility
3. **Network problems:** Confirm [DHCP](#) configuration and device connectivity

12.55.3.1.11.2 Debug Features

- Console logging for POST events
- System diagnostics output
- Network initialization feedback

12.55.4 HTML File Post Task Handoff

12.55.4.1 File Post Task Handoff Example

12.55.4.1.1 Overview

This application demonstrates two different methods for handling file uploads in a web server environment without blocking the main HTTP task. The example provides a web interface for file uploads and processes them using task handoff techniques to maintain server responsiveness during potentially time-consuming file operations.

12.55.4.1.2 Application Architecture

The application consists of several key components:

- **Main HTTP Server:** Handles incoming web requests
- **File Upload Handlers:** Two different approaches for processing uploaded files
- **Task Handoff System:** Prevents blocking of the main HTTP thread during file processing

12.55.4.1.3 File Upload Methods

12.55.4.1.3.1 Method 1: Buffered File Descriptor Handoff (`filepost_bufferedFD.cpp`)

This method uses a file descriptor buffer to hand off file processing to a separate task.

Key Features:**

- Uses `FDcBuf` adapter for buffered file descriptor operations
- Implements mailbox communication between HTTP and processing tasks
- Supports both binary and text file display
- Buffer size limited to 10,000 bytes
- Selective handoff based on filename (`userfile1` is processed synchronously)

Flow:**

1. HTTP task receives file upload
2. File data is read into a handoff buffer
3. File descriptor is passed to processing task via mailbox
4. Processing task handles file display and socket closure
5. Semaphore coordinates task availability

12.55.4.1.3.2 Method 2: Buffered Request Handoff (`filepost_bufferedReq.cpp`)

This method hands off the entire HTTP request to a separate processing task.

Key Features:**

- Copies the complete HTTP request structure
- Uses raw post callback handler
- Processes files in dedicated task context
- Direct request forwarding to existing callback system
- Buffer size limited to 10,000 bytes

Flow:**

1. Raw post callback intercepts HTTP request
2. Request structure is copied for handoff
3. Socket is passed to processing task via mailbox
4. Processing task uses existing callback system
5. Semaphore ensures single request processing

12.55.4.1.4 Technical Components

12.55.4.1.4.1 Synchronization Primitives

- **OS_MBOX:** Mailbox for passing socket handles between tasks
- **OS_SEM:** Semaphores for coordinating task availability
- **Timeouts:** HTTP_TIMEOUT used for non-blocking operations

12.55.4.1.4.2 Buffer Management

- **FILE_BUFFER_SIZE:** 10,000 byte limit for file processing
- **FileRxBuffer:** Main buffer for synchronous processing
- **HandoffFileRxBuffer:** Separate buffer for handoff operations

12.55.4.1.4.3 Web Interface

- **HTML Forms:** Two different forms for testing each handoff method
- **File Display:** Support for both text and binary file visualization
- **Checkbox Control:** Option to display binary files

12.55.4.1.5 File Processing Features

12.55.4.1.5.1 File Reading

- Reads uploaded files in chunks up to buffer limit
- Handles both text and binary file types
- Graceful handling of files exceeding buffer size

12.55.4.1.5.2 Display Options

- **Text Mode:** Displays file content as readable text
- **Binary Mode:** Shows binary file content (controlled via checkbox)
- **HTML Formatting:** Uses `<PRE>` tags for proper text formatting

12.55.4.1.5.3 Error Handling

- Timeout protection for task handoffs
- Graceful degradation when handoff is unavailable
- Error messages for failed file extractions

12.55.4.1.6 Usage

1. **Compile and Deploy:** Build the application using the NNDK framework
2. **Network Setup:** Application waits for [DHCP](#) address assignment
3. **Web Access:** Navigate to the device's IP address on port 80
4. **File Upload:** Use either form to upload files:
 - `filepost_bufferedFD.html` - for buffered FD method
 - `filepost_bufferedReq.html` - for buffered request method
5. **Options:** Check binary display option if needed

12.55.4.1.7 Configuration

12.55.4.1.7.1 Task Priorities

- **HttpHandoffTask_BufferedFd:** MAIN_PRIO+1
- **HttpHandoffTask_BufferedReq:** MAIN_PRIO+2

12.55.4.1.7.2 Timeouts

- **Network Wait:** 5 seconds for active network
- **Handoff Timeout:** HTTP_TIMEOUT/4 for task availability

12.55.4.1.7.3 Buffer Limits

- **Maximum File Size:** 10,000 bytes per upload
- **Read Chunk Size:** 80 bytes for display operations

12.55.4.1.8 Benefits

12.55.4.1.8.1 Performance

- Non-blocking HTTP server operation
- Concurrent file processing capability
- Responsive web interface during uploads

12.55.4.1.8.2 Flexibility

- Two different handoff approaches for various use cases
- Configurable binary/text display modes
- Selective processing based on file identification

12.55.4.1.9 File Structure

```
/
main.cpp                # Application entry point
filepost_bufferedFD.cpp  # Buffered FD handoff method
filepost_bufferedReq.cpp # Buffered request handoff method
filepost_bufferedFD.html # Web form for FD method (referenced)
filepost_bufferedReq.html # Web form for request method (referenced)
```

12.55.5 Flash Form

12.55.5.1 FlashForm Example Application

12.55.5.1.1 Overview

The FlashForm example is a NetBurner embedded web application that demonstrates three key capabilities:

1. **Dynamic HTML** - Server-side function calls embedded in HTML
2. **HTML Form Processing** - Handling POST requests from web forms
3. **User Flash Parameter Storage** - Persistent data storage in flash memory

12.55.5.1.2 Application Structure

12.55.5.1.2.1 Core Files

- `main.cpp` - Main application entry point and network initialization
- `formcode.cpp` - Web server functionality and form processing logic
- `index.html` - Web interface with dynamic content and form
- `ReadMe.txt` - Original project documentation

12.55.5.1.2.2 Main Application (`main.cpp`)

The main application initializes the NetBurner network stack and starts the web server:

- Initializes the network stack
- Enables system diagnostics
- Starts HTTP server on default port 80
- Waits for [DHCP](#) network configuration
- Displays application information and NNDK revision
- Enters main loop with 1-second delays

12.55.5.1.3 Features

12.55.5.1.3.1 Dynamic HTML Content

The application uses special HTML comments to embed server-side function calls:

```
<!--FUNCTIONCALL DoMessageName -->
<!--FUNCTIONCALL DoMessageBody -->
```

These tags are replaced with dynamic content when the page is served:

- `DoMessageName()` - Displays the stored user name
- `DoMessageBody()` - Displays the stored message

12.55.5.1.3.2 Form Processing

The web interface includes a form that allows users to:

- Enter their name (up to 40 characters)
- Enter a message (up to 128 characters)
- Submit the form to store the data in flash memory

Form data is processed by the `PostCallBack()` function which handles:

- `eStartingPost` - Initialize temporary storage
- `eVariable` - Process each form field (name and comment)
- `eEndOfPost` - Save data to flash and redirect to index page

12.55.5.1.3.3 Flash Storage

User data is stored persistently in flash memory using a structured approach:

```
struct MyOwnDataStore
{
    uint32_t verify_key;    // Verification key (0x48666050)
    char name[40];          // User name
    char msg[128];          // User message
};
```

The application uses NetBurner's user parameter storage functions:

- [GetUserParameters\(\)](#) - Read stored data (read-only)
- [SaveUserParameters\(\)](#) - Write data to flash memory

12.55.5.1.4 Security Features

- **Data Validation** - Uses a verification key to ensure data integrity
- **HTML Escaping** - Uses `writesafestring()` to prevent HTML injection
- **Buffer Protection** - Uses `strncpy()` with size limits to prevent buffer overflows

12.55.5.1.5 Usage

1. **Setup:** Deploy the application to a NetBurner device
2. **Access:** Open a web browser and navigate to the device's IP address
3. **View:** See any previously stored name and message
4. **Update:** Use the form to enter a new name and message
5. **Save:** Click "Save New Message" to store the data in flash

12.55.5.1.6 Technical Details

12.55.5.1.6.1 Network Configuration

- Uses [DHCP](#) for automatic IP configuration
- HTTP server runs on port 80
- Waits up to 5 seconds for network activation

12.55.5.1.6.2 Memory Management

- Static storage for form processing to maintain state across callbacks
- Flash storage persists data across power cycles
- Verification key ensures data integrity

12.55.5.1.6.3 Error Handling

- Displays "?" for invalid/missing name data
- Displays "No Message Stored" for invalid/missing message data
- Reports success/failure of flash programming operations

12.55.5.1.7 Development Notes

- Functions exposed to HTML must be declared as `extern "C"` to prevent C++ name mangling
- The `PostCallBack` function uses static variables to maintain state across multiple calls
- As of NetBurner 3.x release, the configuration server/storage system is recommended for new applications

12.55.5.1.8 Dependencies

- NetBurner NNDK (NetBurner Network Development Kit)
- Standard NetBurner libraries: [http.h](#), [iosys.h](#), [system.h](#), [httppost.h](#)

12.55.6 GIF Canvas

12.55.6.1 GIF Canvas Example

A web-based application that generates dynamic GIF images through a web interface using embedded HTTP server functionality.

12.55.6.1.1 Overview

This application demonstrates real-time GIF image generation on an embedded system. Users can configure various parameters through a web form to create different types of GIF images including solid boxes, gradients, text overlays, and random patterns.

12.55.6.1.2 Features

- **Web-based Interface:** HTML form for configuring image parameters
- **Dynamic GIF Generation:** Real-time creation of GIF images based on user input
- **Multiple Image Types:**
 - Solid filled boxes with borders
 - Gradient-filled boxes
 - Text overlay on colored backgrounds
 - Random pattern generation with transparency support
- **Customizable Parameters:**
 - Canvas dimensions (width/height)
 - Fill colors (RGB values)
 - Border colors (RGB values)
 - Text colors (RGB values)

12.55.6.1.3 Architecture

12.55.6.1.3.1 Core Components

1. **Web Server:** Built-in HTTP server handling GET/POST requests
2. **Image Generation:** DrawImageObject class for creating pixel-based graphics
3. **GIF Compression:** Custom miGIF run-length encoding compression
4. **Font Rendering:** Multiple font sizes (Tiny, Small, Medium, Large, Giant)
5. **Form Processing:** URL parameter parsing and form data handling

12.55.6.1.3.2 Key Files

- `main.cpp`: Application entry point and network initialization
- `web.cpp`: HTTP request handlers and GIF generation functions
- `drawimage.cpp/h`: Core graphics drawing functionality
- `gifCompress.cpp/h`: GIF compression implementation
- `webFormValues.cpp/h`: Form data parsing and URL encoding
- `fontdata.c`: Bitmap font definitions
- `index.html`: Web interface form

12.55.6.1.4 Technical Details

12.55.6.1.4.1 Image Generation Process

1. User submits form with color and dimension parameters
2. Parameters are encoded in URL query string format
3. Web handlers parse parameters and create DrawImageObject
4. Graphics operations (lines, boxes, text) are performed
5. Image is compressed using run-length encoding
6. GIF data is streamed directly to HTTP response

12.55.6.1.4.2 Color Management

- Supports up to 256 colors per image
- RGB color values (0-255 range)
- Transparency support for overlays
- Dynamic color palette generation for gradients

12.55.6.1.4.3 Font System

- Bitmap-based font rendering
- 5 different font sizes available
- Monospace character layout
- Custom font data format with bit-packed character definitions

12.55.6.1.5 Usage

1. Connect to the device's web server (default port 80)
2. Navigate to the main interface
3. Configure canvas dimensions and colors
4. Click "Update Canvas" to apply changes
5. View generated GIF images in the preview grid

12.55.6.1.6 Network Requirements

- DHCP-enabled network for automatic IP assignment
- Standard HTTP port 80 access
- Web browser with HTML form and image support

12.55.6.1.7 Compression Technology

Uses miGIF compression algorithm:

- Run-length encoding instead of LZW to avoid patent issues
- Optimized for images with solid color regions
- Produces GIF-compatible output streams
- Efficient for line art, text, and geometric shapes

12.55.6.1.8 Memory Management

- Dynamic buffer allocation for image data
- Efficient pixel storage and manipulation
- Streaming output to minimize memory usage
- Automatic cleanup of image resources

12.55.7 HTML Application File Post

12.55.7.1 NetBurner Application Update Server

12.55.7.1.1 Overview

This application provides a web-based interface for updating embedded application code remotely. It creates an HTTP server that accepts file uploads to update the device's application firmware through a web browser.

12.55.7.1.2 Features

- **Web-based Update Interface:** Hosts a web page (`Apppost.html`) that allows users to upload application binary files
- **Real-time Processing:** Handles file uploads and immediately processes them for application updates
- **Authentication Support:** Optional username/password protection for the update process
- **Automatic Reboot:** Automatically reboots the device after successful updates
- **Status Feedback:** Provides immediate feedback on upload success or failure

12.55.7.1.3 Application Flow

1. **Initialization:** The application starts the HTTP server and waits for network connectivity
2. **File Upload:** Users access the web interface to upload application binary files
3. **Processing:** The uploaded file is processed through `ReadBinaryApplicationCodeFromStream()`
4. **Response:** The system provides feedback on the update status
5. **Reboot:** On successful update, the device automatically reboots after a 5-second delay

12.55.7.1.4 Security Options

12.55.7.1.4.1 Basic Mode (Default)

- No authentication required
- Direct access to the update interface

12.55.7.1.4.2 Authentication Mode

To enable authentication, uncomment the following line in the source code:

```
#define I_WANT_TO_AUTHENTICATE_UPDATE
```

When authentication is enabled:

- **Username:** `up`
- **Password:** `date`
- **Realm:** `NetBurner`
- Users must provide credentials before accessing the update interface

12.55.7.1.5 Technical Details

12.55.7.1.5.1 Dependencies

- NetBurner RTOS ([nbrtos.h](#))
- HTTP POST handling ([httppost.h](#))
- Stream update functionality ([StreamUpdate.h](#))
- System initialization and HAL components

12.55.7.1.5.2 Callback Events

The application handles the following POST events:

- `eStartingPost`: Initializes the update process
- `eVariable`: Processes form variables (currently unused)
- `eFile`: Handles the uploaded binary file
- `eEndOfPost`: Completes the update and provides status response

12.55.7.1.5.3 Network Configuration

- Uses [DHCP](#) for network address assignment
- Runs HTTP server on default port 80
- Includes system diagnostics (should be disabled for production)

12.55.7.1.6 Usage

1. Build and deploy the application to your NetBurner device
2. Connect the device to your network
3. Access the device's IP address in a web browser
4. Navigate to the `Appppost.html` page
5. Upload your application binary file
6. Wait for the update confirmation and automatic reboot

12.55.7.1.7 Development Notes

- The application includes system diagnostics that should be removed for production builds
- Authentication credentials are hardcoded and should be changed for production use
- The 5-second delay before reboot allows time for the HTTP response to complete
- All POST data is consumed during authentication to prevent buffer issues

12.55.7.1.8 Error Handling

The application provides basic error handling:

- Failed updates return an error message via the web interface
- Network timeouts are handled during authentication
- Invalid credentials prompt for re-authentication

12.55.8 HTML Cookie

12.55.8.1 NetBurner HTML Cookie Example

12.55.8.1.1 Overview

The NetBurner HTML Cookie Example is an embedded web application that demonstrates HTTP cookie handling capabilities on NetBurner devices. This application showcases how to set, retrieve, and display cookies in a web browser through an embedded web server.

Three web pages are created: an index page, a page to set the cookie value to "MyCookie", and a page to view the cookie value.

- Select the "set cookie" link to set the cookie. Note that if you select "show cookie" before setting the value, no value will be displayed.
- Select the "show cookie" link to view the cookie

A cookie will apply to a "site". In this example a "site" will be the IP address of your NetBurner device. The cookie will be sent as part of the HTML header for any page requested from the "site".

12.55.8.1.2 Application Structure

12.55.8.1.2.1 Core Files

- `main.cpp` - Main application entry point and network initialization
- `webfuncs.cpp` - Web server functionality and cookie handling logic
- `index.html` - Web interface with dynamic content display

12.55.8.1.2.2 Main Application (`main.cpp`)

The main application initializes the NetBurner network stack and starts the web server:

- Initializes the network stack using `init()`
- Enables system diagnostics for debugging
- Starts HTTP server on default port 80
- Waits for **DHCP** network configuration (5 second timeout)
- Displays application information and NNDK revision
- Enters main loop with 1-second delays

```
void UserMain(void *pd)
{
    init(); // Initialize network stack
    EnableSystemDiagnostics();
    StartHttp(); // Start web server, default port 80
    WaitForActiveNetwork(TICKS_PER_SECOND * 5); // Wait for DHCP address

    iprintf("Application: %s\r\nNNDK Revision: %s\r\n", AppName, GetReleaseTag());

    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND * 1);
    }
}
```

12.55.8.1.3 Cookie Handling Features

12.55.8.1.3.1 Cookie Display (`ShowCookie`)

The application includes a function that demonstrates how to parse and display cookies from HTTP requests:

- Parses HTTP headers to locate "Cookie:" keyword
- Extracts cookie data from the request
- Displays cookie content to the requesting socket
- Handles cases where no cookie is present

12.55.8.1.3.2 Cookie Setting (setCookieGetReqCallback)

The application includes a callback function that sets cookies:

- Intercepts GET requests to `setcookie.html`
- Sends HTTP headers with cookie data ("MyCookie ")
- Serves the appropriate HTML page
- Takes complete responsibility for the HTTP response

12.55.8.1.4 Technical Implementation

12.55.8.1.4.1 Cookie Parsing Logic

The `ShowCookie` function implements a simple cookie parser:

1. Receives the URL and HTTP header data
2. Searches for the "Cookie:" string in the HTTP headers
3. Extracts the cookie value until a space character is found
4. Writes the cookie data back to the client socket

12.55.8.1.4.2 HTTP GET Handler

The application uses NetBurner's callback system to handle specific web requests:

```
CallBackFunctionPageHandler hIndex("setcookie.html",    // Web page to intercept
                                   setCookieGetReqCallback, // Pointer to callback function
                                   tGet,                // Type of request, GET
                                   0,                    // Password level, none
                                   true);               // Take responsibility for entire response
```

12.55.8.1.5 Network Configuration

12.55.8.1.5.1 DHCP Setup

- Automatically obtains IP address via [DHCP](#)
- Waits up to 5 seconds for network activation
- Reports network status to debug console

12.55.8.1.5.2 Web Server

- HTTP server runs on standard port 80
- Serves static HTML files and dynamic content
- Handles both GET requests and cookie operations

12.55.8.1.6 Usage Instructions

1. **Setup:** Deploy the application to a NetBurner device
2. **Network:** Ensure the device is connected to a network with [DHCP](#)
3. **Access:** Open a web browser and navigate to the device's IP address
4. **Set Cookie:** Navigate to `setcookie.html` to set a cookie
5. **View Cookie:** Navigate to `showcookie.html` to display the cookie value

12.55.8.1.7 Development Notes

12.55.8.1.7.1 Function Declarations

- Cookie handling functions must be declared as `extern "C"` to prevent C++ name mangling
- This ensures proper linkage with the NetBurner web server system

12.55.8.1.7.2 Debug Output

- The application outputs debug information to the console
- URL requests and cookie operations are logged for development purposes

12.55.8.1.7.3 Memory Management

- Uses stack-based string parsing for cookie extraction
- No dynamic memory allocation for cookie handling
- Relies on NetBurner's built-in HTTP handling infrastructure

12.55.8.1.8 Dependencies

- NetBurner NNDK (NetBurner Network Development Kit)
- Standard NetBurner libraries:
 - `init.h` - Network initialization
 - `system.h` - System functions
 - `nbrtos.h` - Real-time operating system
 - `http.h` - HTTP server functionality
 - `iosys.h` - Input/output system
 - `htmlfiles.h` - HTML file handling

12.55.8.1.9 Security Considerations

- Cookie data is transmitted in plain text
- No authentication or authorization mechanisms
- Suitable for development and demonstration purposes
- Production use should implement additional security measures

12.55.8.1.10 Troubleshooting

12.55.8.1.10.1 Common Issues

1. **Network Connection:** Ensure [DHCP](#) is available and device is connected
2. **Browser Cache:** Clear browser cache if cookie behavior seems inconsistent
3. **Debug Output:** Monitor console output for network and cookie operation status

12.55.8.1.10.2 Debug Information

The application provides debug output including:

- Application name and NNDK revision
- Network status and IP address assignment
- HTTP request URLs and cookie operations

12.55.9 HTML Form Post

12.55.9.1 HtmlFormPost Example

12.55.9.1.1 Overview

This NetBurner example demonstrates how to handle HTML form submissions using HTTP POST requests on embedded devices. The application showcases complete form processing workflows including data validation, error handling, and user feedback through a multi-page web interface.

12.55.9.1.2 Features

- Interactive HTML form processing with POST request handling
- Multi-page form workflow with user navigation
- Real-time data validation and error reporting
- Persistent data storage during form processing sessions
- Console logging for debugging and monitoring
- Safe string handling with buffer overflow protection

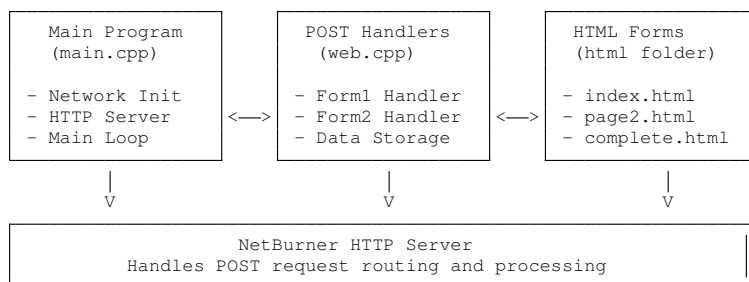
12.55.9.1.3 Project Structure

```

HtmlFormPost/
├── src/
│   ├── main.cpp          # Main application and HTTP server startup
│   └── web.cpp            # POST request handlers and form processing
├── html/
│   ├── index.html        # Form 1 - Single field input
│   ├── page2.html        # Form 2 - Multi-field input
│   ├── complete.html     # Results page with processed data
│   └── images/
│       └── netburner-logo.gif # NetBurner logo image
└── makefile              # Build configuration
  
```

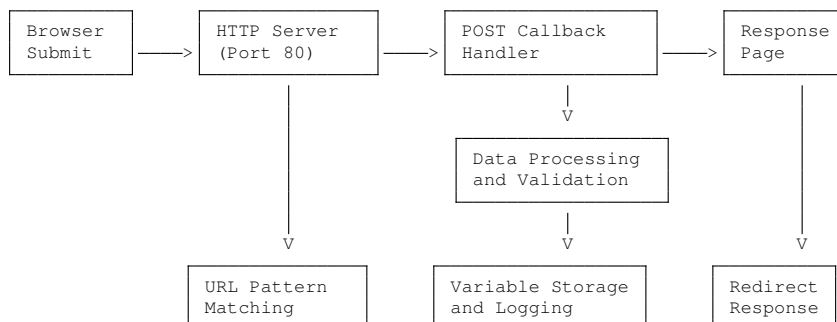
12.55.9.1.4 Architecture

12.55.9.1.4.1 System Components



12.55.9.1.4.2 POST Request Processing Flow

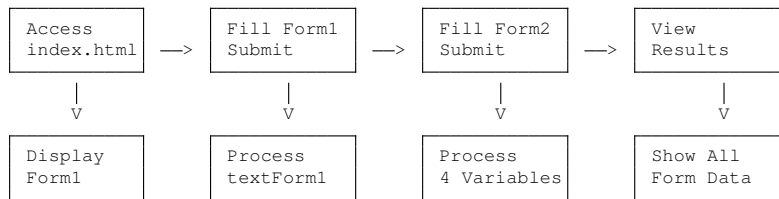
Form Submission Flow:



12.55.9.1.5 Application Flow

12.55.9.1.5.1 User Interaction Sequence

User Workflow:



12.55.9.1.6 Key Components

12.55.9.1.6.1 Main Application (main.cpp)

The main application handles system initialization and HTTP server setup:

```

void UserMain(void *pd) {
    init(); // Initialize network stack
    EnableSystemDiagnostics(); // Enable debug output
    StartHttp(); // Start HTTP server on port 80
    WaitForActiveNetwork(TICKS_PER_SECOND * 5); // Wait for DHCP connectivity

    iprintf("Application: %s\r\nNNDK Revision: %s\r\n", AppName, GetReleaseTag());

    // Main application loop - all form processing handled by callbacks
    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Key Functions:**

- System and network initialization
- HTTP server startup on port 80
- **DHCP** network configuration with 5-second timeout
- Continuous operation loop

12.55.9.1.6.2 Form Processing System (web.cpp)

Form 1: Single Field Processing

Data Storage:**

```

const int varLenForm1 = 80;
char textForm1[varLenForm1];

```

POST Callback Handler:**

```

int form1PostCallBack(int sock, PostEvents event, const char *pName, const char *pValue) {
    switch (event) {
        case eStartingPost:
            // Initialize processing
            break;

        case eVariable:
            // Process form variable
            if (strcmp("textForm1", pName) == 0) {
                strncpy(textForm1, pValue, varLenForm1 - 1);
                textForm1[varLenForm1 - 1] = '\0';
                iprintf("textForm1 set to: \"%s\"\r\n", pValue);
            }
            break;

        case eFile:
            // Handle file uploads (not used)
            break;

        case eEndOfPost:
            // Send response after processing
            RedirectResponse(sock, "page2.html");
            break;
    }
    return 0;
}

```

Registration:**

```

HtmlPostVariableListCallback postForm1("form1*", form1PostCallBack);

```

Form 2: Multi-Field Processing

Data Storage:**

```
const int numVars = 4;
const int varLen = 80;
char strVar[numVars][varLen];
bool form2ParseError = false;
```

POST Callback Handler:**

```
int form2PostCallBack(int sock, PostEvents event, const char *pName, const char *pValue) {
    switch (event) {
        case eStartingPost:
            // Initialize all variables
            for (int i = 0; i < numVars; i++) {
                strVar[i][0] = '\0';
            }
            form2ParseError = false;
            break;

        case eVariable:
            // Process each variable with validation
            if (strcmp("Var0", pName) == 0) {
                strncpy(strVar[0], pValue, varLen - 1);
                strVar[0][varLen - 1] = '\0';
                iprintf("strVar0 set to: \"%s\"\\r\\n", strVar[0]);
            } else if (strcmp("Var1", pName) == 0) {
                // Similar processing for Var1, Var2, Var3
            } else {
                // Unknown variable detected
                iprintf("*** Error: Form variable name not found\\r\\n");
                form2ParseError = true;
            }
            break;

        case eEndOfPost:
            // Display summary and redirect
            iprintf("Variable Summary:\\r\\n");
            for (int i = 0; i < numVars; i++) {
                iprintf("  Var%d = \"%s\"\\r\\n", i, strVar[i]);
            }
            RedirectResponse(sock, "complete.html");
            break;
    }
    return 0;
}
```

12.55.9.1.6.3 Dynamic Content Functions

Form 1 Data Display

```
void WebTextForm1(int sock, PCSTR url) {
    fdprintf(sock, "\\\"%s\\\"", textForm1);
}
```

Form 2 Data Display with Error Reporting

```
void WebTextForm2(int sock, PCSTR url) {
    fdprintf(sock, "<pre>\\r\\n");
    for (int i = 0; i < numVars; i++)
        fdprintf(sock, "  Var%d = \"%s\"\\r\\n", i, strVar[i]);
    if (form2ParseError)
        fdprintf(sock, "*** Error: Form variable name not found\\r\\n </pre>");
}
```

12.55.9.1.7 HTML Form Structure

12.55.9.1.7.1 Form 1: Single Field Input (index.html)

```
<html>
<title>HTML Post Example</title>
<body>

<h1><font face="arial">HTML Form Post Example, First Form</font></h1>

<form action="form1" method=post>
  <table>
    <tr>
      <td width=40> </td>
      <td>Enter data to post:
        <input type="text" name="textForm1" value="<!--CPPCALL WebTextForm1 -->">
      </td>
      <td><input type="submit" value="Submit"></td>
    </tr>
```

```

</table>
</form>
</body>
</html>

```

Form Elements:**

- **Action:** form1 - Routes to Form 1 POST handler
- **Method:** post - Uses HTTP POST for data transmission
- **Input field:** textForm1 - Single text input with preserved value
- **Dynamic content:** <!--CPPCALL WebTextForm1 --> - Shows current value

12.55.9.1.7.2 Form 2: Multi-Field Input (page2.html)

```

<html>
<title>HTML Post Example</title>
<body>

<h1><font face=arial>HTML Form Post Example, Second Form</font></h1>

<form action="form2" method=post>
  Enter data to post var0: <input type="text" name="Var0" value="Var0"><br><br>
  Enter data to post var1: <input type="text" name="Var1" value="Var1"><br><br>
  Enter data to post var2: <input type="text" name="Var2" value="Var2"><br><br>
  Enter data to post var3: <input type="text" name="Var3" value="Var3"><br><br>

  <input type="submit" value="Submit">
</form>
</body>
</html>

```

Form Elements:**

- **Action:** form2 - Routes to Form 2 POST handler
- **Method:** post - HTTP POST transmission
- **Multiple inputs:** Var0, Var1, Var2, Var3 - Four text fields
- **Default values:** Pre-populated with variable names for testing

12.55.9.1.7.3 Results Page: Data Display (complete.html)

```

<html>
<title>HTML Post Example</title>
<body>

<h1><font face="arial">HTML Form Post Example, Completion Page</font></h1>

<font face="arial" size=5>Congratulations, Form Processing Complete</font>

Form1 Data: <!--CPPCALL WebTextForm1 -->

Form2 Data:
<!--CPPCALL WebTextForm2 -->

<font face=veranda><a href="index.html">Back to Home Page</a></font>
</body>
</html>

```

Dynamic Content:**

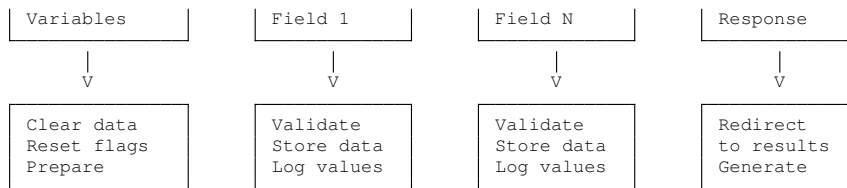
- **Form 1 results:** <!--CPPCALL WebTextForm1 --> - Displays processed Form 1 data
- **Form 2 results:** <!--CPPCALL WebTextForm2 --> - Shows all Form 2 variables
- **Navigation:** Link back to start page for repeated testing

12.55.9.1.8 POST Event Processing

12.55.9.1.8.1 Event Types and Sequence

Event Processing Timeline:





eStartingPost

- **When:** Called at the beginning of form processing
- **Purpose:** Initialize variables and reset state
- **Use cases:** Clear previous data, reset error flags, prepare for new submission

eVariable

- **When:** Called once for each form variable
- **Purpose:** Process individual form fields
- **Parameters:** Variable name and value provided
- **Use cases:** Data validation, storage, format conversion

eFile

- **When:** Called for file upload fields
- **Purpose:** Handle uploaded files
- **Note:** Not used in this example but available for file upload forms

eEndOfPost

- **When:** Called after all variables processed
- **Purpose:** Send response to browser
- **Use cases:** Generate response pages, redirect to results, send confirmations

12.55.9.1.9 Data Validation and Error Handling

12.55.9.1.9.1 Buffer Overflow Protection

All string operations use safe copying with size limits:

```
// Safe string copying with explicit size limit
strncpy(textForm1, pValue, varLenForm1 - 1);
textForm1[varLenForm1 - 1] = '\0'; // Ensure null termination
```

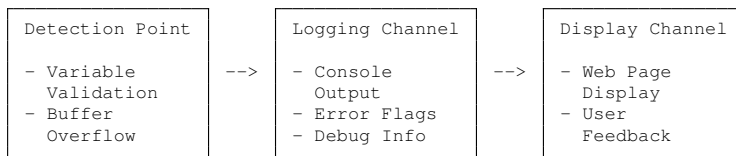
12.55.9.1.9.2 Variable Name Validation

Form 2 includes comprehensive variable validation:

```
if (strcmp("Var0", pName) == 0) {
    // Valid variable - process normally
    strncpy(strVar[0], pValue, varLen - 1);
    fprintf("strVar0 set to: \"%s\"\r\n", strVar[0]);
} else {
    // Invalid variable - set error flag
    fprintf("*** Error: Form variable name not found\r\n");
    form2ParseError = true;
}
```

12.55.9.1.9.3 Error Reporting Channels

Error Reporting Flow:



Errors are reported through multiple channels:

- **Console output:** Real-time debugging information
- **Error flags:** Boolean variables track processing status
- **Web display:** Error messages shown on result pages

12.55.9.1.10 Usage Instructions

12.55.9.1.10.1 Build and Deployment

1. **Build the application:**

```
make clean
make
```

2. **Deploy to NetBurner device:**

```
make load DEVIP=<device_ip_address>
```

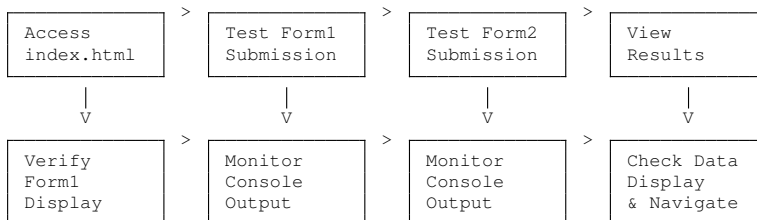
3. **Monitor console output:**

- Connect serial terminal (115200 baud, 8-N-1)
- Observe startup messages and form processing logs

12.55.9.1.10.2 Web Interface Testing

Complete Form Processing Workflow

Testing Flow:



1. **Access main page:**

- Open browser to http://<device_ip>/
- Verify Form 1 display with NetBurner logo

2. **Test Form 1 submission:**

- Enter text in "Enter data to post" field
- Click "Submit" button
- Observe console output: `textForm1 set to: "<your_text>"`
- Verify redirect to `page2.html`

3. **Test Form 2 submission:**

- Modify default values in Var0, Var1, Var2, Var3 fields
- Click "Submit" button
- Observe console output for each variable
- Verify redirect to `complete.html`

4. View results page:

- Check Form1 Data display
- Verify Form2 Data shows all four variables
- Test "Back to Home Page" navigation

12.55.9.1.10.3 Expected Console Output

Application Startup

Application: HTML Form Post Example
NNDK Revision: 3.x.x Build xxxx

Form 1 Processing

textForm1 set to: "User entered text"

Form 2 Processing

```
strVar0 set to: "Modified Var0"
strVar1 set to: "Modified Var1"
strVar2 set to: "Modified Var2"
strVar3 set to: "Modified Var3"
```

```
Variable Summary:
  Var0 = "Modified Var0"
  Var1 = "Modified Var1"
  Var2 = "Modified Var2"
  Var3 = "Modified Var3"
```

Error Conditions

Error: Form variable name not found

12.55.9.1.11 URL Pattern Matching

The POST handler registration uses wildcard patterns:

```
// Matches any URL starting with "form1"
HttpPostVariableListCallback postForm1("form1*", form1PostCallback);
```

```
// Matches any URL starting with "form2"
HttpPostVariableListCallback postForm2("form2*", form2PostCallback);
```

Pattern Examples:**

- form1 - Exact match
- form1.html - With extension
- form1/submit - With path
- form1?param=value - With query parameters

12.55.9.1.12 Technical Implementation

12.55.9.1.12.1 Memory Management

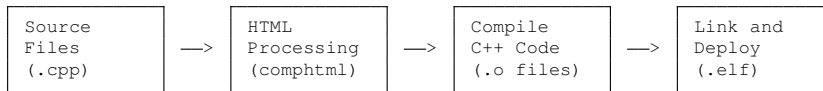
- **Static allocation:** Fixed-size buffers for predictable memory usage
- **Buffer sizing:** Appropriate sizes based on expected input lengths
- **String safety:** Always ensure null termination and bounds checking

12.55.9.1.12.2 Processing Efficiency

- **Minimal processing:** Fast and efficient callback functions
- **Non-blocking:** Avoid long operations in POST callbacks
- **Resource cleanup:** Release resources promptly after processing

12.55.9.1.12.3 Build Configuration

Build Process:



The makefile handles HTML compilation automatically:

NAME = HtmlFormPost

```
# Source files
CPP_SRC += src/main.cpp src/web.cpp
```

```
# HTML compilation
CPP_SRC += src/htmldata.cpp
CREATEDTARGS += src/htmldata.cpp
src/htmldata.cpp : $(wildcard html/*.*)
    comphtml html -osrc/htmldata.cpp
```

```
include $(NNDK_ROOT)/make/boilerplate.mk
```

12.55.9.1.13 Troubleshooting

12.55.9.1.13.1 Common Issues

Troubleshooting Matrix:

Symptom	Possible Cause	Solution
Form not submitting	Action URL mismatch	Check pattern match
Data not appearing	Variable name error	Verify exact names
Redirect not working	Missing eEndOfPost	Check callback
Console output empty	Diagnostics disabled	Enable diagnostics

Form Not Submitting

Symptoms:** Form submission doesn't trigger callback Solutions:**

- Verify form action URL matches registered pattern
- Check HTTP method is set to "post"
- Confirm POST handler is registered properly
- Monitor console for error messages

Data Not Appearing

Symptoms:** Form data not stored or displayed Solutions:**

- Verify variable names match exactly between HTML and callback
- Check case sensitivity of variable names
- Ensure proper string copying and null termination
- Monitor console output for processing messages

Redirect Not Working

Symptoms:** Browser doesn't redirect after form submission Solutions:**

- Verify target page exists and is accessible
- Check [RedirectResponse\(\)](#) is called in eEndOfPost event
- Ensure callback returns 0 for successful processing
- Test with different browsers

12.55.9.1.14 Dependencies

Required NetBurner system libraries:

- `init.h` - System initialization
- `nbrtos.h` - NetBurner RTOS functions
- `http.h` - HTTP server support
- `httppost.h` - POST request handling
- `fdprintf.h` - File descriptor printf functions
- `iosys.h` - I/O system functions
- `string.h` - String manipulation functions

12.55.9.1.15 Learning Objectives

This example teaches:

- **HTTP POST processing:** Complete form submission handling workflows
- **Event-driven programming:** Using callbacks for asynchronous form processing
- **Data validation patterns:** Input validation and error handling techniques
- **Web application workflows:** Multi-page form processing sequences
- **String handling safety:** Buffer overflow protection and safe string operations
- **User interface design:** Creating intuitive web forms and feedback systems

12.55.9.1.16 Advanced Usage

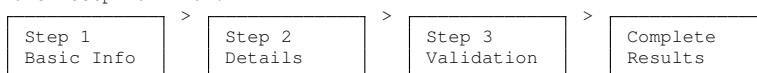
12.55.9.1.16.1 Custom Validation Logic

Extend form processing with custom validation:

```
case eVariable:
    if (strcmp("email", pName) == 0) {
        if (validateEmail(pValue)) {
            strncpy(emailField, pValue, EMAIL_LEN - 1);
            emailField[EMAIL_LEN - 1] = '\0';
        } else {
            validationError = true;
        }
    }
    break;
```

12.55.9.1.16.2 Multi-Step Form Processing

Multi-Step Workflow:



Implement complex workflows:

```
case eEndOfPost:
    if (currentStep == 1) {
        RedirectResponse(sock, "form_step2.html");
    } else if (currentStep == 2) {
        RedirectResponse(sock, "complete.html");
    }
    break;
```

12.55.10 HTML Form Post 2.X Compatible

12.55.10.1 NetBurner HTML Form POST Example

12.55.10.1.1 Overview

This application demonstrates HTML Form POST handling on NetBurner embedded devices using the NetBurner Development Kit (NNDK). It provides backward compatibility with NNDK 2.x POST handling methodology while running on NNDK 3.x platforms.

The standard HTTP POST methodology in 3.x should be used whenever creating a new project. The 3.X methods enable cleaner code, less URL parsing, and better performance.

For customers migrating large and complex applications from legacy NNDK 2.X tools and platforms to NNDK 3.X, it is possible to create a HTTP POST callback function that in turn calls a `MyDoPost()` style 2.X function. This example assumes familiarity with the NNDK 3.X HTTP POST methods.

- A NNDK 3.X HTTP POST callback function is declared with a wildcard mask ("*") so that it is called for all HTTP POST operations.
- Strings are created for the URL containing the form action name and the posted form data
- The 2.X style `MyDoPost()` function is called with the URL and data strings
- The `ExtractPostData()` function is copied from the 2.X tools and included as a function in the application

Once the example has been loaded, interaction is through the device's web page. There are two web pages with forms for data entry, and a summary page at the end to display the assigned values from the post. Status messages are displayed on the serial port that can be viewed with a serial terminal, such as the MTTY utility.

While every attempt has been made to ensure compatibility, every application must be tested to verify proper operation. Also, this example will not work with multi-part forms.

12.55.10.1.2 Purpose

The application illustrates how to:

- Process HTML POST requests from web browsers
- Handle multiple web forms with different data fields
- Extract and parse form data from HTTP POST requests
- Provide web page responses and redirects
- Maintain compatibility between NNDK 2.x and 3.x POST handling methods

12.55.10.1.3 Key Features

- **Dual POST Handling:** Implements both legacy 2.x style `MyDoPost()` function and modern 3.x callback mechanisms
- **Multiple Form Support:** Handles two separate HTML forms with different data fields
- **Data Extraction:** Includes URL decoding for form data (handles spaces, special characters)
- **Web Page Flow:** Demonstrates proper HTTP response handling with redirects
- **Debug Output:** Provides serial console output for monitoring POST operations

12.55.10.1.4 Application Structure

12.55.10.1.4.1 Main Components

1. **main.cpp:** Application entry point and initialization
2. **web.cpp:** Web server POST handling and form processing logic

12.55.10.1.4.2 Form Processing

The application processes two types of forms:

Form 1 (`/form1`)*:

- `formValue1`: First text input
- `formValue2`: Second text input
- `formValue3`: Third text input
- Redirects to `page2.htm` after processing

Form 2 (`/form2`)*:

- `textForm2`: Single text input
- Redirects to `complete.htm` after processing

12.55.10.1.5 Technical Details

12.55.10.1.5.1 POST Data Processing

The `ExtractPostData()` function handles:

- URL-encoded data parsing
- Special character decoding (XX hex codes)
- Space character handling (+ to space conversion)
- Parameter separation (& delimited)

12.55.10.1.5.2 Compatibility Layer

The application provides a bridge between NNDK versions:

- **3.x Callback:** `form1PostCallBack()` handles modern POST events
- **2.x Function:** `MyDoPost()` maintains legacy function signature
- **Event Processing:** Converts 3.x events to 2.x compatible data structures

12.55.10.1.6 Setup and Configuration

12.55.10.1.6.1 Network Configuration

- Uses [DHCP](#) for automatic IP address assignment
- Waits up to 5 seconds for network connectivity
- Default web page: `index.htm`

12.55.10.1.6.2 Build Requirements

- NetBurner Development Kit (NNDK) 3.x
- Compatible NetBurner hardware platform
- Web server files (HTML forms)

12.55.10.1.7 Usage

1. **Compile and Load:** Build the application and load it onto your NetBurner device
2. **Network Connection:** Ensure the device obtains a network connection via [DHCP](#)
3. **Web Interface:** Access the device's IP address through a web browser
4. **Form Interaction:**
 - Fill out Form 1 on the main page
 - Submit to process and redirect to page 2
 - Fill out Form 2 and submit
 - View results on the completion page

12.55.10.1.8 Serial Console Output

The application provides detailed debugging information via serial console:

```
Application: HTML Form Post Example 2.x Port
NNDK Revision: [version]
----- Processing Post -----
Post URL: /form1
Post Data: formValue1=test&formValue2=data&formValue3=example
```

12.55.10.1.9 Migration Notes

12.55.10.1.9.1 For 2.x to 3.x Migration

This example is specifically designed for customers migrating complex applications from NNDK 2.x to 3.x:

- **Minimal Code Changes:** Existing `MyDoPost()` functions can be reused
- **Gradual Migration:** Allows incremental updates to newer POST handling methods
- **Testing Framework:** Provides compatibility verification for existing applications

12.55.10.1.9.2 Limitations

- **Multi-part Forms:** This compatibility layer does not support multi-part form uploads
- **Performance:** 3.x native methods provide better performance for new applications
- **Maintenance:** New projects should use standard 3.x POST methodology

12.55.10.1.10 Web Page Requirements

The application expects the following web pages:

- `index.htm`: Main page with Form 1
- `page2.htm`: Second page with Form 2
- `complete.htm`: Results/completion page

Each form should use appropriate `action` attributes (`form1`, `form2`) and matching input field names.

12.55.10.1.11 Development Recommendations

12.55.10.1.11.1 For New Projects

Use the standard NNDK 3.x HTTP POST methodology for:

- Cleaner code structure
- Better performance
- Reduced URL parsing overhead
- Enhanced functionality

12.55.10.1.11.2 For Legacy Migration

This example provides a proven path for:

- Large application migrations
- Complex form processing systems
- Gradual modernization of existing codebases

12.55.11 HTML Password

Warning

Web browsers will cache the username and password once entered. To force a password request again you will need to clear the browser cache, restart the browser, or open a new private browsing session.

12.55.11.1 Methods to Set an Access Group

Web page user names and passwords are controlled by "Access Group" levels that can range from 0 to 255, with 0 being no password protection. There are 3 methods to set the Access Group levels as described below. An time Access Groups are used in an application, the `CheckHttpAccess()` function must be created in the application to process the access groups and determine if access should be granted. The function is defined as:

```
HTTP_ACCESS CheckHttpAccess(int sock, int access_level, HTTP_Request &Req)
```

1. In a HTML page using `<!--ACCESSGROUP n -->` HTML tag, where n is a number from 0 to 255, typically 1 for admin and 2 for user, etc.. This provides the finest level of control on a page by page basis by adding the ACCESSGROUP tag as the first line in any web page that needs password protection. The application must create a `CheckHttpAccess()` function, that is automatically be called by the system, to process each request, such as checking a user name and password against stored values.

1. For dynamic web content, create a HTML GET handler callback function. If you do not have a web page compiled into an application in which to put an ACCESSGROUP tag, the access group can be specified in the handler callback function declaration:

```
CallbackFunctionPageHandler(const char *pUrl, http_gethandlerfunc *pFunction,
    HTTP_RequestTypes reqType = tGet, int accessGroup = 0,
    bool beforeFiles = false)
```

For example, if you have a dynamically created page for administration of users and passwords that generates a list of such information the access group could be set to 1 in the callback declaration:

```
CallbackFunctionPageHandler getHandlerAdmin("admin.html*", getAdmin, tGet, 1);
```

For reference, a GET callback declaration with no access group specified defaults to 0 and only needs the first two parameters:

```
CallbackFunctionPageHandler getHandlerDynamicPage("dynamicPage.html", getDynamicPage);
```

1. Create a dynamic HTML POST handler callback function:

```
HtmlPostVariableListCallback(const char *pUrl, postvarhandler *pCallback, int accessGroup = 0)
```

Similar to the GET handler above, creating a callback with no access group relies on ACCESSGROUP tags:

```
HtmlPostVariableListCallback postForm1("form1*", form1PostCallBack);
```

And specifying the access group for a post can also be used:

```
HtmlPostVariableListCallback postForm2("form2*", form2PostCallBack, 2);
```

12.55.11.2 ACCESSGROUP HTML Tags

The application consists of an index page with links to three additional pages with access group numbers: 0, 1 and 2.

- Group 0 = Open access
- Group 1 = Admin access
- Group 2 = User access

The function

```
HTTP_ACCESS CheckHttpAccess(int sock, int access_level, HTTP_Request & Req)
```

is used to authenticate the user names and passwords. If you create a function with this signature, it will be called any time one of the following access group tags are encountered in a web page:

Any web pages without an ACCESSGROUP tag will default to open access.

12.55.12 HtmlPostDateTime

12.55.12.1 HTML Date Time Post Application

12.55.12.1.1 Overview

The HTML Date Time Post Application is a web-based embedded system application that demonstrates how to handle date and time operations through a web interface. Built using the NetBurner NNDK (NetBurner Network Development Kit), this application provides a complete example of:

- Web server functionality with HTTP POST handling
- Date/time configuration management through web forms
- Time synchronization via NTP (Network Time Protocol)
- Time zone handling and display formatting
- Configuration persistence across system restarts

Config System Date and Time variables are used to post and display date and time from the config system to/from an HTML form. This uses the config form post helper functions.

12.55.12.1.2 Features

12.55.12.1.2.1 Core Functionality

- **Web Server:** Built-in HTTP server running on port 80
- **Date/Time Forms:** HTML forms for setting and displaying date/time values
- **NTP Synchronization:** Automatic time synchronization from NTP pool servers
- **Time Zone Support:** Configurable for PST/PDT with daylight saving transitions
- **Configuration Persistence:** Saves time settings to non-volatile storage
- **Time Calculations:** Shows time intervals and differences

12.55.12.1.2.2 Web Interface Components

- **Date Input Form** (`dateform`): Manual date/time entry
- **Auto Date Form** (`autodateform`): Automated configuration handling
- **Time Display**: Shows stored time in both local and UTC formats
- **Time Delta**: Displays time difference from current time

12.55.12.1.3 Technical Architecture

12.55.12.1.3.1 Key Components

1. Configuration Management

- `config_time_t` `StoredTimeObj`: Manages persistent time storage
- Default time: "2022-05-14T17:05:00"
- Automatic saving to configuration storage

2. HTTP POST Handling

- `DateformPostCallBack()`: Processes manual date form submissions
- `HtmlPostVariableListCallback`: Handles form variables
- `HtmlPostConfigVariableHandler`: Manages configuration forms

3. Display Functions

- `FillInDateValue()`: Outputs stored time in local format
- `FillInDateValueUtc()`: Outputs stored time in UTC format
- `ShowDeltaTime()`: Calculates and displays time differences
- `ShowTimeInterval()`: Formats time intervals (days, hours, minutes, seconds)

4. Network & Time Services

- **DHCP** network configuration
- NTP time synchronization with retry logic
- Time zone configuration for PST/PDT

12.55.12.1.3.2 System Requirements

- NetBurner NNDK development environment
- Network connectivity for **DHCP** and NTP
- Web browser for interface access

12.55.12.1.4 Configuration

12.55.12.1.4.1 Time Zone Setting

The application is configured for Pacific Time Zone:

```
tzsetchar("PST8PDT7,M3.2.0/02:00:00,M11.1.0/02:00:00");
```

- PST: Pacific Standard Time (UTC-8)
- PDT: Pacific Daylight Time (UTC-7)
- Automatic DST transitions in March and November

12.55.12.1.4.2 Default Values

- **Application Name:** "HTML Date Time Post Example"
- **Default Time:** 2022-05-14T17:05:00
- **Network Timeout:** 5 seconds for [DHCP](#)
- **NTP Retry Interval:** 5 seconds

12.55.12.1.5 Usage

1. System Startup

- Device initializes network stack
- Attempts [DHCP](#) network configuration
- Starts HTTP server on port 80
- Synchronizes time via NTP

2. Web Access

- Navigate to device IP address in web browser
- Use HTML forms to set date/time values
- View current stored time and time differences

3. Form Submission

- Submit new date/time through `dateform`
- Values are parsed and stored persistently
- Browser redirects to `index.html` after submission

12.55.12.1.6 Development Notes

12.55.12.1.6.1 Key Files Expected

- `index.html`: Main web interface page
- Configuration storage for persistent settings
- Network configuration for DHCP/static IP

12.55.12.1.6.2 Debugging Features

- System diagnostics enabled (should be disabled for production)
- Console output for time synchronization status
- Application and NNDK revision information display

12.55.12.1.6.3 Error Handling

- Graceful handling of NTP synchronization failures
- Retry mechanism for time setting
- Validation of submitted date/time values

12.55.12.1.7 Build Requirements

This application requires the NetBurner NNDK with the following modules:

- Network stack initialization
- HTTP POST processing
- Configuration system
- NTP client functionality
- Time zone management

For production deployment, consider disabling system diagnostics and implementing additional security measures for web access.

12.55.12.1.8 API Reference

12.55.12.1.8.1 Main Functions

- `UserMain()`: Application entry point and main loop
- `DateformPostCallBack()`: Handles POST form submissions
- `ShowTimeInterval()`: Formats time duration display
- `FillInDateValue()` / `FillInDateValueUtc()`: Provide time values for web display

12.55.12.1.8.2 Configuration Objects

- `StoredTimeObj`: Persistent time storage with automatic loading/saving
- Time format: ISO 8601 compatible string parsing and generation

12.55.13 HTML Server GET Request

12.55.13.1 HtmlServerGetRequest Example

12.55.13.1.1 Overview

This NetBurner example demonstrates how to implement and use web server GET request callbacks to intercept browser requests. The application showcases custom processing capabilities and dynamic web content generation when specific web pages or URLs are requested, providing developers with powerful tools for creating interactive embedded web applications.

12.55.13.1.2 Features

- HTTP GET request interception and custom processing
- Dynamic content generation based on system state
- File system integration with fallback mechanisms
- Real-time request logging with client information
- Conditional content routing and time-based page selection
- Seamless integration of static and dynamic content

12.55.13.1.3 Project Structure

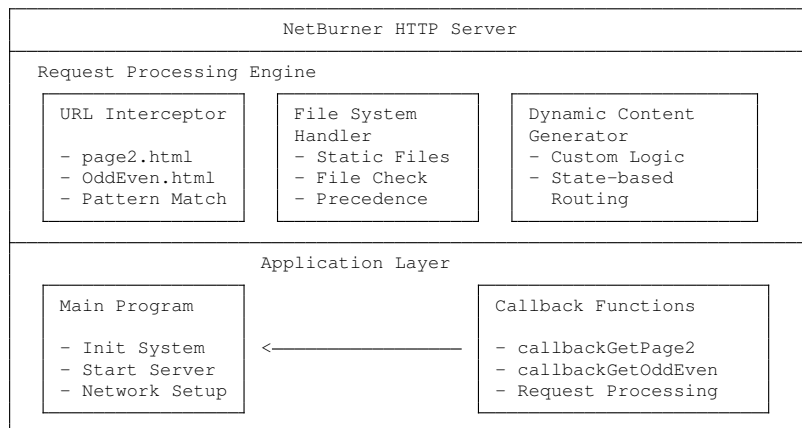
```

HtmlServerGetRequest/
├── src/
│   ├── main.cpp          # Main application and HTTP server startup
│   ├── web.cpp           # GET request callback implementations
│   └── htmlvar.h         # HTML variable declarations
├── html/
│   ├── index.html        # Main navigation page
│   ├── odd.html          # Content for odd seconds
│   ├── even.html         # Content for even seconds
│   ├── page2-rename.html # Demo file for testing precedence
│   └── images/
│       └── netburner-logo.gif # NetBurner logo image
└── makefile              # Build configuration

```

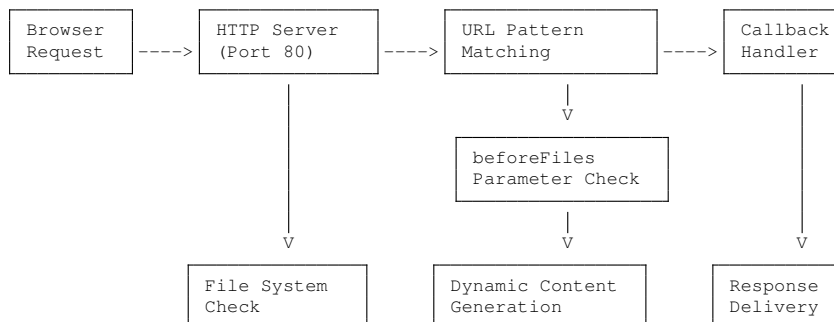
12.55.13.1.4 Architecture

12.55.13.1.4.1 System Components



12.55.13.1.4.2 Request Processing Flow

Browser Request Processing Flow:



12.55.13.1.5 Key Components

12.55.13.1.5.1 Main Application (main.cpp)

The main application handles system initialization and HTTP server setup:

```

void UserMain(void *pd) {
    init(); // Initialize network stack
    EnableSystemDiagnostics(); // Enable debug output
    StartHttp(); // Start HTTP server on port 80
    WaitForActiveNetwork(TICKS_PER_SECOND * 5); // Wait for network connectivity

    iprintf("Application: %s\r\nNNDK Revision: %s\r\n", AppName, GetReleaseTag());

    // Main application loop - all web processing handled by callbacks
    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Key Functions:**

- System and network initialization
- HTTP server startup on port 80
- **DHCP** network configuration with 5-second timeout
- Continuous operation loop

12.55.13.1.5.2 Web Server Callbacks (web.cpp)

Callback 1: page2.html Handler

Purpose:** Demonstrates conditional content generation when files don't exist

```
int callbackGetPage2(int sock, HTTP_Request &pHttpRequest) {
    SendHTMLHeader(sock);

    // Custom processing and logging
    iprintf("Executing GET request callback function: callbackGetPage2\r\n");
    iprintf("Called because page2.html was not found in the file system.\r\n");

    // Generate dynamic HTML content
    fdprintf(sock, "<html><body>\r\n");
    fdprintf(sock, "<h1><font face=\"arial\">");
    fdprintf(sock, "Content when a page2.html file does not exist");
    fdprintf(sock, "</font></h1>\r\n");
    fdprintf(sock, "</body></html>\r\n");

    return 1; // Indicate request was handled
}
```

Registration Parameters:**

```
CallbackFunctionPageHandler getHandlerPage2(
    "page2.html", // URL to intercept
    callbackGetPage2, // Callback function pointer
    tGet, // HTTP GET request type
    0, // No authentication required
    false // beforeFiles: only call if file not found
);
```

Callback 2: OddEven.html Handler

Purpose:** Demonstrates time-based dynamic content selection

```
int callbackGetOddEven(int sock, HTTP_Request &pHttpRequest) {
    SendHTMLHeader(sock);

    // Request logging with client details
    iprintf("Executing GET request callback function: callbackGetOddEven\r\n");
    iprintf("Request URL: %s, from: %I\r\n",
        pHttpRequest.pURL, pHttpRequest.client_IPAddr);

    // Time-based content selection
    if ((Secs % 2) == 0) {
        SendFileFragment("even.html", sock);
    } else {
        SendFileFragment("odd.html", sock);
    }

    return 1; // Indicate request was handled
}
```

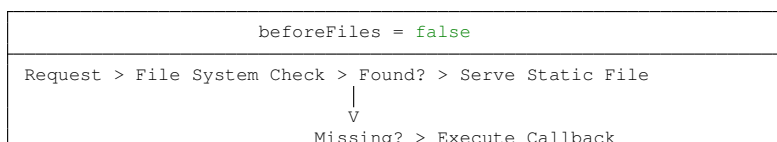
Registration Parameters:**

```
CallbackFunctionPageHandler getHandlerOddEven(
    "OddEven.html", // URL to intercept (virtual URL)
    callbackGetOddEven, // Callback function pointer
    tGet, // HTTP GET request type
    0, // No authentication required
    true // beforeFiles: always call callback
);
```

12.55.13.1.6 Callback Control Mechanisms

12.55.13.1.6.1 beforeFiles Parameter Behavior

beforeFiles Parameter Control Flow:



beforeFiles = true		
Request > Execute Callback > Generate Dynamic Content		

beforeFiles = false (File System Priority)

Behavior:**

- File system checked first for requested URL
 - If file exists, static file served directly
 - If file missing, callback executes to generate content
 - Callback only runs as fallback mechanism
- Use Cases:**
- Fallback content when files are missing
 - Development with uploadable content
 - Conditional dynamic generation

beforeFiles = true (Callback Priority)

Behavior:**

- Callback always executes regardless of file existence
 - No file system check performed
 - Complete control over response generation
 - Always dynamic content
- Use Cases:**
- Always dynamic content requirements
 - Real-time data streams
 - Custom processing for every request

12.55.13.1.6.2 HTTP_Request Structure

The callback functions receive detailed request information:

[HTTP_Request](#) Structure:

Field	Type	Description
pURL	char*	Requested URL string
client_IPaddr	IPADDR	Client IP address
Additional fields...	Various	Headers, parameters, etc.

Available Information:**

- **Request URL:** Complete URL path requested
- **Client IP:** Source IP address of requesting client
- **HTTP headers:** Additional request header information
- **Query parameters:** URL parameters and values

12.55.13.1.7 HTML Content Structure

12.55.13.1.7.1 Static Web Pages

Main Navigation Page (index.html)

```
<html>
<title>HTML GET Request Example</title>
<body>

<font face="arial">

<h1>HTML Server GET Request Response Example</h1>

<a href="page2.html">Run GET Request Example for page2.html</a><br><br>
<a href="OddEven.html">Run GET Request Example for OddEven.html</a><br><br>

</font>
</body>
</html>
```

Navigation Elements:**

- NetBurner logo display
- Links to demonstration pages
- Clear page structure and formatting

Time-based Content Pages

odd.html - Content for Odd Seconds:**

```
<html>
<title>HTML GET Request Example</title>
<body>

<font face="arial">

<h1>System seconds timer has an odd value: <!--VARIABLE Secs --></h1>
<a href="index.html"><< Back</a>

</font>
</body>
</html>
```

even.html - Content for Even Seconds:**

```
<html>
<title>HTML GET Request Example</title>
<body>

<font face="arial">

<h1>System seconds timer has an even value: <!--VARIABLE Secs --></h1>
<a href="index.html"><< Back</a>

</font>
</body>
</html>
```

Demo File for Testing (page2-rename.html)

```
<html>
<title>HTML GET Request Example</title>
<body>

<h1><font face="arial">page2.html from file system</font></h1>
</body>
</html>
```

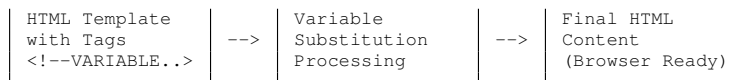
Purpose:**

- Demonstrates file system precedence behavior
- Shows difference between static and dynamic content
- Testing tool for beforeFiles parameter functionality

12.55.13.1.7.2 Dynamic Variable Integration

Variable Substitution Process:





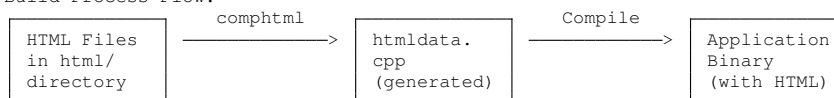
The HTML files use NetBurner's variable substitution system:

- ******** - Displays current system seconds timer
- **Automatic substitution** - Variables replaced with actual values when page served
- **Real-time updates** - Values reflect current system state when page loaded

12.55.13.1.8 File System Integration

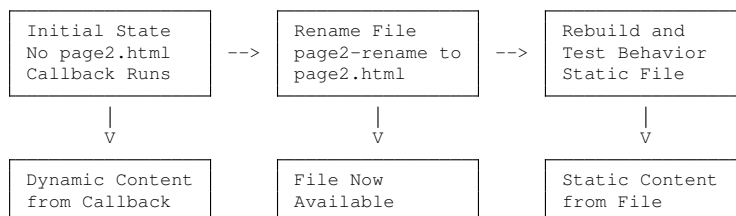
12.55.13.1.8.1 HTML Content Compilation Process

Build Process Flow:



12.55.13.1.8.2 File System Precedence Testing

File Precedence Testing Workflow:



The example includes demonstration of file system precedence:

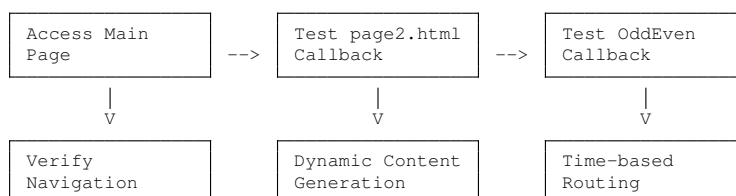
1. **Initially:** No `page2.html` file exists, callback executes
2. **Rename test file:** Change `page2-rename.html` to `page2.html`
3. **Rebuild application:** Static file now embedded in application
4. **Result:** Static file served instead of callback execution

12.55.13.1.9 Usage Instructions

12.55.13.1.9.1 Web Interface Testing

Complete Testing Workflow

Testing Sequence:



Access Main Page

1. **Open browser** to `http://<device_ip>/`
2. **Verify display** of main page with navigation links
3. **Check images** - NetBurner logo should load correctly

Test page2.html Callback

1. **Click link** "Run GET Request Example for page2.html"
2. **Observe result** - Dynamic content from callback function
3. **Check console** - Callback execution messages displayed
4. **Expected content:** "Content when a page2.html file does not exist"

Test OddEven.html Callback

1. **Click link** "Run GET Request Example for OddEven.html"
2. **Observe content** - Either odd.html or even.html content
3. **Note timer value** - System seconds displayed in page
4. **Refresh page** - Content may change based on timer parity
5. **Check console** - Request logging with client IP address

Test File System Precedence

1. **Rename file** - Change page2-rename.html to page2.html
2. **Rebuild application** - Recompile with new file included
3. **Redeploy** to device
4. **Test page2.html** - Now serves static file instead of callback
5. **Expected content:** "page2.html from file system"

12.55.13.1.9.2 Expected Console Output**Application Startup**

Application: HTML Server GET Request
 NNDK Revision: 3.x.x Build xxxx

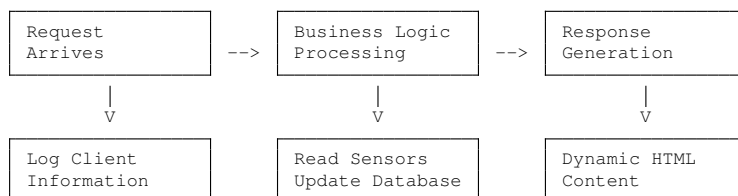
Callback Execution Logging

Executing GET request callback function: callbackGetPage2
 Called because page2.html was not found in the file system.

Executing GET request callback function: callbackGetOddEven
 Request URL: OddEven.html, from: 192.168.1.50

12.55.13.1.10 Advanced Usage Patterns**12.55.13.1.10.1 Custom Processing Integration**

Custom Processing Flow:



Callbacks can include any custom processing before generating responses:

```

int customCallback(int sock, HTTP_Request &pHttpRequest) {
    SendHTMLHeader(sock);

    // Custom business logic
    readSensorData();
    updateDatabase();
    logUserActivity(pHttpRequest.client_IPAddr);
  }
  
```

```
// Generate response based on processing results
generateDynamicContent(sock);

return 1;
}
```

12.55.13.1.10.2 Error Handling Implementation

Implement robust error handling in callbacks:

```
int errorHandlerCallback(int sock, HTTP_Request &pHttpRequest) {
    try {
        SendHTMLHeader(sock);

        // Potentially failing operations
        if (!performOperation()) {
            fdprintf(sock, "<html><body>");
            fdprintf(sock, "<h1>Operation Failed</h1>");
            fdprintf(sock, "</body></html>");
            return 1;
        }

        // Success path
        generateSuccessContent(sock);
        return 1;
    } catch (...) {
        // Error recovery
        fdprintf(sock, "<html><body><h1>Internal Error</h1></body></html>");
        return 1;
    }
}
```

12.55.13.1.10.3 Multiple Callback Registration

Register multiple callbacks for different URL patterns:

```
// Different callbacks for various URL patterns
CallbackFunctionPageHandler handler1("api/data.json", jsonDataCallback, tGet, 0, true);
CallbackFunctionPageHandler handler2("status/*", statusCallback, tGet, 0, true);
CallbackFunctionPageHandler handler3("config.html", configCallback, tGet, 1, false);
```

12.55.13.1.11 Performance Considerations

12.55.13.1.11.1 Callback Efficiency Guidelines

Performance Optimization:

Callback Performance Best Practices
- Minimize processing time to avoid blocking web server - Avoid long-running operations in callback functions - Use efficient string operations and memory management - Clean up resources properly before returning - Consider caching for frequently requested dynamic content

Efficiency Best Practices

- **Minimize processing time** - Keep callback execution fast
- **Avoid blocking operations** - Don't block web server threads
- **Memory management** - Clean up resources properly
- **Connection handling** - Return promptly to free connections

Caching Strategies

- **Static content** - Use compiled files for frequently accessed content
- **Dynamic data** - Cache computed results when appropriate
- **Resource optimization** - Minimize file system access in callbacks

12.55.13.1.12 Troubleshooting**12.55.13.1.12.1 Common Issues and Solutions**

Troubleshooting Matrix:

Symptom	Possible Cause	Solution
Callback not executing	Registration error beforeFiles setting	Check parameters Verify URL match
Content not displaying	Missing HTML header Socket parameter	Call <code>SendHTMLHeader</code> Check <code>fdprintf</code>
File system conflicts	File naming beforeFiles param	Check html/ Rebuild after changes
Network issues	IP configuration Server startup	Verify device IP Check console

Callback Not Executing

Symptoms:** Expected callback doesn't run Solutions:**

- Verify callback registration parameters
- Check `beforeFiles` setting matches intended behavior
- Ensure URL matches exactly (case-sensitive)
- Confirm no conflicting static files exist

Content Not Displaying

Symptoms:** Blank or incomplete web pages Solutions:**

- Verify `SendHTMLHeader()` called before content
- Check `fdprintf()` syntax and socket parameter
- Ensure callback returns 1 to indicate completion
- Monitor console for error messages

Network Connectivity Issues

Symptoms:** Can't access web interface Solutions:**

- Verify device IP address from console output
- Check network connectivity with ping
- Confirm HTTP server started successfully
- Test with different browsers or clients

12.55.13.1.13 Dependencies

Required NetBurner system libraries:

- `init.h` - System initialization
- `nbrtos.h` - NetBurner RTOS functions
- `system.h` - System utilities and version information
- `http.h` - HTTP server support
- `iosys.h` - I/O system functions

- `htmlfiles.h` - HTML file handling
- `httppost.h` - HTTP POST request handling
- `fdprintf.h` - File descriptor printf functions
- `string.h` - String manipulation functions

12.55.14 HTMLVariables

12.55.14.1 HtmlVariables Example

12.55.14.1.1 Overview

This NetBurner example demonstrates how to create dynamic web content using HTML variables and function calls embedded directly in HTML pages. The application shows how to display real-time system information and execute C++ functions from web pages using special HTML tags.

12.55.14.1.2 Features

- Dynamic variable display using VARIABLE tags
- Function execution from HTML using CPPCALL tags
- Real-time system information display
- Parameter passing from HTML to C++ functions
- Uptime counter that updates on page refresh

12.55.14.1.3 Project Structure

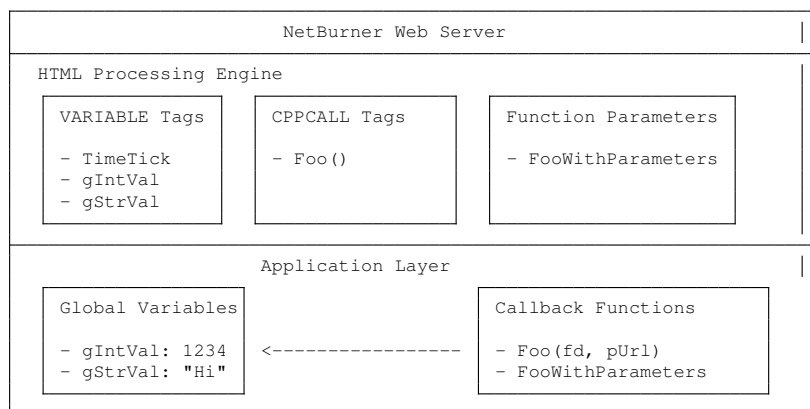
```

HtmlVariables/
├── src/
│   ├── htmlvar.h      # Header file with variable and function declarations
│   ├── main.cpp       # Main application with web server setup
│   └── htmldata.cpp   # Auto-generated file from HTML content
├── html/
│   ├── index.html     # Main web page with dynamic content
│   └── robots.txt     # Web crawler instructions
└── makefile           # Build configuration

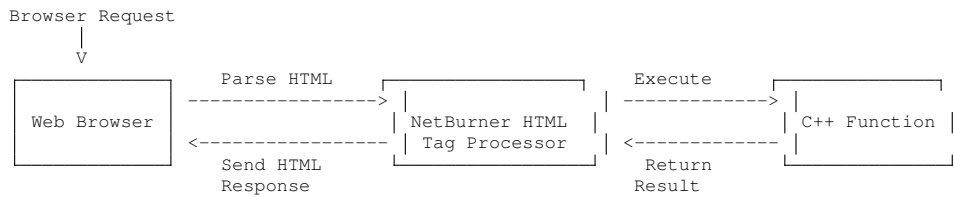
```

12.55.14.1.4 Architecture

12.55.14.1.4.1 System Components



12.55.14.1.4.2 Data Flow



12.55.14.1.5 Key Components

12.55.14.1.5.1 Header File (htmlvar.h)

The header file declares global variables accessible from HTML:

```

#ifndef HTMLVARS_H_
#define HTMLVARS_H_

#include <nbrtos.h> // For access to TimeTick

extern int gIntVal;
extern char gStrVal[];

const char *FooWithParameters(int fd, int v);

#endif /*HTMLVARS_H_*/

```

Exported Elements:**

- extern int gIntVal - Integer value displayed on web page
- extern char gStrVal[] - String value displayed on web page
- const char *FooWithParameters(int fd, int v) - Function with parameters

12.55.14.1.5.2 Main Application (main.cpp)

Variable Initialization

```

const char *AppName = "HTML Variable Example";
int gIntVal = 1234;
char gStrVal[] = "Hello World";

```

System Startup

```

void UserMain(void *pd) {
    init(); // Initialize network stack
    EnableSystemDiagnostics(); // Enable debug output
    StartHttp(); // Start web server, default port 80
    WaitForActiveNetwork(); // Wait for DHCP address

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Key Functions:**

- System and network initialization
- HTTP server startup on port 80
- Network activation wait
- Continuous operation loop

12.55.14.1.5.3 Callback Functions

URL Display Function

```

void Foo(int fd, PCSTR pUrl) {
    fdprintf(fd, "The URL value is: %s\r\n", pUrl);
}

```

Purpose:** Displays URL information on the web page

- **fd:** Socket file descriptor from web server
- **pUrl:** URL string from the HTTP request
- **Output:** Writes URL information directly to the web page

Parameterized Function

```
const char *FooWithParameters(int fd, int v) {
    fprintf(fd, "This function was called with parameter v = %d\r\n", v);
    return "\0"; // Return empty string to avoid additional output
}
```

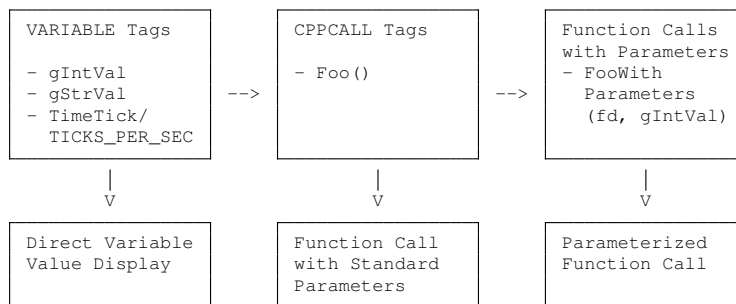
Purpose:** Demonstrates parameter passing from HTML to C++

- **fd:** Socket file descriptor from web server
- **v:** Integer parameter passed from HTML page
- **Return:** Empty string to avoid additional HTML output
- **Output:** Writes parameter information to the web page

12.55.14.1.6 HTML Tag Processing

12.55.14.1.6.1 Tag Types and Usage

HTML Tag Processing Flow:



VARIABLE Tags

Display C++ variables directly in HTML:

```
<!--VARIABLE gIntVal -->
<!--VARIABLE gStrVal -->
<!--VARIABLE TimeTick/TICKS_PER_SECOND -->
```

Features:**

- Direct variable value insertion
- Mathematical expressions (division)
- Real-time value updates

CPPCALL Tags

Execute C++ functions without parameters:

```
<!--CPPCALL Foo -->
```

Features:**

- Function execution with standard parameters (fd, pUrl)
- Direct HTML output from function
- Automatic parameter passing

VARIABLE with Parameters

Call functions with custom parameters:

```
<!--VARIABLE FooWithParameters(fd, gIntVal) -->
```

Features:**

- Custom parameter passing
- Mixed parameter types (file descriptor, variables)
- Function return value handling

12.55.14.1.7 HTML Content Structure

12.55.14.1.7.1 Main Page (index.html)

```
<html>
<body>
<h1>HTML Variable Example</h1>
This example program uses the CPPCALL and VARIABLE tags to create dynamic content<br><br><br>

<h2>VARIABLE Tag</h2>
Number of seconds since system boot: <!--VARIABLE TimeTick/TICKS_PER_SECOND --><br>
The tag is: <pre> &lt;!--VARIABLE TimeTick/TICKS_PER_SECOND --&gt; </pre>
Application integer value: <!--VARIABLE gIntVal --><br>
Application string value: <!--VARIABLE gStrVal --><br>

<h2>Function Call with CPPCALL Tag</h2>
The CPPCALL tag passes the socket file descriptor and URL to a function in your application:<br>
<!--CPPCALL Foo --><br>
The tag is: <pre> &lt;!--CPPCALL Foo --&gt; </pre>

<h2>Function Call with Parameters Using the VARIABLE Tag</h2>
The VARIABLE tag can be used to call a function with a parameter.<br>
Number of time ticks since system boot: <!--VARIABLE FooWithParameters(fd, gIntVal) --><br>
The tag is: <pre> &lt;!--VARIABLE FooWithParameters( fd, gIntVal ) --&gt; </pre>
</body>
</html>
```

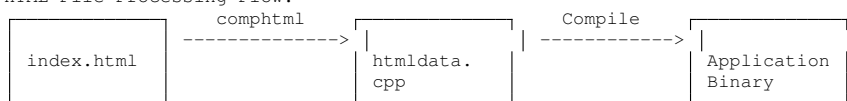
Dynamic Elements:**

- System uptime calculation
- Application variable display
- URL information via function call
- Parameterized function demonstration

12.55.14.1.8 Technical Implementation

12.55.14.1.8.1 HTML Tag Processing

HTML File Processing Flow:

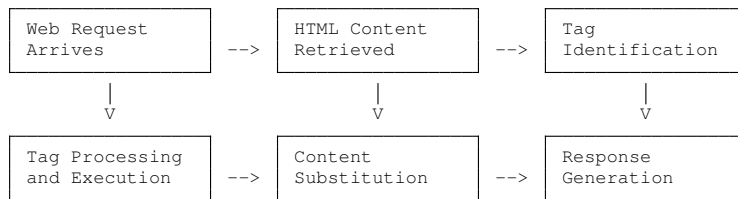


Build Process Steps

1. **HTML Collection:** `comhtml` tool processes all files in `html/` directory
2. **Code Generation:** Creates `htмлdata.cpp` with embedded HTML content
3. **Compilation:** Standard C++ compilation includes generated file
4. **Tag Resolution:** Runtime processing replaces tags with dynamic content

12.55.14.1.8.2 Runtime Tag Resolution

Tag Processing Sequence:

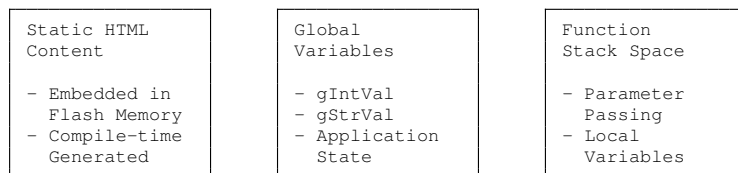


When a web request arrives:

1. Web server identifies HTML tags in content
2. VARIABLE tags resolve to variable values or function returns
3. CPPCALL tags execute functions with standard parameters (fd, pUrl)
4. Results are inserted into HTML before sending response

12.55.14.1.8.3 Memory and Performance

Memory Layout:



Performance Characteristics:**

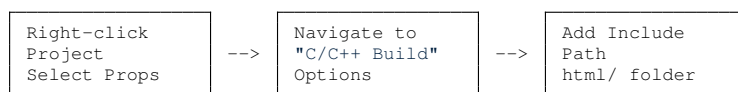
- Static HTML content stored in flash memory
- Dynamic tag processing occurs at request time
- Minimal RAM usage for variable storage
- Fast tag resolution through direct function calls

12.55.14.1.9 Setup Requirements

12.55.14.1.9.1 Include Path Configuration

For NBEclipse users, the auto-generated `htmldata.cpp` file requires access to `htmlvar.h`:

NBEclipse Configuration:

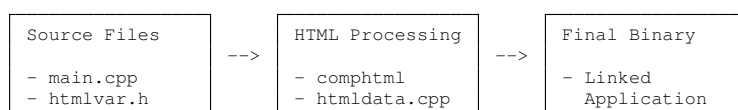


Configuration Steps:**

1. Right-click project and select "Properties"
2. Navigate to "C/C++ Build" options
3. Select "GNU C++ Compiler" > "Directories"
4. Add the project's "HtmlVariables\html" folder to include paths

12.55.14.1.9.2 Build Process

Automated Build Flow:



The build process automatically:

- Processes HTML files with `comphtml` tool
- Generates `htmldata.cpp` from HTML content
- Links HTML content into the application binary

12.55.14.1.10 Web Interface

12.55.14.1.10.1 Dynamic Content Display

Web Page Content Layout:

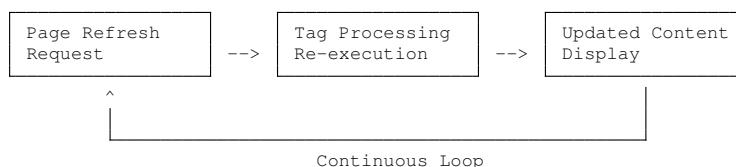
HTML Variable Example
VARIABLE Tag Section: - System Uptime: [TimeTick/TICKS_PER_SECOND] seconds - Integer Value: [gIntVal = 1234] - String Value: [gStrVal = "Hello World"]
CPPCALL Tag Section: - URL Display: [Output from Foo() function]
Function with Parameters Section: - Parameter Demo: [FooWithParameters(fd, gIntVal)]

The web page displays:

- **System Uptime:** Real-time seconds since boot using `TimeTick/TICKS_PER_SECOND`
- **Application Variables:** Current values of `gIntVal` and `gStrVal`
- **URL Information:** Current request URL via `Foo()` function
- **Parameter Demo:** Function call with `gIntVal` parameter

12.55.14.1.10.2 Page Update Behavior

Update Cycle:



Update Characteristics:**

- Content updates on each page refresh
- System uptime increments in real-time
- Function calls execute with each page load
- Variables display current application state

12.55.14.1.11 Build Configuration

12.55.14.1.11.1 Makefile Structure

```

NAME = HtmlVariables

CPP_SRC += src/main.cpp

# Auto-generated targets
CREATEDTARGS += src/htmldata.cpp
CPP_SRC += src/htmldata.cpp
src/htmldata.cpp : $(wildcard html/*.*)

```

```
comphtml html -osrc/htmldata.cpp
include $(NNDK_ROOT)/make/boilerplate.mk
Build Components:**
```

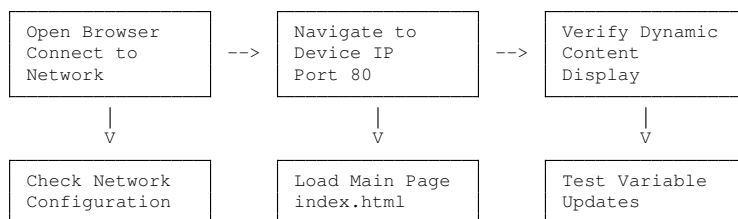
- **NAME:** Project identifier and output binary name
- **CPP_SRC:** C++ source files including generated HTML data
- **CREATEDTARGS:** Files automatically created during build
- **HTML Rule:** Automatic processing of HTML directory contents

12.55.14.1.12 Usage Instructions

12.55.14.1.12.1 Web Interface Testing

Access and Verification

Testing Workflow:



1. Access main page:

- Open browser to `http://<device_ip>/`
- Verify HTML Variable Example page loads

2. Test dynamic content:

- Check system uptime display (should increment)
- Verify integer value shows 1234
- Confirm string value shows "Hello World"
- Observe URL display from Foo() function

3. Verify real-time updates:

- Refresh page multiple times
- Confirm uptime counter increases
- Check that all dynamic content updates properly

12.55.14.1.12.2 Expected Output Examples

Web Page Display

HTML Variable Example

```
VARIABLE Tag
Number of seconds since system boot: 42
Application integer value: 1234
Application string value: Hello World
```

```
Function Call with CPPCALL Tag
The URL value is: /
```

```
Function Call with Parameters Using the VARIABLE Tag
This function was called with parameter v = 1234
```

12.55.14.1.13 Dependencies

Required NetBurner system libraries:

- `init.h` - System initialization
- `nbrtos.h` - NetBurner RTOS functions and timing
- `iosys.h` - I/O system functions
- `netinterface.h` - Network interface management
- `fdprintf.h` - File descriptor printf functions

12.55.14.1.14 Development Notes

12.55.14.1.14.1 Performance Considerations

- Uses NetBurner RTOS timing functions (`TimeTick`, `TICKS_PER_SECOND`)
- Implements standard NetBurner web server initialization
- Application runs in continuous loop with 1-second intervals
- Web server handles automatic HTML tag processing and variable substitution

12.55.14.1.14.2 Production Recommendations

- System diagnostics enabled for development (consider disabling for production)
- Static memory allocation for predictable performance
- Minimal processing overhead for tag resolution
- Flash-based storage for HTML content

12.55.14.1.14.3 Debugging and Monitoring

- Serial console output available at 115200 baud
- System diagnostics provide detailed operational information
- Web server status monitoring through RTOS functions
- Network connectivity verification through standard NetBurner tools

12.55.14.1.15 Learning Objectives

This example teaches:

- **Dynamic HTML content generation:** Creating interactive web pages with embedded C++ integration
- **Tag-based web development:** Using custom HTML tags for dynamic content insertion
- **Real-time data display:** Showing live system information through web interfaces
- **Function integration:** Calling C++ functions directly from HTML pages
- **Parameter passing:** Transferring data between HTML and C++ application layers
- **Web server integration:** Understanding NetBurner HTTP server capabilities

12.55.14.1.16 Advanced Usage Patterns

12.55.14.1.16.1 Custom Variable Types

Extend the system with additional variable types:

```
// In htmlvar.h
extern float gFloatVal;
extern bool gBoolVal;

// In main.cpp
float gFloatVal = 3.14159;
bool gBoolVal = true;
```

12.55.14.1.16.2 Complex Function Parameters

Create functions with multiple parameter types:

```
const char *ComplexFunction(int fd, int intParam, const char *strParam) {
    fdprintf(fd, "Int: %d, String: %s", intParam, strParam);
    return "\0";
}
```

12.55.14.1.16.3 Real-time Data Sources

Integrate with hardware sensors or system monitoring:

```
extern int gTemperature;    // Updated by sensor reading task
extern int gCpuUsage;      // Updated by system monitoring
```

12.55.15 Ping From Web

12.55.15.1 NetBurner Ping From Web Example

12.55.15.1.1 Overview

The Ping From Web Example demonstrates how to create a web-based network diagnostic tool using the NetBurner platform. This example creates a web server that allows users to ping remote hosts (by IP address or DNS name) through a simple HTML form interface, showcasing HTTP POST handling and network diagnostics capabilities.

12.55.15.1.2 Application Description

This embedded web application provides a web-based ping utility that runs on a NetBurner device. Users can enter an IP address or hostname through a web form, and the device will perform a ping operation and display the results. This example demonstrates form handling, DNS resolution, and ICMP ping functionality in an embedded environment.

12.55.15.1.2.1 Key Features

- **Web-Based Interface:** Simple HTML form for entering ping targets
- **DNS Resolution:** Supports both IP addresses and hostnames
- **HTTP POST Handling:** Processes form submissions using NetBurner's HTTP POST framework
- **Network Diagnostics:** Performs ICMP ping operations with response time measurement
- **Real-Time Results:** Displays ping results directly in the web browser
- **Error Handling:** Provides feedback for failed DNS resolution and ping timeouts

12.55.15.1.3 How It Works

12.55.15.1.3.1 Application Flow

1. **Initialization:** The `UserMain()` function initializes the network stack
2. **Diagnostics:** Enables system diagnostics for debugging (should be removed in production)
3. **Web Server:** Starts the HTTP server on port 80

4. **POST Handler:** Registers `PingCallBack` to handle form submissions to "pingpost.html"
5. **Network Wait:** Waits up to 5 seconds for active network connection
6. **Main Loop:** Enters infinite loop with 1-second delays

12.55.15.1.3.2 Ping Process

1. **Form Submission:** User enters IP address or hostname and submits form
2. **Parameter Extraction:** `PingCallBack` extracts the "paddr" parameter from POST data
3. **DNS Resolution:** Uses `GetHostByName()` to resolve hostname to IP address
4. **Ping Execution:** Calls `Ping()` function with 2-second timeout
5. **Result Display:** Shows ping response time or failure message
6. **Return Navigation:** Provides link back to the main ping form

12.55.15.1.3.3 Web Interface

The application serves a simple HTML form (`index.html`) that includes:

- Text input field for IP address or hostname
- Submit button to initiate ping
- Results are displayed on a separate page with navigation back to the form

12.55.15.1.4 Technical Details

12.55.15.1.4.1 Dependencies

- **NBRTOS:** NetBurner Real-Time Operating System
- **Network Stack:** Built-in TCP/IP stack with ICMP support
- **HTTP Server:** Integrated web server with POST handling
- **DNS Client:** Built-in DNS resolution functionality

12.55.15.1.4.2 Key Functions

- `PingCallBack()`: HTTP POST event handler that processes ping requests
- `GetHostByName()`: DNS resolution function
- `Ping()`: ICMP ping function with configurable timeout
- `HtmlPostVariableListCallback`: NetBurner class for handling HTTP POST variables

12.55.15.1.4.3 System Requirements

- NetBurner compatible hardware
- Network connectivity (Ethernet or WiFi depending on platform)
- NNDK (NetBurner Network Development Kit)

12.55.15.1.5 Usage

1. **Compile:** Build the application using the NetBurner development environment
2. **Deploy:** Load the application onto your NetBurner device
3. **Connect:** Ensure the device is connected to your network
4. **Access:** Open a web browser and navigate to the device's IP address
5. **Ping:** Enter an IP address or hostname in the form and click "Ping"
6. **View Results:** The ping results will be displayed with response time or error message

12.55.15.1.6 Example Usage

12.55.15.1.6.1 Successful Ping

```
Enter "google.com" in the form:  
- Address resolved to [142.250.191.46]  
- Ping Responded in 234 Ticks
```

12.55.15.1.6.2 Failed Ping

```
Enter "nonexistent.domain.com" in the form:  
- Failed to resolve address
```

12.55.15.1.7 Development Notes

12.55.15.1.7.1 Production Considerations

- Remove `EnableSystemDiagnostics()` call for production deployments
- Consider adding input validation for the ping target field
- Implement rate limiting to prevent ping flooding
- Add authentication for administrative access
- Consider adding traceroute or other network diagnostic tools

12.55.15.1.7.2 Customization

This example can be extended to include:

- Multiple ping targets in a single request
- Ping history and statistics
- Network topology discovery
- Port scanning capabilities
- Integration with SNMP monitoring
- Automated network health checks

12.55.15.1.7.3 Security Notes

- Ping functionality can be used for network reconnaissance
- Consider restricting ping targets to specific IP ranges in production
- Implement proper input sanitization to prevent injection attacks
- Add logging for audit trails of ping requests

12.55.15.1.8 Console Output

When running, the application displays:

```
Web Application: Ping From Web  
NNDK Revision: [Version Information]
```

12.55.15.1.9 Error Handling

The application handles several error conditions:

- **Empty ping target:** "Ping address empty"
- **DNS resolution failure:** "Failed to resolve address"
- **Ping timeout/failure:** "Ping Failed"
- **Successful ping:** "Ping Responded in X Ticks"

12.55.16 Raw Form Post 2pX

12.55.16.1 Raw Post Callback 2pX

12.55.16.1.1 Overview

This is a NetBurner embedded web server application that demonstrates how to handle **raw** HTTP POST requests. The application creates a web server that can receive and process form data submitted through HTTP POST requests.

12.55.16.1.2 Application Components

12.55.16.1.2.1 Main Application (`main.cpp`)

- **Application Name:** "Raw Post Callback 2pX"
- **Purpose:** Initializes the network stack and starts an HTTP web server
- **Key Features:**
 - Network stack initialization
 - System diagnostics enabled
 - HTTP server startup on default port 80
 - **DHCP** network configuration with 5-second timeout
 - Continuous operation loop

12.55.16.1.2.2 Form Handler (`formcode.cpp`)

- **Purpose:** Handles HTTP POST requests from web forms
- **Key Features:**
 - Raw POST data processing with boundary marker support
 - Real-time data streaming and display
 - HTML response generation
 - Buffer management for large POST data
 - Multipart form data handling capability

12.55.16.1.2.3 Web Interface (`index.html`)

- **Purpose:** Provides a simple web form for user input
- **Features:**
 - NetBurner logo display
 - Single text input field
 - Form submission to `post.html` endpoint
 - Clean HTML table-based layout

12.55.16.1.3 How It Works

1. **Initialization:** The application starts by initializing the network stack and enabling system diagnostics
2. **Web Server:** HTTP server starts on port 80
3. **Network Setup:** Waits for **DHCP** to assign an IP address
4. **Form Display:** Users can access the web interface to see the input form
5. **Data Submission:** When users submit the form, data is sent via POST to `post.html`
6. **Processing:** The `RawPostCallBack` function processes the POST data
7. **Response:** The server responds with an HTML page showing the submitted data

12.55.16.1.4 Technical Details

12.55.16.1.4.1 POST Data Processing

- Uses `ReadSimpleBody()` to read POST data in chunks
- Supports multipart form data with boundary markers
- Handles buffer overflow situations gracefully
- Displays received data in real-time during processing

12.55.16.1.4.2 Buffer Management

- File buffer size: 10,000 bytes
- Streaming data processing to handle large submissions
- Proper memory management for continuous operation

12.55.16.1.4.3 Network Configuration

- Default HTTP port: 80
- **DHCP** network configuration
- 5-second timeout for network activation
- Continuous monitoring loop with 1-second intervals

12.55.16.1.5 Usage

1. Flash the application to a NetBurner device
2. Connect the device to a network with **DHCP**
3. Navigate to the device's IP address in a web browser
4. Enter text in the form field and click "Submit"
5. View the processed POST data in the response page

12.55.16.1.6 Development Environment

- Built for NetBurner 2pX platform
- Uses NetBurner NNDK (NetBurner Network Development Kit)
- Requires NetBurner development tools and libraries

12.55.16.1.7 Files

- `main.cpp` - Main application entry point and initialization
- `formcode.cpp` - HTTP POST request handler and form processing
- `index.html` - Web interface with input form

12.55.16.1.8 Notes

- System diagnostics are enabled for debugging (should be disabled in production)
- The application runs continuously in a loop
- Shows practical implementation of embedded web server POST handling

12.55.17 Remote Web Console for Serial/stdout

12.55.17.1 NetBurner Remote Web Console Example

12.55.17.1.1 Overview

This example demonstrates the **Remote Web Console** feature for NetBurner devices. The Remote Web Console creates a web-based interface that redirects the device's stdout output (normally sent to the USB serial port or DB9) to a webpage accessible at `http://your-device-ip/console.html`. For example, a `printf("↵ Hello Wolrd");` in the application will appear in the web page console window just as it would a serial port terminal.

The Remote Console can be added to any application with:

- `#include <remoteconsole.h>`
- `EnableRemoteConsole();`

This feature is particularly useful for:

- Remote debugging and monitoring
- Accessing device output without physical USB connection
- Web-based device interaction and command input
- Development and troubleshooting in network environments

12.55.17.1.2 Features

- **Web-based Console:** Access device output through any web browser
- **Real-time Output:** Stdout redirected to web interface
- **Interactive Commands:** Simple command processing demonstration
- **Network Accessibility:** Remote access via device IP address

12.55.17.1.3 Files Description

12.55.17.1.3.1 main.cpp

The main application file containing:

- Network stack initialization
- HTTP server startup
- Remote console enablement
- Simple command processing loop
- Periodic status output

12.55.17.1.3.2 ReadMe.txt

Documentation file explaining the Remote Web Console implementation and usage.

12.55.17.1.4 Implementation Details

12.55.17.1.4.1 Key Components

1. Network Initialization

```
init(); // Initialize network stack
StartHttp();
WaitForActiveNetwork(TICKS_PER_SECOND * 5);
```

2. Remote Console Setup

```
#include <remoteconsole.h>
EnableRemoteConsole();
```

3. Command Processing

- Listens for character input
- Responds to 'h' command with "Hello!"
- Provides feedback for unknown commands
- Outputs periodic status dots

12.55.17.1.4.2 Required Headers

- `<init.h>` - Network stack initialization
- `<nbrtos.h>` - NetBurner RTOS functions
- `<system.h>` - System utilities
- `<iosys.h>` - I/O system functions
- `<remoteconsole.h>` - Remote console functionality

12.55.17.1.5 Usage Instructions

12.55.17.1.5.1 Building and Deploying

1. Compile the Application

- Use NetBurner development tools to build the project
- Ensure all required libraries are linked

2. Deploy to Device

- Flash the compiled binary to your NetBurner device
- Connect the device to your network

3. Network Configuration

- Ensure the device obtains an IP address
- Note the device's IP address for web access

12.55.17.1.5.2 Accessing the Web Console

1. Open Web Browser

- Navigate to `http://[device-ip]/console.html`
- Replace `[device-ip]` with your device's actual IP address

2. Interact with Console

- View real-time output from the device
- Send commands through the web interface
- Monitor periodic status updates (dots every second)

12.55.17.1.5.3 Available Commands

- `**h**`: Displays "Hello!" message
- **Other characters**: Shows "Unknown command" message
- **Periodic Output**: Status dots appear every second

12.55.17.1.6 Technical Notes

12.55.17.1.6.1 System Diagnostics

```
EnableSystemDiagnostics();
```

- Enabled for development and debugging
- Consider disabling for production deployments

12.55.17.1.6.2 Timing and Delays

```
OSTimeDly(TICKS_PER_SECOND);
```

- Main loop executes every second
- Provides regular status updates
- Prevents excessive CPU usage

12.55.17.1.6.3 Network Dependencies

- Requires active network connection
- HTTP server must be running
- Device must be accessible via IP address

12.55.17.1.7 Customization Options

12.55.17.1.7.1 Extending Commands

Add more command processing in the main loop:

```
if (c == 'h')
{
    printf("\r\nHello!\r\n");
}
else if (c == 's')
{
    printf("\r\nStatus: Running\r\n");
}
// Add more commands here
```

12.55.17.1.7.2 Modifying Output Frequency

Change the delay value to adjust status update frequency:

```
OSTimeDly(TICKS_PER_SECOND * 2); // 2-second intervals
```

12.55.17.1.7.3 Adding Application Logic

Insert your application-specific code within the main loop while preserving the console functionality.

12.55.17.1.8 Troubleshooting

12.55.17.1.8.1 Common Issues

1. Cannot Access Web Console

- Verify device IP address
- Check network connectivity
- Ensure HTTP server is running
- Confirm firewall settings

2. No Output Visible

- Check if RemoteConsole is enabled
- Verify printf statements are executing
- Ensure web page is properly loaded

3. Commands Not Responding

- Verify character input handling
- Check if `charavail()` is functioning
- Ensure main loop is executing

12.55.17.1.8.2 Debug Steps

1. Monitor USB serial output during development
2. Use network tools to verify device connectivity
3. Check HTTP server status and logs
4. Verify application is running and not stuck

12.55.18 Serial Webserver

12.55.18.1 Serial Webserver

A NetBurner RTOS application that bridges serial communication with HTTP web services, allowing HTTP requests to be received and processed over a serial port connection.

12.55.18.1.1 Overview

The Serial Webserver demonstrates how to create a bidirectional communication bridge between a serial port and TCP/HTTP connections. This application enables HTTP requests to be sent over a serial connection and receive corresponding web responses back through the same serial interface.

It will receive and reply to HTTP requests over a serial port, using the NetBurner web server implementation. The example will display a Hello World message on the device's index page `index.html`.

12.55.18.1.2 Key Features

- **Serial-to-HTTP Bridge:** Receives HTTP requests via serial port and forwards them to the built-in web server
- **Bidirectional Communication:** Splices data between serial and TCP connections in both directions
- **Automatic Connection Management:** Handles TCP connection creation and cleanup automatically
- **Error Recovery:** Automatically reopens serial connections on errors
- **Built-in Web Server:** Serves static content including HTML, CSS, and images

12.55.18.1.3 Technical Specifications

- **Serial Port:** Port 1 at 115200 baud
- **HTTP Port:** Standard port 80
- **Buffer Size:** Uses Ethernet buffer size for data transfer
- **Network Stack:** NetBurner RTOS with TCP/IP support

12.55.18.1.4 Application Architecture

The application follows a simple event-driven architecture:

1. **Initialization:** Sets up network stack, enables diagnostics, and starts HTTP server
2. **Serial Monitoring:** Waits for incoming serial data
3. **Connection Creation:** Establishes TCP connection to local web server when data arrives
4. **Data Splicing:** Uses `select()` to monitor both serial and TCP file descriptors
5. **Bidirectional Transfer:** Forwards data between serial and TCP connections
6. **Error Handling:** Closes and reopens connections on errors

12.55.18.1.5 HTTP Request Format

The application expects standard HTTP requests over the serial connection. A typical request format:

```
GET /INDEX.HTML HTTP/1.1\r\n\r\n
```

Important**: The web server requires a double carriage return line feed (`\r\n\r\n`) to properly parse HTTP requests.

12.55.18.1.6 Expected Response

A successful request to the index page returns:

```
HTTP/1.0 200 OK
Pragma: no-cache
MIME-version: 1.0
Content-Type: text/html
Content-length: 231
```

```
<html>
<head><link rel="stylesheet" type="text/css" href="style.css"></head>
<body>
<a href="http://www.netburner.com"></a>
<h1>Thank you for NetBurning!</h1>
</body>
</html>
```

12.55.18.1.7 Web Content

The application serves the following static files:

- **index.html:** Main page with NetBurner branding and welcome message
- **style.css:** Stylesheet defining Arial font family for the page
- **logo.jpg:** NetBurner logo image (referenced but not included in source)

12.55.18.1.8 Usage

1. Flash the application to your NetBurner device
2. Connect to the device's serial port (Port 1, 115200 baud)
3. Send HTTP requests through the serial connection
4. Receive HTTP responses back through the same serial connection

12.55.18.1.9 Development Notes

- System diagnostics are enabled for development (consider removing for production)
- The application uses blocking I/O operations
- Memory allocation uses `FAST_USER_VAR` for the splice buffer

12.55.18.1.10 Use Cases

This application is ideal for:

- Testing HTTP communication through serial interfaces
- Bridging legacy serial devices to modern web services
- Remote web server access through serial connections

12.55.18.1.11 Build Requirements

- NetBurner NBRTOS development environment
- Standard NetBurner libraries (init, system, iosys, serial, tcp, netinterface)
- Compatible NetBurner hardware platform

12.55.19 Simple HTML

12.55.19.1 NetBurner Simple HTML Example

12.55.19.1.1 Overview

The Simple HTML Example is a basic demonstration of the NetBurner web server implementation and serves as an excellent starting point for custom embedded web applications. This example creates a minimal web server that displays a "Hello World" type message on the device's index page.

12.55.19.1.2 Application Description

This embedded web application initializes a NetBurner device to serve HTML content over HTTP. The application demonstrates the fundamental components needed to create a web-enabled embedded system using the NetBurner Real-Time Operating System (NBRTOS).

12.55.19.1.2.1 Key Features

- **Web Server:** Runs an HTTP server on the default port 80
- **Network Stack:** Initializes the complete network stack for web connectivity
- **System Diagnostics:** Includes optional system diagnostic capabilities
- **Real-Time Operation:** Built on NBRTOS for reliable real-time performance
- **Simple Interface:** Displays a basic HTML page with NetBurner branding

12.55.19.1.3 File Structure

```
/
main.cpp      - Main application entry point
index.html    - Default web page served by the application
style.css     - CSS styling for the web interface
```


12.55.19.1.4 How It Works

12.55.19.1.4.1 Application Flow

1. **Initialization:** The `UserMain()` function initializes the network stack
2. **Diagnostics:** Enables system diagnostics for debugging (should be removed in production)
3. **Web Server:** Starts the HTTP server on port 80
4. **Network Wait:** Waits up to 5 seconds for active network connection
5. **Status Display:** Prints application name and NNDK revision to console
6. **Main Loop:** Enters infinite loop with 1-second delays

12.55.19.1.4.2 Web Content

The application serves a simple HTML page (`index.html`) that includes:

- NetBurner logo and branding
- Basic styling via CSS
- "Thank you for NetBurning!" message
- Link to NetBurner website

12.55.19.1.5 Technical Details

12.55.19.1.5.1 Dependencies

- **NBRTOS:** NetBurner Real-Time Operating System
- **Network Stack:** Built-in TCP/IP stack
- **HTTP Server:** Integrated web server functionality

12.55.19.1.5.2 System Requirements

- NetBurner compatible hardware
- Network connectivity (Ethernet or WiFi depending on platform)
- NNDK (NetBurner Network Development Kit)

12.55.19.1.6 Usage

1. **Compile:** Build the application using the NetBurner development environment
2. **Deploy:** Load the application onto your NetBurner device
3. **Connect:** Ensure the device is connected to your network
4. **Access:** Open a web browser and navigate to the device's IP address
5. **View:** The simple HTML page will be displayed

12.55.19.1.7 Development Notes

12.55.19.1.7.1 Production Considerations

- Remove `EnableSystemDiagnostics()` call for production deployments
- Customize the HTML content in `index.html` for your specific application
- Modify CSS styling in `style.css` to match your branding
- Add additional web pages and functionality as needed

12.55.19.1.7.2 Customization

This example serves as a foundation for more complex web applications. You can:

- Add dynamic content generation
- Implement form handling
- Create RESTful APIs
- Add authentication and security features
- Integrate with sensors and other hardware

12.55.19.1.8 Console Output

When running, the application displays:

```
Web Application: Simple HTML Example  
NNDK Revision: [Version Information]
```

12.55.20 TicTacToe

12.55.20.1 NetBurner TicTacToe Web Application

12.55.20.1.1 Overview

This is a web-based TicTacToe game designed for NetBurner development boards. The application demonstrates embedded web server capabilities, interactive HTML content, and real-time gameplay through a web browser interface.

12.55.20.1.2 Features

12.55.20.1.2.1 Core Functionality

- **Interactive TicTacToe Game:** Play against the computer through a web interface
- **Web Server:** Built-in HTTP server running on port 80
- **LED Manipulation:** Control LEDs on the NetBurner development board (URL encoding example)
- **Request Echo:** View web browser requests sent to the server
- **Auto-Refresh Pages:** Meta tag example for automatically reloading web pages

12.55.20.1.2.2 Technical Capabilities

- **Network Stack:** Full TCP/IP implementation with [DHCP](#) support
- **Serial Console:** Debug interface with command-line utilities
- **Diagnostic Tools:** Network diagnostics, [ARP](#) tables, TCP debugging
- **Real-time Updates:** Dynamic content generation and state management

12.55.20.1.3 Game Rules

- **Player Symbol:** X (Cross)
- **Computer Symbol:** O (Circle)
- **Objective:** Get three symbols in a row (horizontal, vertical, or diagonal)
- **Turn Order:** Player moves first by clicking on empty squares
- **Game State:** Preserved in URL parameters between moves

12.55.20.1.4 File Structure

```

/
main.cpp           # Main application entry point and command processor
tictactoe.cpp      # TicTacToe game logic and web handlers
index.html         # Default homepage with navigation links
tt.htm            # TicTacToe game interface
once.htm          # Single-load counter demonstration
repeat.htm        # Auto-refreshing page example

```

12.55.20.1.5 Web Interface

12.55.20.1.5.1 Available Pages

- / or /index.html: Main navigation page
- /TT.HTM: TicTacToe game interface
- /REPEAT.HTM: Auto-refreshing demonstration page
- /ONCE.HTM: Page load counter example

12.55.20.1.5.2 Game Assets

- CROSS.GIF: X symbol image
- NOT.GIF: O symbol image
- BLANK.GIF: Empty square image

12.55.20.1.6 Command Line Interface

The application provides a serial console interface with the following commands:

Command	Description
A	Show ARP table
C	Show network counters
E	Dump Ethernet statistics
P x.x.x.↵ x.x	Ping specified IP address
W	Display current system time
?	Show help menu

12.55.20.1.6.1 Debug Commands (when _DEBUG is defined)

Command	Description
T	Dump TCP debug information
L	Setup debug log levels
U	Dump OS tasks
S	Dump OS stacks
H	Show task changes

12.55.20.1.7 Technical Implementation

12.55.20.1.7.1 Game Logic

The TicTacToe game uses a bit-encoding system to represent the board state:

Player (X) Positions:**
 0x01 0x04 0x10
 0x40 0x100 0x400
 0x1000 0x4000 0x10000

Computer (O) Positions:**
0x02 0x08 0x20
0x80 0x200 0x800
0x2000 0x8000 0x20000

12.55.20.1.7.2 State Management

- Game state is encoded in URL parameters
- Each move generates a new URL with updated state
- Computer moves are calculated using strategic algorithms
- Win/lose conditions are checked after each move

12.55.20.1.7.3 Web Server Integration

- Uses NetBurner's built-in HTTP server
- Dynamic content generation through function calls
- HTML templates with embedded C++ function calls
- Real-time state updates without page refresh

12.55.20.1.8 Hardware Requirements

- NetBurner development board with network capability
- Ethernet connection for web access
- Serial port for debug console (115200 baud)
- LEDs for visual feedback (board-dependent)

12.55.20.1.9 Network Configuration

- **DHCP**: Automatically obtains IP address
- **Web Server**: Port 80 (HTTP)
- **Serial Console**: Available for diagnostics
- **Ping Support**: Built-in network connectivity testing

12.55.20.1.10 Getting Started

1. **Setup**: Connect NetBurner board to network via Ethernet
2. **Power On**: Board will automatically start and obtain **DHCP** address
3. **Access**: Open web browser and navigate to board's IP address
4. **Play**: Click "Play Tic Tak Toe" to start the game
5. **Debug**: Use serial console for network diagnostics

12.56 Web Client

Web Socket Examples:

- [Earthquake](#)
- [Find My IP](#)
- [Find My IP Task](#)
- [WebClient Http Request Methods](#)
- [Message Passer](#)
- [MessagePasserTask JSON](#)

12.56.1 Earthquake

12.56.1.1 Earthquake Monitor Application

12.56.1.1.1 Overview

This is a simple C++ embedded application that demonstrates real-time earthquake monitoring by retrieving and parsing JSON data from the USGS (United States Geological Survey) earthquake feed. The application fetches information about significant earthquakes (magnitude 4.5 and above) that have occurred in the last 24 hours and displays detailed information about each event.

12.56.1.1.2 Features

- **Real-time Data:** Fetches live earthquake data from USGS API
- **Time Synchronization:** Uses NTP (Network Time Protocol) to set accurate system time
- **JSON Parsing:** Processes earthquake data in GeoJSON format
- **Detailed Reporting:** Shows magnitude, location, and time elapsed since each earthquake
- **Network Stack Integration:** Built for embedded systems with network capabilities

12.56.1.1.3 Data Source

The application retrieves data from:

https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/4.5_day.geojson

This endpoint provides all earthquakes with magnitude 4.5 or greater from the past day in GeoJSON format.

12.56.1.1.4 Output Example

```
Application started
Time Set
In the last Day we have had 12 Earthquakes > 4.5
Magnitude : 5.2 at 34km WSW of Ferndale, CA 02:15:32 ago
Magnitude : 4.7 at 89km NNE of Adak, Alaska 05:42:18 ago
Magnitude : 4.9 at South of the Fiji Islands 08:30:45 ago
...
```

12.56.1.1.5 Technical Details

12.56.1.1.5.1 Dependencies

- Network stack initialization ([init.h](#))
- HTTP client functions ([webclient/http_funcs.h](#))
- JSON lexer and parser ([json_lexer.h](#))
- NTP time synchronization ([nbtime.h](#))
- Buffer management ([buffers.h](#))

12.56.1.1.5.2 Key Components

1. **Network Initialization:** Sets up the network stack and waits for [DHCP](#)
2. **Time Synchronization:** Uses NTP to get accurate system time
3. **HTTP Client:** Makes GET requests to the USGS API
4. **JSON Parser:** Processes the GeoJSON response format
5. **Data Display:** Formats and presents earthquake information

12.56.1.1.5.3 Application Flow

1. Initialize network stack and enable diagnostics
2. Wait for active network connection (up to 5 seconds)
3. Synchronize system time using NTP
4. Fetch earthquake data from USGS API
5. Parse JSON response and extract earthquake count
6. Iterate through each earthquake event and display:
 - Magnitude (formatted to 1 decimal place)
 - Location description
 - Time elapsed since the event (if time sync successful)
7. Enter infinite loop with 1-second delays

12.56.1.1.5.4 JSON Data Structure

The application expects GeoJSON format with the following structure:

```
{
  "metadata": {
    "count": number_of_earthquakes
  },
  "features": [
    {
      "properties": {
        "mag": earthquake_magnitude,
        "place": "location_description",
        "time": timestamp_in_milliseconds
      }
    }
  ]
}
```

12.56.1.1.6 Error Handling

- **Network Issues:** Reports "Failed to contact server" if HTTP request fails
- **Parse Errors:** Individual earthquake entries that fail to parse are reported with error messages
- **Time Sync:** Application continues without time information if NTP sync fails

12.56.1.1.7 Usage

This application is designed for embedded systems and runs continuously, fetching earthquake data once at startup and then entering a maintenance loop. It's suitable for:

- Environmental monitoring systems
- Educational demonstrations of web API integration
- Real-time earthquake awareness systems
- JSON parsing examples in embedded contexts

12.56.1.1.8 Configuration

The earthquake magnitude threshold and time period can be modified by changing the API endpoint URL. Available USGS feed options include:

- `all_hour.geojson` - All earthquakes in the past hour
- `2.5_day.geojson` - Magnitude 2.5+ in the past day
- `4.5_week.geojson` - Magnitude 4.5+ in the past week
- `significant_month.geojson` - Significant earthquakes in the past month

12.56.1.1.9 Build Notes

Ensure all required header files and libraries are available in your build environment. The application uses proprietary networking and JSON parsing libraries that must be linked during compilation.

12.56.2 Find My IP

12.56.2.1 FindMyIP Application

12.56.2.1.1 Overview

FindMyIP is a NetBurner embedded application that demonstrates network connectivity by retrieving and displaying the device's public IP address. The application connects to an external web service to determine the IP address that the device appears to have from the internet's perspective.

12.56.2.1.2 Description

This program creates a higher-priority task that performs an HTTP GET request to retrieve network information. Specifically, it connects to <http://httpbin.org/ip> and parses the JSON response to extract the device's public IP address.

12.56.2.1.3 Features

- **Network Connectivity Testing:** Verifies that the device can successfully connect to external web services
- **Public IP Detection:** Retrieves the public-facing IP address of the device
- **JSON Response Parsing:** Demonstrates parsing of JSON data returned from web services
- **System Diagnostics:** Includes diagnostic capabilities for debugging network issues

12.56.2.1.4 How It Works

1. **Initialization:** The application initializes the system and enables diagnostics
2. **Network Wait:** Waits up to 5 seconds for an active network connection ([DHCP](#))
3. **HTTP Request:** Performs a GET request to `httpbin.org/ip`
4. **JSON Parsing:** Extracts the "origin" field from the JSON response
5. **Display Results:** Prints the discovered IP address to the console
6. **Continuous Loop:** Enters an infinite loop with 1-second delays

12.56.2.1.5 Code Structure

12.56.2.1.5.1 Main Components

- **Network Interface:** Uses NetBurner's network stack for HTTP communication
- **JSON Lexer:** Parses JSON responses from the web service
- **HTTP Functions:** Handles web client operations
- **System Initialization:** Sets up the embedded system environment

12.56.2.1.5.2 Key Functions

- `init()`: Initializes the system
- `EnableSystemDiagnostics()`: Enables debugging output
- `WaitForActiveNetwork()`: Waits for network connectivity
- `DoGet()`: Performs HTTP GET request and parses JSON response

12.56.2.1.6 Usage

When the application runs, you will see output similar to:

```
Application started  
My IP address is 203.0.113.42
```

If the connection fails, you will see:

```
Application started  
Failed to contact server
```

12.56.2.1.7 Dependencies

- NetBurner RTOS and network stack
- HTTP client library (`webclient/http_funcs.h`)
- JSON parsing library (`json_lexer.h`)
- Serial communication support

12.56.2.1.8 Configuration

The target URL is configured as a constant:

```
const char *url = "http://httpbin.org/ip";
```

This can be modified to use different IP detection services that return JSON responses.

12.56.2.1.9 Notes

- System diagnostics are enabled by default and should be disabled for production builds
- The application waits 5 seconds for network connectivity before attempting the request
- The main loop includes a 1-second delay to prevent excessive CPU usage
- Error handling is included for network connectivity issues

12.56.3 Find My IP Task

12.56.3.1 Find My IP Application

12.56.3.1.1 Overview

The Find My IP application is a C++ embedded system program that demonstrates web client functionality by retrieving and displaying the external IP address of the device. This application creates a separate, higher-priority task that navigates to a web service and retrieves IP address information.

12.56.3.1.2 Purpose

This program serves as an example of web client implementation on an embedded system, specifically showcasing how to:

- Initialize and configure network stack
- Make HTTP GET requests to external web services
- Parse JSON responses
- Handle network errors gracefully
- Implement task-based architecture with proper priority management

12.56.3.1.3 How It Works

1. **Network Initialization:** The application initializes the network stack and waits for an active network connection (DHCP address assignment)
2. **Web Client Setup:** Configures a web client task with higher priority (MAIN_PRIO - 2) to handle HTTP requests
3. **HTTP Request:** Makes a GET request to <http://httpbin.org/ip> which returns JSON data containing the client's external IP address
4. **JSON Parsing:** Parses the JSON response to extract the "origin" field containing the IP address
5. **Display Results:** Prints the retrieved IP address to the console
6. **Error Handling:** Implements comprehensive error handling with descriptive error messages

12.56.3.1.4 Key Features

- **Asynchronous Operation:** Uses separate tasks for network operations
- **JSON Support:** Built-in JSON parsing capabilities
- **Error Reporting:** Comprehensive error handling and reporting system
- **Network Diagnostics:** Optional system diagnostics for debugging
- **Configurable Timing:** 10-second intervals between operations

12.56.3.1.5 Technical Details

12.56.3.1.5.1 Dependencies

- Network stack initialization ([init.h](#))
- HTTP client functionality ([webclient/http_funcs.h](#), [webclient/web_client.h](#))
- JSON parsing ([json_lexer.h](#))
- Buffer management ([buffers.h](#))

12.56.3.1.5.2 Key Functions

- `UserMain()`: Main application entry point
- `DoActualClientRequest()`: Handles the actual HTTP request and JSON parsing
- `HandleErrorState()`: Processes and displays error conditions

12.56.3.1.5.3 Configuration

- **Target URL:** <http://httpbin.org/ip>
- **Application Name:** "FindMyIp"
- **Network Timeout:** 5 seconds for DHCP acquisition
- **Request Interval:** 10 seconds between requests
- **Task Priority:** MAIN_PRIO - 2 (higher than main task)

12.56.3.1.6 Usage

The application runs continuously once started, making periodic requests to determine the external IP address. This is useful for:

- Network connectivity testing
- IP address monitoring
- Embedded system network diagnostics
- Learning web client implementation patterns

12.56.3.1.7 Output

When successful, the application displays:

```
Application started  
My IP address is [YOUR_EXTERNAL_IP]
```

Error conditions are reported with descriptive messages including error codes and descriptions.

12.56.3.1.8 Notes

- System diagnostics are enabled by default but should be disabled for production code
- The application uses standard HTTP (not HTTPS) for simplicity
- Designed for embedded systems with real-time operating system (RTOS) support
- Requires active internet connection and DNS resolution capability

12.56.4 WebClient Http Request Methods

12.56.4.1 HTTP Methods Test Application

12.56.4.1.1 Overview

This repository contains two complementary networking applications designed for testing and development purposes:

1. **HTTP Methods Test Client** (main.cpp) - A NetBurner embedded C++ application that tests various HTTP methods against a test server
2. **HTTP Echo Server** (main.go) - A Go-based HTTP server that echoes back request details in both HTML and JSON formats

12.56.4.1.2 HTTP Methods Test Client (main.cpp)

12.56.4.1.2.1 Description

The HTTP Methods Test Client is a NetBurner embedded application written in C++ that systematically tests different HTTP methods against a configurable endpoint. The application demonstrates proper usage of NetBurner's HTTP client libraries and JSON handling capabilities.

12.56.4.1.2.2 Features

- **Multiple HTTP Method Testing:** Tests POST, PUT, GET, HEAD, and DELETE methods
- **JSON Request/Response Handling:** Sends structured JSON data and parses responses
- **HTML Content Retrieval:** Can request and display HTML responses
- **Device Information:** Includes MAC address and uptime in requests
- **Automated Testing Loop:** Continuously cycles through different HTTP methods
- **Network Diagnostics:** Optional HTTP debugging output
- **Configurable Target:** Easy to change target URL for different test servers

12.56.4.1.2.3 HTTP Methods Tested

1. **POST** - Sends JSON message with device info and timestamp
2. **PUT** - Sends JSON message with device info and timestamp
3. **GET** - Requests HTML echo response
4. **HEAD** - Requests headers-only response
5. **DELETE** - Sends DELETE request for testing

12.56.4.1.2.4 Configuration

- **Target URL:** Currently set to `http://10.2.0.63:20035/`
- **Request Headers:** Configurable Accept headers (application/json or text/html)
- **Timeout:** 10 seconds per request
- **Test Interval:** 2 seconds between each method test

12.56.4.1.2.5 JSON Request Format

The application sends JSON requests with the following structure:

```
{
  "DEVICEID": "XX:XX:XX:XX:XX:XX",
  "UPTIME": 12345,
  "MESSAGE": "At the tone the time will be: 12345"
}
```

12.56.4.1.2.6 Dependencies

- NetBurner RTOS and libraries
- [init.h](#) - System initialization
- [buffers.h](#) - Buffer management
- [json_lexer.h](#) - JSON parsing
- [webclient/http_funcs.h](#) - HTTP client functions
- [netinterface.h](#) - Network interface management

12.56.4.1.2.7 Usage

1. Configure the target URL in the source code
2. Build and deploy to NetBurner hardware
3. Ensure network connectivity
4. Application will automatically start testing HTTP methods
5. Monitor output via serial console or system diagnostics

12.56.4.1.3 HTTP Echo Server (main.go)

12.56.4.1.3.1 Description

The HTTP Echo Server is a Go-based web server designed to echo back HTTP request details. It serves as a perfect companion for testing HTTP clients by providing detailed information about received requests in both human-readable HTML and machine-readable JSON formats.

12.56.4.1.3.2 Features

- **Content Negotiation:** Returns HTML or JSON based on Accept headers
- **Complete Request Echo:** Displays method, resource, headers, body, and raw request
- **Template-Based Responses:** Uses Go templates for flexible output formatting
- **Configurable Port:** Uses SERVER_PORT environment variable or default port 20035
- **Error Handling:** Proper HTTP error responses for template or processing errors

12.56.4.1.3.3 Response Formats

HTML Response

Provides a formatted web page showing:

- HTTP Method used
- Resource/URI requested
- All request headers with values
- Request body content
- Complete raw HTTP request dump

JSON Response

Returns structured JSON with:

- Method string
- Resource/URI string
- Headers object with arrays for multi-value headers
- Body content as string
- Complete raw request as string

12.56.4.1.3.4 Configuration

- **Default Port:** 20035
- **Environment Variable:** SERVER_PORT overrides default port
- **Content Type Detection:** Automatically detects JSON requests via Accept header

12.56.4.1.3.5 Running the Server

```
# Use default port 20035
go run main.go

# Use custom port via environment variable
SERVER_PORT=8080 go run main.go
```

12.56.4.1.3.6 Template System

The server uses Go's template system with custom functions:

- `set`: Sets a counter value
- `incr`: Increments a counter value

This allows for complex template logic and formatting of response data.

12.56.4.1.4 Integration Testing

These two applications work together to provide a complete HTTP testing environment:

1. **Start the Go echo server** on the target network
2. **Configure the NetBurner client** with the server's IP address and port
3. **Deploy and run the client** - it will automatically begin testing
4. **Monitor responses** via the client's serial output and server logs

12.56.4.1.5 Network Requirements

- Both applications should be on the same network or have routing between them
- Default port 20035 should be available (or configure alternative)
- NetBurner device needs [DHCP](#) or static IP configuration
- Firewall rules should allow HTTP traffic between client and server

12.56.4.1.6 Development Notes

12.56.4.1.6.1 NetBurner Client Development

- Uses NetBurner's RTOS and networking stack
- Requires proper initialization and network waiting
- JSON handling through NetBurner's [ParsedJsonDataSet](#) class
- HTTP operations through DoJsonHttp and DoHttpEx functions

12.56.4.1.6.2 Go Server Development

- Standard Go HTTP server with template system
- Cross-platform compatibility
- Minimal dependencies (standard library only)
- Easy to extend with additional endpoints or features

12.56.4.1.7 Troubleshooting

12.56.4.1.7.1 Common Issues

1. **Network Connectivity:** Ensure both devices can reach each other
2. **Port Conflicts:** Check if port 20035 is available or configure alternative
3. **Timeout Issues:** Adjust timeout values if network is slow
4. **JSON Parsing Errors:** Verify JSON format compatibility
5. **Template Errors:** Check server logs for template parsing issues

12.56.4.1.7.2 Debug Features

- NetBurner client: Enable SetHttpDiag(true) for verbose HTTP debugging
- Go server: Built-in request logging to console
- System diagnostics available on NetBurner platform

12.56.5 Message Passer

12.56.5.1 MessagePasser

A real-time messaging application that periodically sends timestamped messages to a remote server via HTTP POST requests.

12.56.5.1.1 Overview

MessagePasser is an embedded C++ application designed to communicate with a remote web service. The application continuously sends JSON-formatted messages containing device identification, uptime information, and timestamped messages to a specified HTTP endpoint.

12.56.5.1.2 Features

- **Automatic Device Identification:** Uses the device's MAC address as a unique identifier
- **Real-time Communication:** Sends HTTP POST requests with JSON payloads
- **Uptime Tracking:** Reports system uptime with each message
- **Network Integration:** Waits for active network connection ([DHCP](#)) before operation
- **Periodic Messaging:** Sends messages every 10 seconds automatically
- **JSON Response Handling:** Processes server responses and displays success status

12.56.5.1.3 Application Flow

1. **Initialization:** System initialization and diagnostics enablement
2. **Network Setup:** Waits up to 5 seconds for active network connection
3. **Message Loop:** Continuous loop that:
 - Creates timestamped messages
 - Sends JSON data to remote server
 - Processes server responses
 - Waits 10 seconds before next iteration

12.56.5.1.4 Message Format

The application sends JSON messages with the following structure:

```
{
  "DEVICEID": "AABBCCDDEEFF",
  "UPTIME": 12345,
  "MESSAGE": "At the tone the time will be: 12345"
}
```

Where:

- **DEVICEID:** 12-character hexadecimal string representing the device MAC address
- **UPTIME:** Current system uptime in seconds
- **MESSAGE:** Timestamped message string

12.56.5.1.5 Server Endpoints

The application is configured to communicate with:

- Primary: <http://burnedcloud.herokuapp.com/messagehandler>
- Alternative (commented): <http://66.27.60.212:9090/>

12.56.5.1.6 Implementation Details

12.56.5.1.6.1 Key Functions

- `**SendRxMessage(const char *msg)**`: Modern JSON object-based message sending
- `**SendManualRxMessage(const char *msg)**`: Manual JSON string formatting (alternative implementation)
- `**UserMain(void *pd)**`: Main application entry point and message loop

12.56.5.1.6.2 Dependencies

The application requires several system libraries:

- `init.h`: System initialization
- `stdio.h`: Standard I/O operations
- `ctype.h`: Character type functions
- `buffers.h`: Buffer management
- `json_lexer.h`: JSON parsing capabilities
- `webclient/http_funcs.h`: HTTP client functionality
- `netinterface.h`: Network interface management

12.56.5.1.6.3 Network Requirements

- Active network interface with `DHCP` configuration
- Internet connectivity to reach the remote server
- HTTP POST capability

12.56.5.1.7 Configuration

12.56.5.1.7.1 Timing Parameters

- **Network Wait**: 5 seconds maximum for `DHCP` address acquisition
- **Message Interval**: 10 seconds between transmissions
- **HTTP Timeout**: 10 seconds for server response

12.56.5.1.7.2 Customization Options

- Server URL can be modified by changing the `Url` constant
- Message content can be customized in the `UserMain` function
- Timing intervals can be adjusted by modifying the `OSTimeDly` parameters

12.56.5.1.8 Output

The application provides console output including:

- Application startup confirmation
- Success/failure status for each HTTP request
- Server response messages
- Error notifications for failed requests

12.56.5.1.9 Use Cases

This application is suitable for:

- IoT device monitoring and reporting
- Remote system status updates
- Periodic data collection from embedded devices
- Network connectivity testing and validation

12.56.5.1.10 Technical Notes

- Built for embedded systems with real-time OS capabilities
- Uses system ticks for timing (`TICKS_PER_SECOND`)
- Implements proper JSON object building and parsing
- Includes system diagnostics for development/debugging

12.56.6 MessagePasserTask JSON

12.56.6.1 MessagePasserTask Application

12.56.6.1.1 Overview

MessagePasserTask is a NetBurner embedded application that demonstrates JSON-based client-server communication. The application periodically sends status information to a web server via HTTP POST requests and processes the server's responses to control local hardware.

12.56.6.1.2 Purpose

This application serves as an example of how to:

- Implement web client functionality on embedded NetBurner devices
- Send and receive JSON data over HTTP
- Integrate hardware I/O (switches and LEDs) with network communication
- Handle server responses and error conditions

12.56.6.1.3 Key Features

12.56.6.1.3.1 Network Communication

- Makes HTTP POST requests to a configurable server endpoint
- Sends JSON data containing device status information
- Processes JSON responses from the server
- Includes comprehensive error handling for network operations

12.56.6.1.3.2 Hardware Integration

- Reads DIP switch positions to monitor user input
- Controls LEDs based on server responses
- Automatically triggers network requests when switch positions change

12.56.6.1.3.3 Data Exchange

The application sends the following information to the server:

- Device MAC address (as DEVICEID)
- System uptime in seconds
- Current timestamp message
- Student name identifier
- Current switch position values

The server responds with:

- Success flag indicating message processing status
- Current switch value for LED control
- Response message for display

12.56.6.1.4 Configuration

12.56.6.1.4.1 Server URL

The target server URL is configurable via the `Url` config object:

```
static config_string Url(appdata, "http://class.netburner.com/post/", "MessageUrl");
```

12.56.6.1.4.2 Student Identity

Update the student name before deployment:

```
const char *StudentName = "Attendee Name";
```

12.56.6.1.5 Application Flow

1. Initialization

- Initialize NetBurner system and network stack
- Wait for active network connection ([DHCP](#))
- Display IP address information
- Start web client task

2. Main Loop

- Monitor DIP switch positions
- Trigger network request when switches change
- Send periodic status updates every 10 seconds

3. Network Request Process

- Build JSON object with current status
- Send HTTP POST to configured server
- Parse JSON response
- Update LEDs based on server response
- Display status messages

12.56.6.1.6 Error Handling

The application includes robust error handling:

- Network connection timeouts (10 second limit)
- JSON parsing errors
- Server response validation
- Hardware I/O error recovery

Error states are logged to the console with descriptive messages using the `HandleErrorState()` function.

12.56.6.1.7 Hardware Requirements

- NetBurner development board with network capability
- DIP switches for input
- LEDs for status output
- Network connection (Ethernet)

12.56.6.1.8 Build Dependencies

Required NetBurner libraries:

- `init.h` - System initialization
- `webclient/web_client.h` - HTTP client functionality
- `json_lexer.h` - JSON parsing and generation
- `config_obj.h` - Configuration management
- `ipshow.h` - Network diagnostics
- `utils.h` - Hardware I/O utilities

12.56.6.1.9 Usage Notes

- The application automatically starts network communication after initialization
- LED patterns indicate server response status
- Switch changes immediately trigger new server requests
- Console output provides detailed operation logging
- System diagnostics are enabled for debugging (should be disabled in production)

12.56.6.1.10 Troubleshooting

Common issues and solutions:

- **Network timeouts:** Check server availability and network connection
- **JSON parsing errors:** Verify server response format matches expected structure
- **LED not responding:** Check server CURSWITCH value in response
- **No SUCCESS flag:** Server may be rejecting the request format

Enable HTTP diagnostics for detailed network troubleshooting:

```
SetHttpDiag(true);
```

12.56.6.1.11 Development Notes

This application demonstrates best practices for NetBurner embedded web clients and can serve as a foundation for more complex IoT communication scenarios.

12.57 Web Sockets

Web Socket Examples:

- [Web Socket Connect](#)
- [Web Socket Console](#)
- [Web Socket DIP Switch](#)
- [Web Socket Echo](#)
- [HTTPS WebSocket Demo](#)

12.57.1 Web Socket Connect

12.57.1.1 WebSocket Connect Application

12.57.1.1.1 Overview

This application demonstrates a WebSocket client implementation that connects to a server and maintains a persistent connection for real-time communication.

12.57.1.1.2 Features

- **WebSocket Client Connection:** Establishes a connection to a WebSocket server
- **Real-time Data Exchange:** Sends periodic time tick messages to the server
- **Asynchronous Socket Handling:** Uses a dedicated task for reading incoming data
- **Network Stack Integration:** Built on NetBurner's network initialization framework
- **Error Handling:** Monitors socket status and handles connection errors gracefully

12.57.1.1.3 Application Structure

The application consists of two main components:

12.57.1.1.3.1 Main Task (UserMain)

- Initializes the network stack and waits for [DHCP](#)
- Establishes WebSocket connection to the specified server
- Sends periodic "TimeTick" messages every second
- Creates a separate task for handling incoming socket data

12.57.1.1.3.2 Socket Handling Task (UserTask)

- Monitors the WebSocket connection using [select \(\)](#)
- Handles incoming data from the server
- Manages socket errors and connection cleanup
- Echoes received data to stdout

12.57.1.1.4 Configuration

The application is configured with the following default settings:

```
#define SERVER_IP "192.168.1.209"
```

- **Server IP:** 192.168.1.209
- **WebSocket Path:** "/"
- **Port:** 80
- **Message Interval:** 1 second

12.57.1.1.5 Key Functions

12.57.1.1.5.1 Connection Management

- `NB::WebSocket::Connect()`: Establishes WebSocket connection
- `NB::WebSocket::ws_flush()`: Flushes WebSocket buffers

12.57.1.1.5.2 Network Operations

- `select()`: Monitors socket file descriptors for activity
- `read()`: Receives data from the WebSocket
- `writestring()`: Sends string data through the WebSocket
- `writeall()`: Outputs received data to stdout

12.57.1.1.5.3 System Integration

- `init()`: Initializes the NetBurner network stack
- `WaitForActiveNetwork()`: Waits for network connectivity
- `OSSimpleTaskCreatewName()`: Creates the socket handling task

12.57.1.1.6 Message Format

The application sends periodic status messages in the format:

```
TimeTick: [current_tick_value]
```

Where `[current_tick_value]` is the current system tick count, providing a timestamp for each message.

12.57.1.1.7 Error Handling

The application includes robust error handling:

- Socket error detection using `select()` with error file descriptor sets
- Automatic socket cleanup on connection failure
- Graceful task termination when connection is lost

12.57.1.1.8 Build Requirements

- NetBurner development environment
- WebSocket library (`websockets.h`)
- Standard C libraries
- NetBurner initialization framework (`init.h`)

12.57.1.1.9 Usage

1. Configure the `SERVER_IP` define to match your WebSocket server
2. Build and deploy to your NetBurner device
3. The application will automatically:
 - Initialize the network stack
 - Connect to the specified WebSocket server
 - Begin sending periodic time tick messages
 - Display any received messages on the console

12.57.1.1.10 Technical Details

- **Task Priority:** Socket handling task runs at `MAIN_PRIO + 1`
- **Buffer Size:** Uses `ETHER_BUFFER_SIZE` for data reception
- **Timing:** Uses `TICKS_PER_SECOND` for consistent timing intervals
- **Diagnostics:** Includes system diagnostics (should be disabled for production)

12.57.2 Web Socket Console

12.57.2.1 WebSocket Console

A NetBurner NNDK application that provides WebSocket-based access to UART/serial ports and stdio console functionality.

12.57.2.1.1 Overview

This application creates WebSocket endpoints that allow web-based clients to communicate with:

- Standard I/O (stdin/stdout/stderr) via WebSocket
- Up to 10 UART/serial ports with configurable parameters

The application acts as a bridge between WebSocket connections and serial communication interfaces, enabling remote serial console access through a web browser.

12.57.2.1.2 Features

- **WebSocket Console Access:** Connect to the device's standard I/O streams via WebSocket
- **UART Bridging:** Access up to 10 UART ports through individual WebSocket connections
- **Configurable Serial Parameters:** Set baud rate, data bits, parity, and stop bits per UART
- **Real-time Data Transfer:** Bidirectional data flow between WebSocket and serial interfaces
- **Connection Management:** Automatic cleanup on connection errors
- **Status Monitoring:** Built-in diagnostics and connection status reporting

12.57.2.1.3 WebSocket Endpoints

12.57.2.1.3.1 Standard I/O Console

- **Endpoint:** `stdio*`
- **Function:** Redirects stdin, stdout, and stderr to the WebSocket connection
- **Usage:** General console access and debugging

12.57.2.1.3.2 UART Access

- **Endpoint:** `uart*`
- **Function:** Provides WebSocket access to individual UART ports
- **Parameters** (via query string):
 - `uart`: UART port number (0-9)
 - `baud`: Baud rate (1-230400)
 - `data_bits`: Data bits (5-8)
 - `parity`: Parity mode (0-2)
 - `stop_bits`: Stop bits (1-3)

12.57.2.1.4 Configuration Parameters

12.57.2.1.4.1 UART Configuration

- **Port Range:** 0-9 (10 total ports supported)
- **Baud Rate Range:** 1 to 230,400 bps
- **Data Bits:** 5, 6, 7, or 8 bits
- **Parity:** 0 (None), 1 (Odd), 2 (Even)
- **Stop Bits:** 1, 2, or 3 bits

12.57.2.1.4.2 System Settings

- **Buffer Management:** Automatic buffer allocation and cleanup
- **Connection Limit:** One WebSocket connection per UART port
- **Error Handling:** Automatic connection cleanup on errors

12.57.2.1.5 Usage Examples

12.57.2.1.5.1 Connecting to Standard I/O

`ws://device-ip/stdio`

12.57.2.1.5.2 Connecting to UART Port

`ws://device-ip/uart?uart=0&baud=115200&data_bits=8&parity=0&stop_bits=1`

This example connects to UART port 0 with:

- 115,200 bps baud rate
- 8 data bits
- No parity
- 1 stop bit

12.57.2.1.6 Application Architecture

12.57.2.1.6.1 Main Components

1. **WebSocket Handlers:** Process incoming WebSocket connections
2. **UART Management:** Handle serial port configuration and I/O
3. **Data Bridging:** Transfer data between WebSocket and serial interfaces
4. **Connection Monitoring:** Track active connections and handle errors

12.57.2.1.6.2 Key Functions

- `StdioWS()`: Handles stdio WebSocket connections
- `UartWS()`: Processes UART WebSocket requests
- `spliceToWS()`: Transfers data from UART to WebSocket
- `spliceToUart()`: Transfers data from WebSocket to UART
- `ParseUartConfig()`: Validates and parses UART configuration parameters

12.57.2.1.7 Error Handling

The application includes robust error handling for:

- Invalid UART configuration parameters
- Port availability conflicts
- WebSocket connection errors
- Serial port communication failures
- Buffer overflow conditions

12.57.2.1.7.1 HTTP Responses

- **400 Bad Request**: Invalid configuration parameters
- **404 Not Found**: Invalid WebSocket upgrade request
- **503 Not Available**: Port already in use or unavailable

12.57.2.1.8 System Requirements

- NetBurner NNDK (NetBurner Real-Time Operating System)
- Network connectivity (Ethernet with [DHCP](#))
- HTTP server support
- WebSocket protocol support
- Available UART hardware ports

12.57.2.1.9 Compilation Dependencies

- [init.h](#) - System initialization
- [nbrtos.h](#) - NetBurner RTOS functions
- [websockets.h](#) - WebSocket protocol support
- [iosys.h](#) - I/O system functions
- [serial.h](#) - Serial port management
- [dhcpcd.h](#) - [DHCP](#) server functionality

12.57.2.1.10 Status Monitoring

The application provides real-time status information including:

- Active WebSocket connections
- UART port assignments
- Buffer usage statistics
- Connection health monitoring

Status updates are printed to the console every 10 seconds, showing:

- Current socket file descriptors
- UART port mappings
- Memory buffer availability

12.57.2.1.11 Troubleshooting

12.57.2.1.11.1 Common Issues

1. **Port Already in Use:** Ensure UART port is not already connected
2. **Invalid Parameters:** Check baud rate and bit configuration ranges
3. **Connection Dropped:** Monitor network stability and WebSocket client implementation
4. **Buffer Overflow:** Monitor system buffer usage in high-throughput scenarios

12.57.2.1.11.2 Diagnostic Features

- Built-in task stack checking (when `NBRTOS_STACKCHECK` is enabled)
- Connection status reporting
- Buffer usage monitoring
- Automatic error detection and cleanup

12.57.3 Web Socket DIP Switch

Supported Platforms: MODM7AE70, MOD54415, MOD54417

12.57.3.1 Real-Time DIP Switch States via WebSocket

12.57.3.1.1 Overview

This NetBurner application demonstrates real-time monitoring of DIP switch states through a web interface using WebSockets. The application continuously polls DIP switches on the development board and instantly updates a webpage whenever switch positions change, without requiring page refreshes.

The application continuously polls the DIP switches and sends the state of the DIP switches via a websocket to the client. It's not the most efficient implementation when considering CPU utilization but this app demonstrates the capabilities of a WebSocket to allow the server(NetBurner device) to send the state of a variable to the client(webpage) with low latency and minimal packet size. This app also illustrates how to use the NetBurner JSON library to build and send JSON objects from the NetBurner device to the client. In this case, JSON objects are used to pass the state of the DIP switches to the webpage.

12.57.3.1.2 Features

- **Real-time Updates:** DIP switch state changes are immediately reflected on the connected webpage
- **WebSocket Communication:** Low-latency, minimal packet size communication between device and client
- **LED Control:** Interactive LED control through the web interface
- **JSON Data Exchange:** Structured data transfer using NetBurner's JSON library
- **Multi-platform Support:** Compatible with multiple NetBurner platforms

12.57.3.1.3 Supported Platforms

- MODM7AE70
- MOD5441x

12.57.3.1.4 Hardware Requirements

- NetBurner development board (one of the supported platforms)
- MOD-DEV-70CR carrier board
- 8 DIP switches (connected to specific pins)
- 8 LEDs (connected to specific pins)

12.57.3.1.5 Pin Configuration

12.57.3.1.5.1 LED Connections (J2/P2 Connector)

- LED 1: Pin 15
- LED 2: Pin 16
- LED 3: Pin 31
- LED 4: Pin 23
- LED 5: Pin 37
- LED 6: Pin 19
- LED 7: Pin 20
- LED 8: Pin 24

12.57.3.1.5.2 DIP Switch Connections (J2/P2 Connector)

- Switch 1: Pin 8 (A/D channel 7)
- Switch 2: Pin 6 (A/D channel 6)
- Switch 3: Pin 7 (A/D channel 5)
- Switch 4: Pin 10 (A/D channel 3)
- Switch 5: Pin 9 (A/D channel 4)
- Switch 6: Pin 11 (A/D channel 1)
- Switch 7: Pin 12 (A/D channel 0)
- Switch 8: Pin 13 (A/D channel 2)

12.57.3.1.6 How It Works

12.57.3.1.6.1 System Architecture

1. **Main Loop:** Continuously polls DIP switch states
2. **WebSocket Server:** Handles client connections and data transmission
3. **Input Task:** Processes incoming WebSocket messages for LED control
4. **JSON Communication:** Structures data exchange between device and web client

12.57.3.1.6.2 Data Flow

1. Application reads DIP switch positions using analog-to-digital conversion
2. Switch states are converted to JSON format
3. JSON data is transmitted via WebSocket to connected web clients
4. Web interface receives LED control commands and updates hardware accordingly

12.57.3.1.6.3 Switch State Detection

- **MCF5441X Platform:** Uses analog-to-digital conversion to read switch positions
- **Other Platforms:** Uses built-in `getdipsw()` function
- **State Logic:** 0 = switch ON, 1 = switch OFF
- **Change Detection:** Only transmits updates when switch states actually change

12.57.3.1.7 Key Components

12.57.3.1.7.1 WebSocket Endpoint

```
CallBackWSEndPoint ws("data", StartWS);
```

Creates a WebSocket endpoint at `/data` path for client connections.

12.57.3.1.7.2 JSON Data Structure

The application sends DIP switch states in the following JSON format:

```
{
  "dipSwitches": {
    "dip1": "On/Off",
    "dip2": "On/Off",
    ...
    "dip8": "On/Off"
  }
}
```

12.57.3.1.7.3 LED Control

LEDs can be controlled via incoming JSON messages with the format:

```
{
  "ledcb1": true/false,
  "ledcb2": true/false,
  ...
}
```

12.57.3.1.8 Configuration Parameters

- `INCOMING_BUF_SIZE`: 8192 bytes for incoming WebSocket data
- `REPORT_BUF_SIZE`: 512 bytes for outgoing JSON reports
- `NUM_LEDS`: 8 LEDs supported
- `NUM_SWITCHES`: 8 DIP switches supported
- `STATE_BUF_SIZE`: 8 bytes per switch state string

12.57.3.1.9 Usage

1. **Build and Deploy:** Compile and flash the application to your NetBurner device
2. **Network Connection:** Ensure the device is connected to your network
3. **Web Interface:** Connect to the device's web interface through a browser
4. **Real-time Monitoring:** Flip DIP switches to see immediate updates on the webpage
5. **LED Control:** Use the web interface to control individual LEDs

12.57.3.1.10 Performance Considerations

While this implementation continuously polls the DIP switches, it's designed for demonstration purposes rather than optimal CPU utilization. The polling approach clearly shows WebSocket capabilities for real-time data transmission with minimal latency.

12.57.3.1.11 Development Notes

- Uses NetBurner RTOS for task management
- Implements proper WebSocket message parsing with JSON object boundary detection
- Handles WebSocket connection lifecycle (connect/disconnect)
- Includes error handling for socket operations
- Uses semaphores for task synchronization

12.57.4 Web Socket Echo

12.57.4.1 WebSocket Echo Application

12.57.4.1.1 Overview

The WebSocket Echo Application is a real-time communication server that implements a WebSocket endpoint capable of echoing messages back to connected clients. Built for embedded systems or microcontroller environments, this application demonstrates basic WebSocket server functionality using a network stack framework.

12.57.4.1.2 Application Details

Application Name:** WebsocketEcho

12.57.4.1.3 Features

- **WebSocket Server:** Implements a WebSocket endpoint at the pattern "echo*"
- **Real-time Echo:** Receives messages from WebSocket clients and echoes them back
- **Network Stack Integration:** Built on top of a custom network initialization framework
- **HTTP Server:** Includes web server functionality on default port 80
- **Connection Management:** Handles WebSocket connection lifecycle with proper cleanup
- **System Diagnostics:** Optional diagnostic capabilities for development

12.57.4.1.4 Architecture

12.57.4.1.4.1 Core Components

1. **WebSocket Endpoint Handler:** `CallBackWSEndPoint wsep("echo*", WSStart)`

- Registers a WebSocket endpoint that matches URLs starting with "echo"
- Calls `WSStart()` callback when new connections are established

2. **Connection Management:**

- Global WebSocket file descriptor (`ws_fd`)
- Semaphore-based synchronization (`waitingForWS`)
- Select-based I/O multiplexing for non-blocking operations

3. **Echo Logic:**

- Reads incoming WebSocket data
- Writes data to stdout (for logging/debugging)
- Echoes the same data back to the WebSocket client
- Flushes WebSocket buffers to ensure immediate transmission

12.57.4.1.4.2 Application Flow

1. **Initialization Phase:**

- Initialize network stack
- Enable system diagnostics (development feature)
- Start HTTP server on port 80
- Wait for active network connection (5-second timeout)
- Initialize file descriptor sets for `select()` operations

2. **Main Loop:**

- Wait for WebSocket connection using semaphore
- Monitor WebSocket file descriptor for read/error events
- Handle incoming data by echoing it back to the client
- Manage connection cleanup on errors or disconnections
- Implement 1-second delay between connection attempts

12.57.4.1.5 Technical Implementation

12.57.4.1.5.1 Key Functions

- `WSStart(int rv)`: Callback function for new WebSocket connections
- `UserMain(void *pd)`: Main application entry point and event loop

12.57.4.1.5.2 System Dependencies

- Custom network initialization framework (`init.h`)
- WebSocket implementation (`websockets.h`)
- POSIX-style socket operations (`select`, `read`, `write`, `close`)
- Real-time operating system primitives (semaphores, timing)

12.57.4.1.5.3 Buffer Management

- Uses `SMPoolPtr` for memory pool management
- Buffer size defined by `ETHER_BUFFER_SIZE`
- Automatic buffer cleanup and reuse

12.57.4.1.6 Usage

12.57.4.1.6.1 Starting the Application

The application automatically starts when deployed to the target system:

1. Network stack initializes
2. HTTP server starts on port 80
3. WebSocket endpoint becomes available at URLs matching "echo*"
4. Application waits for WebSocket connections

12.57.4.1.6.2 Connecting Clients

WebSocket clients can connect to endpoints such as:

- `ws://[device-ip]/echo`
- `ws://[device-ip]/echo-test`
- `ws://[device-ip]/echo-anything`

12.57.4.1.6.3 Message Flow

1. Client sends WebSocket message
2. Server receives and logs message to stdout
3. Server immediately echoes message back to client
4. Client receives the echoed message

12.57.4.1.7 Development Notes

12.57.4.1.7.1 Production Considerations

- System diagnostics should be disabled in production builds
- Consider implementing authentication for WebSocket endpoints
- Add error handling for memory allocation failures
- Implement rate limiting for DoS protection

12.57.4.1.7.2 Debugging Features

- Real-time message logging to stdout
- Connection status messages
- Socket close notifications

12.57.4.1.8 Error Handling

The application includes error handling:

- Automatic connection cleanup on socket errors
- Graceful handling of multiple connection attempts
- Resource management with proper file descriptor cleanup
- Timeout handling for network operations

12.57.5 HTTPS WebSocket Demo

12.57.5.1 SSL/TLS WebSocket Demo

A NetBurner application demonstrating secure and non-secure HTTP/HTTPS web server functionality with WebSocket support and SSL/TLS authentication.

12.57.5.1.1 Overview

This application showcases how to implement both secure (HTTPS) and non-secure (HTTP) access to web resources alongside WebSocket communication. The server provides different access levels based on connection security, demonstrating practical SSL/TLS authentication patterns.

12.57.5.1.2 Features

- **Dual Protocol Support:** Serves content over both HTTP (port 80) and HTTPS (port 443)
- **SSL/TLS WebSocket:** Secure WebSocket communication with authentication
- **Access Control:** Directory-based security with SSL-only restricted areas
- **Dynamic Content:** Server-side functions for generating dynamic links and content
- **Echo WebSocket:** Real-time bidirectional communication demonstration

12.57.5.1.3 Directory Structure

```
html/
  httpsdir/
    ssl.html          # HTTPS-only accessible content
  images/
    SSL-Good.gif      # Shown for secure connections
    SSL-Bad.gif       # Shown for non-secure connections
  index.html         # Main page (HTTP/HTTPS accessible)
  script.js          # Client-side JavaScript
```

12.57.5.1.4 Access Control

The application implements a three-tier access control system:

- **Level 0:** Public access (HTTP/HTTPS)
- **Level 1:** HTTPS-only access
- **Higher Levels:** Custom authentication (extensible)

12.57.5.1.4.1 Security Implementation

```
HTTP_ACCESS CheckHttpAccess(int sock, int access_level, HTTP_Request &Req)
```

This function overrides the system default to enforce SSL/TLS requirements:

- Level 0: Always accessible
- Level 1: Requires SSL connection, returns HTTP_FORBIDDEN for non-SSL
- Extensible for additional security levels

12.57.5.1.5 WebSocket Functionality

12.57.5.1.5.1 Echo WebSocket Endpoint

- **Endpoint:** `/echo*`
- **Functionality:** Echoes received messages back to sender
- **Protocol:** Supports both WS and WSS (secure WebSocket)

12.57.5.1.5.2 WebSocket Lifecycle

1. Connection established via `WSStart()` callback
2. Socket registered and semaphore posted
3. Main loop handles read/write operations
4. Automatic cleanup on connection errors

12.57.5.1.6 Dynamic Content Functions

12.57.5.1.6.1 SSL_Image()

Dynamically serves different images based on connection security:

- SSL connections: `SSL-Good.gif`
- Non-SSL connections: `SSL-Bad.gif`

12.57.5.1.6.2 HTTPS_Ref() / HTTP_Ref()

Generates protocol-specific URLs for navigation:

- Automatically detects host information
- Constructs proper HTTPS/HTTP links
- Supports both hostname and IP address formats

12.57.5.1.6.3 Host_Ref()

Provides the current host identifier (hostname or IP address)

12.57.5.1.7 Configuration Options

12.57.5.1.7.1 Default Configuration

```
StartHttps(); // Starts both HTTP (80) and HTTPS (443)
```

12.57.5.1.7.2 HTTPS-Only Configuration

```
StartHttps(443, 0); // HTTPS only, disables HTTP port
```

12.57.5.1.8 Network Requirements

- **DHCP:** Waits up to 5 seconds for network address assignment
- **Ports:**
 - HTTP: 80 (configurable/disableable)
 - HTTPS: 443 (configurable)
- **SSL/TLS:** Requires valid SSL certificate configuration

12.57.5.1.9 Main Application Flow

1. Initialization

- Network stack initialization
- System diagnostics enablement
- `DHCP` address acquisition

2. Server Startup

- HTTP/HTTPS web server activation
- WebSocket endpoint registration

3. Main Loop

- WebSocket connection monitoring
- Bidirectional data handling
- Error detection and cleanup
- Connection state management

12.57.5.1.10 Development Notes

12.57.5.1.10.1 Memory Management

- Uses `SMPoolPtr` for buffer management
- `ETHER_BUFFER_SIZE` defines maximum message size
- Automatic cleanup on connection errors

12.57.5.1.10.2 Concurrency

- Uses `OS_SEM` semaphore for WebSocket synchronization
- `select()` for non-blocking I/O operations
- Proper file descriptor management

12.57.5.1.10.3 Error Handling

- Connection error detection via `select()`
- Graceful socket closure
- Automatic reconnection support

12.57.5.1.11 Security Considerations

1. **SSL/TLS Configuration:** Ensure proper certificate installation
2. **Access Control:** Directory-based restrictions prevent unauthorized access
3. **Connection Validation:** All SSL connections verified before serving content
4. **Resource Protection:** Sensitive content isolated in `httpsdir/`

12.57.5.1.12 Usage Instructions

1. **Build and Deploy:** Compile with NetBurner development tools
2. **Network Setup:** Ensure device has network connectivity
3. **Access:** Navigate to device IP address via web browser
4. **Test Security:** Compare HTTP vs HTTPS access to different resources
5. **WebSocket:** Use JavaScript client to connect to echo endpoint

12.57.5.1.13 Troubleshooting

- **Connection Issues:** Check network configuration and [DHCP](#)
- **SSL Errors:** Verify certificate installation and validity
- **Access Denied:** Ensure proper file placement in directory structure
- **WebSocket Failures:** Monitor console output for connection status

12.58 WiFi Examples

Wifi Examples:

- [Wifi Config AP](#)
- [Configure AP with JSON](#)
- [Wifi Firmware Update](#)
- [Wifi Access Point](#)
- [Wifi Client](#)
- [Wifi Scan](#)

12.58.1 Wifi Config AP

12.58.1.1 NetBurner WiFi Configuration Access Point

12.58.1.1.1 Overview

This application configures a NetBurner device's WiFi interface as an access point, providing users with a convenient way to connect the device to local wireless networks. When no network credentials are stored, the device automatically starts a WiFi Access Point that hosts a webpage for network configuration.

12.58.1.1.2 Features

- **Automatic AP Mode:** Starts WiFi Access Point when no stored network credentials are found
- **Network Scanning:** Scans and displays available wireless networks
- **Web-based Configuration:** Provides a webpage interface for connecting to local networks
- **Interactive Console:** Command-line interface for manual WiFi management
- **Status Monitoring:** Real-time WiFi connection status and network information
- **Persistent Storage:** Saves WiFi credentials for automatic reconnection

12.58.1.1.3 Default Behavior

12.58.1.1.3.1 Access Point Configuration

- **Default SSID Format:** NB_<last 6 bytes of MAC address>
 - Example: NB_012345
- **Default IP Address:** 192.168.0.1
- **DHCP Range:** Starting from 192.168.0.2

12.58.1.1.3.2 Custom Branding

You can customize the access point SSID prefix by uncommenting and modifying this line in `main.cpp`:

```
extern const NB::SSID_t default_SSID_Prefix = { 4, "FOO_" };
```

This would create access point names like FOO_012345 instead of NB_012345.

12.58.1.1.4 Operation Modes

12.58.1.1.4.1 1. Configuration Mode (No Stored Credentials)

- Device starts as WiFi Access Point
- SSID and IP address printed to serial console
- Web interface accessible for network selection and credential entry
- User can scan for available networks and connect to desired network

12.58.1.1.4.2 2. Client Mode (Stored Credentials Found)

- Device automatically connects to previously configured network
- Normal operation with internet connectivity
- Web server runs on port 80

12.58.1.1.5 Interactive Commands

The application provides a console interface with the following commands:

Command	Function
C	Connect to WiFi network
D	Disconnect from current network
R	Reset stored WiFi SSID credentials
S	Scan for available networks
W	Display WiFi status and configuration
X	Reset WiFi module
Z	System reboot

12.58.1.1.6 Network Information Display

The status command (W) provides comprehensive information including:

12.58.1.1.6.1 Ethernet Interface

- Interface number
- IP address, netmask, gateway, DNS
- MAC address

12.58.1.1.6.2 WiFi Interface

- Interface number and IP configuration
- Connection status
- Current SSID and BSSID
- Signal strength (RSSI)
- Channel information
- Security type (Open, WEP, WPA, WPA2, WPS)
- Cipher type (None, TKIP, AES, Mixed)

12.58.1.1.6.3 Device Information

- Hardware revision
- WiFi firmware revision
- Stored configuration parameters

12.58.1.1.7 Security Support

The application supports various WiFi security protocols:

- **Open**: No security
- **WEP**: Wired Equivalent Privacy
- **WPA**: WiFi Protected Access
- **WPA2**: WiFi Protected Access 2
- **WPS**: WiFi Protected Setup

Cipher support includes:

- **TKIP**: Temporal Key Integrity Protocol
- **AES**: Advanced Encryption Standard
- **Mixed**: Both TKIP and AES

12.58.1.1.8 Technical Details

12.58.1.1.8.1 Initialization Sequence

1. Network stack initialization
2. HTTP server startup (port 80)
3. WiFi interface initialization and scanning
4. Check for stored credentials
5. Start Configuration AP or connect to stored network

12.58.1.1.8.2 Interface Management

- Uses SPI communication with WiFi module
- Automatic interface number assignment
- Dual-interface support (Ethernet + WiFi)

12.58.1.1.8.3 Memory Management

- Automatic scan result deallocation
- Pool-based buffer management for device information

12.58.1.1.9 Configuration Options

Several library defaults can be overridden by uncommenting variables in `main.cpp`:

```
// Custom DHCP lease start address
// const IPAddress configAP_LeaseStart(192, 168, 0, 2);

// Custom access point IP address
// const IPAddress configAP_IPAddr(192, 168, 0, 1);

// Custom SSID prefix for branding
// const NB::SSID_t default_SSID_Prefix = { 4, "FOO_" };
```

12.58.1.1.10 Usage Instructions

1. First Time Setup:

- Power on the device
- Look for the access point SSID printed on serial console
- Connect to the AP using a computer or smartphone
- Navigate to the device's IP address (typically 192.168.0.1)
- Select and configure your local WiFi network

2. Subsequent Boots:

- Device automatically connects to configured network
- Access device via its assigned IP address on local network

3. Manual Configuration:

- Use serial console commands for advanced configuration
- Reset credentials with 'R' command if needed
- Perform network scans with 'S' command

12.58.1.1.11 Development Notes

- Built for NetBurner embedded systems
- Requires NetBurner RTOS and WiFi libraries
- Designed for development kits with dual Ethernet/WiFi interfaces
- System diagnostics enabled (should be disabled for production)

12.58.2 Configure AP with JSON

12.58.2.1 WiFi Configuration Access Point with JSON

12.58.2.1.1 Overview

This NetBurner application configures the WiFi interface as an access point to provide users with a convenient way to connect to local wireless networks. The application creates a web-based configuration interface that allows users to scan for and connect to available WiFi networks.

Note

The difference from the previous ConfigAP example is that the list of networks is generated using JavaScript to parse a JSON object sent by the NetBurner which contains the access point scan results.

For the user's convenience, the device's SSID and WiFi Interface IP address are printed out the serial port. These can be used to access the device's webpage, which allows the user to connect to a local network.

12.58.2.1.2 Key Features

- **Automatic Access Point Creation:** If no network SSID is configured, the device automatically starts as a WiFi Access Point
- **Network Scanning:** Scan for and display available local wireless networks
- **Web-based Configuration:** User-friendly webpage for network selection and connection
- **JSON-based Communication:** JavaScript parsing of JSON objects containing scan results
- **Custom Branding Support:** Configurable SSID prefix for custom product branding
- **Serial Interface:** Command-line interface for manual WiFi operations

12.58.2.1.3 How It Works

12.58.2.1.3.1 Initial Startup

1. On boot, the application checks for a stored network SSID
2. If no SSID is found, it creates a WiFi Access Point
3. The access point SSID follows the format: NB_<last 6 bytes of MAC>
 - Example: NB_012345
4. The device's SSID and IP address are printed to the serial port

12.58.2.1.3.2 Configuration Process

1. Users connect to the NetBurner's access point using a WiFi-enabled device
2. Navigate to the device's webpage using the provided IP address
3. The webpage displays a list of available networks (generated via JavaScript)
4. Users select their desired network and enter credentials
5. The device connects to the selected network and stores the configuration

12.58.2.1.3.3 Custom Branding

To customize the access point SSID prefix, uncomment and modify this line in `main.cpp`:

```
extern const NB::SSID_t default_SSID_Prefix = { 4, "FOO_" };
```

12.58.2.1.4 Serial Commands

The application provides several serial commands for manual WiFi management:

- **C** - Connect to WiFi network
- **D** - Disconnect from current network
- **R** - Reset stored WiFi SSID
- **S** - Perform network scan
- **W** - Display WiFi status and configuration
- **X** - Reset WiFi module
- **Z** - System reboot

12.58.2.1.5 Technical Details

12.58.2.1.5.1 WiFi Security Support

The application supports multiple security protocols:

- Open (no security)
- WEP
- WPA
- WPA2
- WPS

12.58.2.1.5.2 Cipher Support

- None
- TKIP
- AES
- Mixed (TKIP/AES)

12.58.2.1.5.3 Network Information Display

When connected, the application displays:

- SSID (network name)
- BSSID (access point MAC address)
- Signal strength (RSSI)
- Channel
- Security type
- Cipher method
- IP configuration (address, mask, gateway, DNS)

12.58.2.1.6 Configuration Options

The application allows customization of default settings by uncommenting and modifying these variables:

```
// Default DHCP lease start address for AP mode
// extern const IPADDR4 configAP_LeaseStart(192, 168, 0, 2);

// Default IP address for AP mode
// extern const IPADDR4 configAP_IPAddr(192, 168, 0, 1);

// Custom SSID prefix for branding
// extern const NB::SSID_t default_SSID_Prefix = { 4, "FOO_" };
```

12.58.2.1.7 Usage Instructions

1. First Time Setup:

- Power on the NetBurner device
- Connect to the NB_XXXXXX access point
- Open a web browser and navigate to the device's IP address
- Select your WiFi network and enter credentials

2. Subsequent Startups:

- The device will automatically connect to the stored network
- If connection fails, it will revert to access point mode

3. Manual Configuration:

- Use serial commands for direct control
- Monitor connection status and network information

12.58.2.1.8 Development Notes

- The application uses SPI communication for WiFi operations
- Web server runs on default port 80
- System diagnostics are enabled (should be disabled for production)
- Configuration mirroring is enabled for debugging
- The application includes comprehensive error handling and status reporting

12.58.3 Wifi Firmware Update

12.58.3.1 WiFi Firmware Update Application

12.58.3.1.1 Overview

The NBWIFI-IN Firmware Update application is a C++ program designed to update the firmware on NetBurner WiFi modules. This application provides a simple interface to check current firmware information and perform firmware updates on compatible WiFi hardware.

12.58.3.1.2 Features

- **Firmware Information Display:** Shows current hardware and software revision numbers
- **Interactive Firmware Update:** User-prompted firmware update process
- **Network Stack Integration:** Built-in HTTP server and [DHCP](#) support
- **System Diagnostics:** Optional diagnostic capabilities for development
- **Error Handling:** Comprehensive error checking and status reporting

12.58.3.1.3 Application Flow

1. Initialization

- Initializes the network stack
- Enables system diagnostics
- Starts HTTP server on default port 80
- Waits for [DHCP](#) address assignment (5-second timeout)

2. WiFi Module Detection

- Attempts to initialize WiFi interface
- Sets SPI communication speed to 15 MHz
- Validates communication with WiFi module

3. Status Display

- Shows current hardware revision (major.minor)
- Shows current firmware revision (major.minor)

4. Firmware Update Process

- Waits for user input to begin update
- Performs firmware update using embedded firmware image
- Reports success or failure status
- Displays updated firmware information

12.58.3.1.4 Hardware Requirements

- NetBurner development board with WiFi capability
- Compatible WiFi module supporting firmware updates
- Proper SPI connections for WiFi communication

12.58.3.1.5 Key Components

12.58.3.1.5.1 WiFi Interface Management

- Automatic WiFi driver initialization
- SPI speed configuration (15 MHz)
- Module communication validation

12.58.3.1.5.2 Firmware Update Process

- Uses embedded firmware image (`nbWifiImage[]`)
- Validates image length (`nbWifiImageLen`)
- Provides real-time update status feedback

12.58.3.1.5.3 Network Services

- HTTP server for web-based management
- [DHCP](#) client for automatic IP configuration
- Network stack diagnostics

12.58.3.1.6 Error Handling

The application includes robust error handling:

- WiFi module communication failure detection
- Firmware update success/failure reporting
- Graceful degradation when WiFi module is unavailable

12.58.3.1.7 Usage

1. Compile and flash the application to your NetBurner device
2. Connect to the device via serial terminal
3. Wait for network initialization and WiFi module detection
4. Review the current firmware information displayed
5. Press any key when prompted to begin the firmware update
6. Wait for the update process to complete
7. Verify the new firmware information

12.58.3.1.8 Development Notes

- System diagnostics are enabled by default (consider disabling for production)
- The application runs an infinite loop after completion
- HTTP server remains active throughout the application lifecycle
- Serial output provides detailed status information

12.58.3.1.9 Troubleshooting

"Could not talk to Wifi module"**: Check physical connections, power supply, and ensure the WiFi module is properly seated and compatible.

Firmware update failure**: Verify the embedded firmware image is correct for your hardware revision and that the WiFi module supports firmware updates.

12.58.4 Wifi Access Point

12.58.4.1 WiFi Access Point Application

12.58.4.1.1 Overview

This application demonstrates how to configure a NetBurner device to operate as a WiFi Access Point (AP) with an integrated [DHCP](#) server. The application allows other devices to connect directly to the NetBurner module and automatically receive IP addresses.

12.58.4.1.2 Features

- WiFi Access Point mode operation
- Integrated **DHCP** server for automatic IP assignment
- Configurable network settings
- Client connection monitoring
- Serial interface for displaying connected clients

12.58.4.1.3 Default Configuration

The application uses the following default network settings:

- **Access Point SSID:** NetBurnerAP
- **Access Point Password:** password
- **WiFi Interface IP:** 192.168.3.1
- **Subnet Mask:** 255.255.255.0
- **DHCP Start Address:** 192.168.3.2
- **WiFi Channel:** 6 (802.11)

12.58.4.1.4 Network Setup

12.58.4.1.4.1 Recommended Configuration

For proper Access Point operation, configure your network settings as follows:

- WiFi Interface IP: 192.168.0.1 (or your preferred network)
- Subnet Mask: 255.255.255.0
- **DHCP** Start Address: 192.168.0.2

12.58.4.1.4.2 Configuration Methods

1. **IPSetup Tool:** Use the NetBurner IPSetup utility to configure:

- WiFi interface IP address and subnet mask
- SSID and password settings

2. **Code Modification:** Modify the default values in `main.cpp`:

```
IPADDR4 startAddr = AsciiToIp4("192.168.3.2"); // DHCP start address
IPADDR4 defaultWifiIPAddr = AsciiToIp4("192.168.3.1"); // AP IP address
```

12.58.4.1.5 How It Works

1. **Initialization:** The application initializes the system and waits for network activation
2. **WiFi AP Setup:** Configures the WiFi module in Access Point mode using SPI communication
3. **IP Configuration:** Sets up the WiFi interface with IP address and subnet mask
4. **DHCP Server:** Starts a **DHCP** server on the WiFi interface to assign IP addresses to connecting clients
5. **Monitoring:** Provides serial interface to display connected **DHCP** clients

12.58.4.1.6 Function Parameters

The `InitAP_SPI()` function accepts the following parameters:

- `SSID`: Network name (default from config if empty)
- `password`: Network password (8-64 characters, default from config if empty)
- `channel`: 802.11 channel (default: 6)
- `irqNum`: Interrupt number (default: 3)
- `moduleNum`: Module number (default: 1)
- `csNum`: Chip select number
- `connectorNum`: Connector number
- `gpioPinNum`: GPIO pin number
- `resetPinNum`: Reset pin number (default: 42)

12.58.4.1.7 Usage

1. **Compile and Deploy**: Build and flash the application to your NetBurner device
2. **Connect**: Use a WiFi-enabled device to connect to the NetBurnerAP network with password `password`
3. **Access**: The device will automatically receive an IP address via [DHCP](#)
4. **Monitor**: Press any key in the serial terminal to display connected clients
5. **Web Access**: Access the NetBurner web interface at the AP IP address (default: `192.168.3.1`)

12.58.4.1.8 Serial Interface

The application provides a simple serial interface:

- Press any key to display currently connected [DHCP](#) clients
- System diagnostics are enabled for debugging (should be disabled in production)

12.58.4.1.9 Error Handling

The application includes error handling for:

- WiFi initialization failures
- [DHCP](#) server startup failures
- Network configuration issues

If errors occur, the application will display error messages and enter an infinite loop for debugging.

12.58.4.1.10 Production Considerations

- **Security**: Change the default SSID and password before deployment
- **Diagnostics**: Disable `EnableSystemDiagnostics()` in production code
- **Network Planning**: Choose appropriate IP ranges that don't conflict with existing networks
- **Channel Selection**: Select an appropriate WiFi channel based on your environment

12.58.4.1.11 Troubleshooting

- **WiFi fails to initialize:** Check SPI connections and module configuration
- **DHCP server fails:** Verify IP configuration and ensure no conflicts
- **Clients can't connect:** Check SSID/password and WiFi channel settings
- **No IP assignment:** Verify **DHCP** start address is in the same subnet as the AP IP

12.58.5 Wifi Client

12.58.5.1 WiFi Client Application

12.58.5.1.1 Overview

This application demonstrates how to connect to a WiFi network using NetBurner's WiFi driver and access comprehensive status information about the connection. The application provides detailed monitoring capabilities for both Ethernet and WiFi network interfaces.

12.58.5.1.2 Features

- **WiFi Network Connection:** Automatically connects to a configured WiFi network using stored credentials
- **Network Scanning:** Scans and displays available WiFi networks
- **Status Monitoring:** Provides real-time status information including:
 - IP configuration (address, mask, gateway, DNS)
 - MAC addresses
 - Signal strength (RSSI)
 - Security protocols and cipher types
 - Connection status
 - Hardware and firmware versions
- **Interactive Interface:** Serial debug port menu for real-time status updates
- **Multi-Interface Support:** Displays both Ethernet and WiFi interface information

12.58.5.1.3 Hardware Compatibility

This application is designed for NetBurner platforms, with special considerations for:

- **5270 and 5234 processors:** Automatically enables spread spectrum mode to reduce radio interference
- **External Antenna Support:** Optimized for WiFi modules with external antennas

12.58.5.1.4 Security Protocols Supported

The application can identify and display the following security types:

- Open (no security)
- WEP
- WPA
- WPA2
- WPS

12.58.5.1.5 Cipher Support

Supported cipher types include:

- None
- TKIP
- AES
- Mixed (TKIP/AES)

12.58.5.1.6 Configuration

12.58.5.1.6.1 WiFi Credentials

The application uses stored configuration records for WiFi credentials:

- SSID and passphrase are loaded from the system configuration
- Total length of SSID and passphrase must not exceed 68 characters
- Credentials can be configured using IPSetup utility

12.58.5.1.6.2 SPI Interface Configuration

The WiFi module connects via SPI with the following default parameters:

- IRQ Number: 3
- Module Number: 1
- CS Number: Default
- Connector Number: Default
- GPIO Pin Number: Default
- Reset Pin Number: 42

12.58.5.1.7 Usage

1. **Startup:** The application automatically scans for available networks on startup
2. **Connection:** Attempts to connect using stored WiFi credentials
3. **Status Display:** Shows comprehensive network status information
4. **Interactive Mode:** Press any key to refresh and display current status

12.58.5.1.8 Output Information

12.58.5.1.8.1 Ethernet Interface

- IP Address, Subnet Mask, Gateway, DNS
- MAC Address

12.58.5.1.8.2 WiFi Interface

- Interface number and IP configuration
- SSID (Service Set Identifier) - network name
- BSSID (Basic Service Set Identifier) - Access Point MAC address
- Signal strength (RSSI)
- Channel information
- Security and cipher protocols
- Connection status
- Hardware and firmware revision information

12.58.5.1.9 Code Structure

12.58.5.1.9.1 Main Components

- **DisplaySecurity()**: Converts security protocol values to human-readable labels
- **DisplayCipher()**: Converts cipher type values to human-readable labels
- **DisplayWifiStatus()**: Comprehensive status information display
- **DisplayWifiConfigRecordParameters()**: Shows configured SSID and passphrase
- **UserMain()**: Primary application entry point and main loop

12.58.5.1.10 Troubleshooting

- If WiFi interface fails to initialize, check SPI connections and module power
- Ensure WiFi credentials are properly configured in system config records
- Verify antenna connections for optimal signal strength
- Check that the WiFi module is compatible with your NetBurner platform

12.58.6 Wifi Scan

12.58.6.1 WiFi Scan Application

12.58.6.1.1 Overview

This is a NetBurner WiFi scanning and management application that provides an interactive command-line interface for WiFi operations on embedded systems. The application demonstrates how to use the NetBurner WiFi driver to scan for access points, connect/disconnect from networks, and monitor connection status.

12.58.6.1.2 Features

- **WiFi Network Scanning**: Scan for available WiFi access points and display their details
- **Connection Management**: Connect to and disconnect from WiFi networks
- **Network Status Monitoring**: Display detailed WiFi and Ethernet interface information
- **Security Information**: Show security protocols (Open, WEP, WPA, WPA2, WPS) and cipher types
- **Interactive Menu System**: Serial command interface for real-time WiFi management
- **System Diagnostics**: Built-in diagnostic capabilities and system reset functions

12.58.6.1.3 Application Structure

12.58.6.1.3.1 Core Components

- **main.cpp**: Primary application file containing the WiFi management logic
- **ReadMe.txt**: Brief description indicating this is a WiFi scanning example

12.58.6.1.3.2 Key Functions

Network Scanning

- `DoSyncScan()`: Performs synchronous WiFi network scan and displays results
- `WifiInitScanAndShow_SPI()`: Initialize WiFi interface and perform initial scan

Status Display

- `DisplayWifiStatus()`: Shows comprehensive WiFi connection and interface information
- `DisplaySecurity()`: Converts security type codes to readable labels
- `DisplayCipher()`: Converts cipher type codes to readable labels
- `DisplayWifiConfigRecordParameters()`: Shows configured SSID and passphrase

Connection Management

- Connect/disconnect functionality through the NetBurner WiFi driver
- Interface number tracking and object management

12.58.6.1.4 Command Interface

The application provides an interactive serial menu with the following commands:

Command	Function
C	Connect to WiFi network
D	Disconnect from current WiFi network
S	Perform WiFi network scan
W	Display WiFi status and configuration
X	Reset WiFi module
Z	System reboot

12.58.6.1.5 Technical Details

12.58.6.1.5.1 Dependencies

- NetBurner HAL (Hardware Abstraction Layer)
- NetBurner WiFi driver libraries
- DSPI (Serial Peripheral Interface) support
- Standard C/C++ libraries

12.58.6.1.5.2 WiFi Security Support

- Open networks
- WEP encryption
- WPA/WPA2 security protocols
- WPS (WiFi Protected Setup)
- TKIP and AES cipher support

12.58.6.1.5.3 Network Information Displayed

- **Interface Details:** IP address, subnet mask, gateway, DNS servers
- **MAC Addresses:** Both Ethernet and WiFi interface MAC addresses
- **WiFi Specific:** SSID, BSSID, signal strength (RSSI), channel, security type
- **Device Information:** Hardware and firmware revision numbers

12.58.6.1.6 Usage

1. **Initialization:** The application automatically initializes the network stack and performs an initial WiFi scan on startup
2. **Web Server:** Starts an HTTP server on port 80 for additional functionality
3. **Interactive Control:** Use the serial interface to send commands for WiFi management
4. **Network Waiting:** Waits up to 5 seconds for [DHCP](#) address assignment

12.58.6.1.7 Configuration

The application reads WiFi configuration from stored settings including:

- Target SSID (network name)
- Passphrase/password
- Security settings

12.58.6.1.8 Error Handling

- Interface initialization validation
- Connection status verification
- Retry limit monitoring
- Graceful failure reporting

12.58.6.1.9 Notes

- The application is designed for development and testing purposes
- System diagnostics are enabled (should be disabled for production)
- Supports both Ethernet and WiFi interfaces simultaneously
- Uses C++ with NetBurner-specific namespaces and libraries

Chapter 13

Deprecated List

Member [flexcan_clock_source_t](#)

Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

Member [flexcan_config_t](#)

Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

Module [groupFtpServerCallback](#)

For NetBurner customers porting from the 2.x tools to 3.x: FTP_FAIL and FTP_OK were deprecated on Jan 16, 2008. These defines were removed and all server functions (denoted as FTPD) were modified to use FTPD_OK and FTPD_FAIL. This was necessary so that the FTP client and server header files could be used at the same time. If you have any errors in your application code, you can safely change all server functions from FTP_OK/FAIL to FTPD_OK/FAIL.

Chapter 14

NetBurner API

14.1 AES - Advanced Encryption Standard

Classes

- struct `aes_context`
AES context structure.

Functions

- void `aes_set_key` (`aes_context` *ctx, const unsigned char *key, int keysize)
AES key schedule.
- void `aes_encrypt` (`aes_context` *ctx, unsigned char input[16], unsigned char output[16])
AES block encryption (ECB mode)
- void `aes_decrypt` (`aes_context` *ctx, unsigned char input[16], unsigned char output[16])
AES block decryption (ECB mode)
- void `aes_cbc_encrypt` (`aes_context` *ctx, unsigned char iv[16], unsigned char *input, unsigned char *output, int len)
AES-CBC buffer encryption.
- void `aes_cbc_decrypt` (`aes_context` *ctx, unsigned char iv[16], unsigned char *input, unsigned char *output, int len)
AES-CBC buffer decryption.
- int `aes_self_test` (int verbose)
Checkup routine.

14.1.1 Detailed Description

```
#include< aes.h>
```

Function to implement the symmetric block cyper AES

14.1.2 Function Documentation

14.1.2.1 `aes_cbc_decrypt()`

```
void aes_cbc_decrypt (
    aes_context * ctx,
    unsigned char iv[16],
    unsigned char * input,
    unsigned char * output,
    int len)
#include <aes.h>
```

AES-CBC buffer decryption.

Parameters

<i>ctx</i>	AES context
<i>iv</i>	initialization vector (modified after use)
<i>input</i>	buffer holding the ciphertext
<i>output</i>	buffer holding the plaintext
<i>len</i>	length of the data to be decrypted

14.1.2.2 aes_cbc_encrypt()

```
void aes_cbc_encrypt (  
    aes_context * ctx,  
    unsigned char iv[16],  
    unsigned char * input,  
    unsigned char * output,  
    int len)
```

```
#include <aes.h>
```

AES-CBC buffer encryption.

Parameters

<i>ctx</i>	AES context
<i>iv</i>	initialization vector (modified after use)
<i>input</i>	buffer holding the plaintext
<i>output</i>	buffer holding the ciphertext
<i>len</i>	length of the data to be encrypted

14.1.2.3 aes_decrypt()

```
void aes_decrypt (  
    aes_context * ctx,  
    unsigned char input[16],  
    unsigned char output[16])
```

```
#include <aes.h>
```

AES block decryption (ECB mode)

Parameters

<i>ctx</i>	AES context
<i>input</i>	ciphertext block
<i>output</i>	plaintext block

14.1.2.4 aes_encrypt()

```
void aes_encrypt (  
    aes_context * ctx,  
    unsigned char input[16],  
    unsigned char output[16])
```

```
#include <aes.h>
```

AES block encryption (ECB mode)

Parameters

<i>ctx</i>	AES context

Parameters

<i>input</i>	plaintext block
<i>output</i>	ciphertext block

14.1.2.5 aes_self_test()

```
int aes_self_test (
    int verbose)
#include <aes.h>
Checkup routine.
```

Returns

0 if successful, or 1 if the test failed

14.1.2.6 aes_set_key()

```
void aes_set_key (
    aes_context * ctx,
    const unsigned char * key,
    int keysize)
#include <aes.h>
AES key schedule.
```

Parameters

<i>ctx</i>	AES context to be initialized
<i>key</i>	the secret key
<i>keysizes</i>	must be 128, 192 or 256

14.2 ARP - Address Resolution Protocol

Address Resolution Protocol implementation and cache management.

Classes

- struct [ARP](#)
Address Resolution Protocol packet structure.

Typedefs

- typedef [ARP](#) * [PARP](#)
Pointer to [ARP](#) packet structure.

Functions

- void [ShowArp](#) ()
Display the current [ARP](#) cache contents to the standard output.
- void [fShowArp](#) (FILE *fp)
Display the current [ARP](#) cache contents to a specified file stream.
- BOOL [GetArpMacFromIp](#) ([IPADDR4](#) ip, [MACADR](#) &ma)
Check if an IP address is in the [ARP](#) cache and retrieve its MAC address.
- void [sendGratuitousArp](#) (int interfaceNumber, [IPADDR4](#) ip)

- Send a Gratuitous [ARP](#) to announce interface presence and IP address.*

 - **BOOL [IsMyAddressUsedArpDetect](#)** (uint16_t timeout, int interface=0)

Detect if the interface's IP address is already in use on the network.
- **int [GetProperInterface4](#)** (IPADDR4 dst)

Get the appropriate network interface for routing to a destination IP.
- **void [processArp](#)** (PoolPtr p, PEFRAME pf)

Process incoming [ARP](#) packets (internal use only)
- **void [AddCheckArp](#)** (MACADR *ma, IPADDR4 ip, int ifnum)

Add or update [ARP](#) cache entry (internal use only)
- **void [DeleteArpFromIp](#)** (IPADDR4 ip)

Delete [ARP](#) cache entry by IP address (internal use only)
- **void [RouteOut](#)** (PEFRAME pf, IPADDR4 ip, PoolPtr p)

Route packet to destination (internal use only)
- **void [RouteOutVia](#)** (PEFRAME pf, IPADDR4 ipfrom, IPADDR4 ipto, PoolPtr p, int intf)

Route packet via specific interface (internal use only)
- **void [AgeArp](#)** ()

Age out old [ARP](#) cache entries (internal use only)
- **void [AddStaticArp](#)** (MACADR *ma, IPADDR4 ip, int ifnum)

Add static [ARP](#) entry (internal use only)
- **void [InitializeArp](#)** ()

Initialize [ARP](#) subsystem (internal use only)

Variables

- **beuint16_t [ARP::hard_Type](#)**
- Hardware type (1 for Ethernet)*
- **beuint16_t [ARP::prot_Type](#)**
- Protocol type (0x0800 for IPv4)*
- **uint8_t [ARP::hard_size](#)**
- Hardware address length (6 for MAC)*
- **uint8_t [ARP::prot_size](#)**
- Protocol address length (4 for IPv4)*
- **beuint16_t [ARP::op_code](#)**
- Operation: 1=Request, 2=Reply.*
- **MACADR [ARP::sender_phy](#)**
- Sender MAC address.*
- **IPADDR4 [ARP::sender_ip](#)**
- Sender IP address.*
- **MACADR [ARP::target_phy](#)**
- Target MAC address.*
- **IPADDR4 [ARP::target_ip](#)**
- Target IP address.*

14.2.1 Detailed Description

Address Resolution Protocol implementation and cache management.

```
#include< arp.h>
```

The Address Resolution Protocol ([ARP](#)) is a fundamental network protocol that maps Internet Protocol (IP) addresses to physical Media Access Control (MAC) addresses on local Ethernet networks. [ARP](#) enables devices to discover the hardware address of another device on the same network segment when only the IP address is known.

14.2.2 How ARP Works

When a device needs to communicate with another device on the local network:

1. **Cache Check:** First checks local [ARP](#) cache for existing IP-to-MAC mapping
2. **ARP Request:** If not found, broadcasts an [ARP](#) request to all devices
3. **ARP Reply:** Target device responds with its MAC address
4. **Cache Update:** Mapping is stored in local [ARP](#) cache for future use
5. **Communication:** Original packet is sent using the resolved MAC address

14.2.3 ARP Cache Management

NetBurner maintains an [ARP](#) cache that stores recently resolved IP-to-MAC address mappings. This cache:

- Reduces network traffic by avoiding repeated [ARP](#) requests
- Ages out entries after a timeout period
- Can be queried programmatically
- Supports static entries for critical devices

14.2.4 Function Categories

Function	Purpose	Use Case
ShowArp()	Display cache	Debugging, diagnostics
GetArpMacFromIp()	Query cache	Check if address is resolved
sendGratuitousArp()	Announce presence	After IP change or boot
IsMyAddressUsedArpDetect()	Detect conflicts	IP conflict detection

14.2.5 Example Usage

14.2.5.1 Querying the ARP Cache

```
#include <arp.h>

void checkDevicePresence(IPADDR4 deviceIP) {
    MACADDR mac;

    if (GetArpMacFromIp(deviceIP, mac)) {
        iprintf("Device found: ");
        iprintf("%02X:%02X:%02X:%02X:%02X:%02X\n",
            mac.ma[0], mac.ma[1], mac.ma[2],
            mac.ma[3], mac.ma[4], mac.ma[5]);
    } else {
        iprintf("Device not in ARP cache\n");
        // Optionally send a ping to trigger ARP resolution
    }
}
```

14.2.5.2 Displaying ARP Cache Contents

```
#include <arp.h>

void displayNetworkStatus() {
    iprintf("=== Current ARP Cache ===\n");
    ShowArp(); // Outputs to stdio serial port
    iprintf("=====\n");
}

void logArpToFile() {
    FILE *fp = fopen("arp.log", "a");
    if (fp != NULL) {
        fShowArp(fp); // Write to file
        fclose(fp);
    }
}
```


14.2.5.3 Sending Gratuitous ARP

```
#include <arp.h>
#include <netinterface.h>

void announceNewIPAddress(int interface, IPADDR4 newIP) {
    // Update interface IP address
    InterfaceBlock *ib = GetInterfaceBlock(interface);
    ib->ip4.cur_addr = newIP;

    // Announce new address to network
    sendGratuitousArp(interface, newIP);

    iprintf("Gratuitous ARP sent for new IP\n");
}

void UserMain(void *pd) {
    init();

    // Send gratuitous ARP after initialization
    IPADDR4 myIP = AsciiToIp4("192.168.1.100");
    sendGratuitousArp(GetFirstInterface(), myIP);

    // Rest of application...
}
```

14.2.5.4 IP Address Conflict Detection

```
#include <arp.h>

void checkIPConflict(int interface) {
    // Check if our IP is already in use (2 second timeout)
    if (IsMyAddressUsedArpDetect(2, interface)) {
        iprintf("ERROR: IP address conflict detected!\n");
        iprintf("Another device is using this IP address.\n");
        // Handle conflict (change IP, alert user, etc.)
    } else {
        iprintf("No IP conflict detected - address is available\n");
    }
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Check for IP conflicts after network initialization
    checkIPConflict(GetFirstInterface());

    // Continue with application...
}
```

14.2.5.5 Network Device Monitoring

```
#include <arp.h>
#include <icmp.h>

bool isDeviceReachable(IPADDR4 deviceIP) {
    MACADDR mac;

    // First check ARP cache
    if (GetArpMacFromIp(deviceIP, mac)) {
        return true; // Recently communicated with device
    }

    // Not in cache - send ping to trigger ARP
    Ping4(deviceIP, 1, 1); // 1 packet, 1 second timeout
    OSTimeDly(TICKS_PER_SECOND / 2); // Wait for ARP response

    // Check cache again
    return GetArpMacFromIp(deviceIP, mac);
}

void monitorCriticalDevices() {
    IPADDR4 plc = AsciiToIp4("192.168.1.50");
    IPADDR4 gateway = AsciiToIp4("192.168.1.1");

    if (!isDeviceReachable(plc)) {
        iprintf("WARNING: PLC not reachable!\n");
    }

    if (!isDeviceReachable(gateway)) {
        iprintf("WARNING: Gateway not reachable!\n");
    }
}
```

14.2.6 Best Practices

1. **Cache Checking:** Always check [GetArpMacFromIp\(\)](#) return value
2. **Gratuitous ARP:** Send after IP changes to update network caches
3. **Conflict Detection:** Use [IsMyAddressUsedArpDetect\(\)](#) during startup
4. **Triggering Resolution:** Send ping to force [ARP](#) resolution if needed
5. **Cache Monitoring:** Use [ShowArp\(\)](#) for debugging network issues

14.2.7 Troubleshooting

Device Not in Cache:

- Device may be offline or unreachable
- No recent communication with device
- Send ping or UDP packet to trigger [ARP](#) resolution

IP Address Conflict:

- Use [IsMyAddressUsedArpDetect\(\)](#) to verify
- Check for [DHCP](#) server issues
- Verify static IP configurations

Stale Cache Entries:

- Entries age out automatically
- Power cycle or reset may be needed for recovery
- Use gratuitous [ARP](#) to update remote caches

14.2.8 Important Notes

Note

[GetArpMacFromIp\(\)](#) only checks the cache - it does not send [ARP](#) requests. To force [ARP](#) resolution, send a ping or packet to the target first.

Gratuitous [ARP](#) should be sent whenever an interface IP address changes or during system initialization to ensure network-wide cache consistency.

[ARP](#) only works on the local network segment. Devices on different subnets require routing through a gateway.

Warning

IP address conflicts can cause serious network communication issues. Always use [IsMyAddressUsedArpDetect\(\)](#) when configuring static IPs.

The [ARP](#) cache has limited size. Entries are aged out over time to make room for new entries.

14.2.9 Protocol Specifications

- **RFC 826:** Address Resolution Protocol ([ARP](#)) specification
- **RFC 5227:** IPv4 Address Conflict Detection (ACD)
- **RFC 3927:** Dynamic Configuration of IPv4 Link-Local Addresses

See also

[groupICMP](#) For ICMP/Ping functionality

[groupNetInterface](#) For network interface management

[groupIPv4](#) For IPv4 address handling

14.2.10 Typedef Documentation

14.2.10.1 PARP

```
typedef ARP* PARP
```

```
#include <arp.h>
```

Pointer to [ARP](#) packet structure.

Convenience typedef for pointer to [ARP](#) structure. Used throughout the NetBurner stack for packet processing.

14.2.11 Function Documentation

14.2.11.1 fShowArp()

```
void fShowArp (
```

```
    FILE * fp)
```

```
#include <arp.h>
```

Display the current [ARP](#) cache contents to a specified file stream.

Similar to [ShowArp\(\)](#) but outputs to a specified FILE pointer instead of the standard output. This allows [ARP](#) cache contents to be written to:

- Log files on SD card or file system
- Network sockets
- Serial ports via fdopen()
- Any other FILE stream destination

Parameters

<i>fp</i>	Pointer to FILE stream for output. Must be a valid, open FILE pointer (from fopen(), fdopen(), etc.)

Precondition

fp must be a valid, open FILE pointer with write access

Postcondition

[ARP](#) cache contents are written to the specified FILE stream

Note

The FILE pointer must remain valid for the duration of the call

Does not close the FILE pointer - caller is responsible for fclose()

Output format is identical to [ShowArp\(\)](#)

Example: Logging to SD Card

```
#include <arp.h>

void logArpCache() {
    FILE *fp = fopen("SD:/arp_cache.log", "a");
    if (fp != NULL) {
        fprintf(fp, "\n=== ARP Cache at %lu seconds ===\n", Secs);
        fShowArp(fp);
        fprintf(fp, "=====\n");
        fclose(fp);
    } else {
        iprintf("ERROR: Could not open log file\n");
    }
}
```

Example: Sending via Network Socket

```

void sendArpCacheToClient(int clientFd) {
    FILE *fp = fdopen(clientFd, "w");
    if (fp != NULL) {
        fprintf(fp, "Current ARP Cache:\r\n");
        fprintf(fp, "=====\r\n");
        fShowArp(fp);
        fflush(fp);
        // Note: Don't fclose(fp) as it would close the socket
    }
}

```

Example: Multiple Output Destinations

```

void reportArpCache() {
    // Output to serial
    ShowArp();

    // Also log to file
    FILE *fp = fopen("SD:/network.log", "a");
    if (fp) {
        fShowArp(fp);
        fclose(fp);
    }

    // Also send to remote server
    int serverFd = connect4(serverIP, serverPort, 0);
    if (serverFd > 0) {
        FILE *netfp = fdopen(serverFd, "w");
        fShowArp(netfp);
        fflush(netfp);
        fclose(serverFd);
    }
}

```

Example: Custom Formatting Wrapper

```

void formattedArpReport(FILE *fp) {
    time_t now = time(NULL);
    fprintf(fp, "\n");
    fprintf(fp, "┌ ARP CACHE REPORT ────────────┐\n");
    fprintf(fp, "│ %s, ctime(&now); │\n");
    fprintf(fp, "└───────────────────────────┘\n");
    fShowArp(fp);
    fprintf(fp, "\n");
}

```

See also

[ShowArp\(\)](#) For output to standard stdio

[GetArpMacFromIp\(\)](#) To query specific [ARP](#) entries

14.2.11.2 GetArpMacFromIp()

```

BOOL GetArpMacFromIp (
    IPADDR4 ip,
    MACADDR & ma)
#include <arp.h>

```

Check if an IP address is in the [ARP](#) cache and retrieve its MAC address.

Queries the local [ARP](#) cache to determine if a mapping exists for the specified IP address. If found, the corresponding MAC address is returned via the reference parameter.

This function ONLY checks the cache - it does not send [ARP](#) requests over the network. If the entry is not in the cache, you may need to send a packet (such as a ping) to the target IP to trigger [ARP](#) resolution first.

Parameters

	<i>ip</i>	IP address to look up in the ARP cache
out	<i>ma</i>	Reference to MACADDR structure that will receive the MAC address if found. Only modified if the function returns TRUE.

Returns

TRUE if the IP address was found in the cache and MAC address retrieved

FALSE if the IP address is not in the cache

Precondition

ip must be a valid IPv4 address

Postcondition

If TRUE is returned, ma contains the resolved MAC address

If FALSE is returned, ma is unmodified

Note

This function does NOT send [ARP](#) requests - it only checks the cache

To force [ARP](#) resolution, send a ping or packet to the target first

Cache entries age out over time - a FALSE result doesn't mean the device is offline, just that it's not in the current cache

Example: Basic Cache Query

```
#include <arp.h>

void checkDevice(IPADDR4 deviceIP) {
    MACADR mac;

    if (GetArpMacFromIp(deviceIP, mac)) {
        iprintf("Device MAC: %02X:%02X:%02X:%02X:%02X\n",
            mac.ma[0], mac.ma[1], mac.ma[2],
            mac.ma[3], mac.ma[4], mac.ma[5]);
    } else {
        iprintf("Device not in ARP cache\n");
    }
}
```

Example: Forcing ARP Resolution

```
#include <arp.h>
#include <icmp.h>

bool getDeviceMAC(IPADDR4 ip, MACADR &mac) {
    // First check if already in cache
    if (GetArpMacFromIp(ip, mac)) {
        return true;
    }

    // Not in cache - send ping to trigger ARP
    iprintf("Resolving MAC for ");
    ShowIP(ip);
    iprintf("...\n");

    Ping4(ip, 1, 2); // 1 ping, 2 second timeout

    // Wait briefly for ARP resolution
    OSTimeDly(TICKS_PER_SECOND / 2);

    // Check cache again
    if (GetArpMacFromIp(ip, mac)) {
        iprintf("MAC resolved successfully\n");
        return true;
    }

    iprintf("Failed to resolve MAC - device may be offline\n");
    return false;
}
```

Example: Device Presence Detection

```

bool isDevicePresent(IPADDR4 ip) {
    MACADDR mac;
    return GetArpMacFromIp(ip, mac); // True if recently active
}

void monitorDevices() {
    IPADDR4 plc = AsciiToIp4("192.168.1.50");
    IPADDR4 hmi = AsciiToIp4("192.168.1.51");

    while (1) {
        if (!isDevicePresent(plc)) {
            iprintf("WARNING: PLC not responding\n");
        }
        if (!isDevicePresent(hmi)) {
            iprintf("WARNING: HMI not responding\n");
        }
        OSTimeDly(TICKS_PER_SECOND * 10);
    }
}

```

Example: MAC Address Filtering

```

bool isAuthorizedMAC(const MACADDR &mac) {
    // List of authorized MAC addresses
    static const MACADDR authorized[] = {
        {{0x00, 0x11, 0x22, 0x33, 0x44, 0x55}},
        {{0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF}}
    };

    for (size_t i = 0; i < sizeof(authorized)/sizeof(authorized[0]); i++) {
        if (memcmp(&mac, &authorized[i], sizeof(MACADDR)) == 0) {
            return true;
        }
    }
    return false;
}

bool allowConnection(IPADDR4 clientIP) {
    MACADDR mac;

    if (GetArpMacFromIp(clientIP, mac)) {
        return isAuthorizedMAC(mac);
    }

    return false; // Not in cache - deny
}

```

See also

[ShowArp\(\)](#) To display all cached entries

[sendGratuitousArp\(\)](#) To announce device presence

[Ping4\(\)](#) To trigger [ARP](#) resolution

14.2.11.3 GetProperInterface4()

```

int GetProperInterface4 (
    IPADDR4 dst)
#include <arp.h>

```

Get the appropriate network interface for routing to a destination IP.

Determines which network interface should be used to reach a specific IPv4 destination address based on routing table and interface configuration.

Parameters

<i>dst</i>	Destination IPv4 address to route to

Returns

Interface number to use for routing to *dst*, or -1 if no route available

Note

This is primarily for internal stack use
Consider using [GetInterfaceBlock\(\)](#) for interface queries

14.2.11.4 IsMyAddressUsedArpDetect()

```
BOOL IsMyAddressUsedArpDetect (
    uint16_t timeout,
    int interface = 0)
#include <arp.h>
```

Detect if the interface's IP address is already in use on the network.

Performs ARP-based IP address conflict detection by sending [ARP](#) probes and listening for replies. This is used to detect if another device on the network is already using the interface's configured IP address.

This function is critical for:

- Validating static IP configurations
- Detecting duplicate IPs during startup
- Implementing IPv4 Link-Local addressing (169.254.x.x)
- Preventing IP conflicts in dynamic reconfiguration

Parameters

<i>timeout</i>	Maximum time to wait for replies, in seconds. Typical values are 1-5 seconds. Longer timeouts provide more reliable detection but delay startup.
<i>interface</i>	Network interface number to check (default = 0 for primary interface). Use GetFirstInterface() for the default interface.

Returns

TRUE if another device is already using this IP address (conflict detected)
FALSE if the IP address is available (no conflict)

Precondition

The interface must be initialized with a valid IP address
The network interface must be physically active (link up)

Postcondition

Function blocks for up to timeout seconds while probing

Note

This function blocks for the specified timeout period
Multiple [ARP](#) probes may be sent during the detection period
FALSE does not guarantee the IP is permanently available - conflicts can occur after this check completes

Warning

This function can take several seconds to complete. Do not call from time-critical code or ISRs.
A return value of FALSE does not prevent future conflicts. Another device may start using the IP after this check.

Example: Startup IP Conflict Check

```

#include <arp.h>

void UserMain(void *pd) {
    init();

    // Wait for network link
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Check for IP conflict (2 second timeout)
    iprintf("Checking for IP address conflicts...\n");

    if (IsMyAddressUsedArpDetect(2, GetFirstInterface())) {
        iprintf("ERROR: IP address conflict detected!\n");
        iprintf("Another device is using this IP address.\n");
        iprintf("Please reconfigure the IP address.\n");

        // Optionally change to different IP or halt
        while (1) {
            OSTimeDly(TICKS_PER_SECOND);
        }
    } else {
        iprintf("No IP conflict - address is available\n");
    }

    // Continue with application...
}

```

Example: Auto-Recovery from Conflict

```

bool findAvailableIP(int interface, IPADDR4 baseIP) {
    // Try up to 10 different IP addresses
    for (int offset = 0; offset < 10; offset++) {
        IPADDR4 testIP = baseIP + offset;

        // Temporarily configure this IP
        InterfaceBlock *ib = GetInterfaceBlock(interface);
        IPADDR4 oldIP = ib->ip4.cur_addr;
        ib->ip4.cur_addr = testIP;

        iprintf("Testing ");
        ShowIP(testIP);
        iprintf("...\n");

        // Check if available (1 second timeout for speed)
        if (!IsMyAddressUsedArpDetect(1, interface)) {
            iprintf("Available IP found: ");
            ShowIP(testIP);
            iprintf("\n");
            sendGratuitousArp(interface, testIP);
            return true;
        }

        // Restore old IP for next test
        ib->ip4.cur_addr = oldIP;
    }

    return false; // No available IP found
}

```

Example: Link-Local IP Assignment

```

#include <arp.h>
#include <stdlib.h>

IPADDR4 assignLinkLocalIP(int interface) {
    // Link-local range: 169.254.1.0 - 169.254.254.255
    IPADDR4 baseIP = AsciiToIp4("169.254.1.0");

    // Generate random host portion
    srand(Secs);

    for (int attempts = 0; attempts < 100; attempts++) {
        // Random address in link-local range
        int hostBits = (rand() % 254) + 1; // 1-254
        IPADDR4 testIP = baseIP + hostBits;

        // Configure interface
        InterfaceBlock *ib = GetInterfaceBlock(interface);
        ib->ip4.cur_addr = testIP;
        ib->ip4.cur_mask = AsciiToIp4("255.255.0.0");

        // Test for conflicts (2 second timeout)
        if (!IsMyAddressUsedArpDetect(2, interface)) {

```



```

        // No conflict - announce and return
        sendGratuitousArp(interface, testIP);
        return testIP;
    }

    // Conflict detected - try another address
    iprintf("Conflict on ");
    ShowIP(testIP);
    iprintf(" - trying another...\n");
}

return 0; // Failed to find available address
}

```

Example: Configuration Validation

```

bool validateStaticIPConfig(const char *ipStr) {
    IPADDR4 newIP = AsciiToIp4(ipStr);
    int iface = GetFirstInterface();

    // Temporarily set the new IP
    InterfaceBlock *ib = GetInterfaceBlock(iface);
    IPADDR4 oldIP = ib->ip4.cur_addr;
    ib->ip4.cur_addr = newIP;

    iprintf("Validating IP configuration: %s\n", ipStr);

    // Check for conflicts
    bool conflict = IsMyAddressUsedArpDetect(3, iface);

    if (conflict) {
        iprintf("ERROR: IP %s is already in use!\n", ipStr);
        // Restore old IP
        ib->ip4.cur_addr = oldIP;
        return false;
    } else {
        iprintf("IP %s is available\n", ipStr);
        // Keep new IP and announce it
        sendGratuitousArp(iface, newIP);
        return true;
    }
}

```

See also

[sendGratuitousArp\(\)](#) To announce IP after successful check

[GetArpMacFromIp\(\)](#) To identify conflicting device's MAC

[InterfaceIP4\(\)](#) To get interface IP address

14.2.11.5 sendGratuitousArp()

```

void sendGratuitousArp (
    int interfaceNumber,
    IPADDR4 ip)
#include <arp.h>

```

Send a Gratuitous [ARP](#) to announce interface presence and IP address.

Sends a gratuitous [ARP](#) request/reply to inform all devices on the local network about this interface's IP-to-MAC address mapping. This updates the [ARP](#) caches of all devices on the network segment.

Gratuitous [ARP](#) should be sent whenever:

- The system boots up or an interface is initialized
- An IP address is changed or reconfigured
- Taking over an IP from another device (failover)
- After detecting and resolving an IP address conflict

Parameters

<i>interfaceNumber</i>	Network interface number (0 for primary interface, use GetFirstInterface() for the default interface)
<i>ip</i>	IP address to announce. Should be the interface's currently configured IP address. Must be a valid, non-zero IP address.

Precondition

The interface must be initialized and have a valid IP address
 interfaceNumber must be a valid interface number

Postcondition

Gratuitous [ARP](#) is broadcast on the specified interface
 Other devices on the network will update their [ARP](#) caches

Note

This is a broadcast - all devices on the network segment will receive and process the announcement
 It may take a few milliseconds for other devices to update their caches after receiving the gratuitous [ARP](#)

Warning

Do not send gratuitous [ARP](#) before the interface has a valid IP address - this will send an invalid announcement
 Excessive gratuitous [ARP](#) sending can cause unnecessary network traffic. Only send when needed, not continuously.

Example: Startup Announcement

```
#include <arp.h>
#include <netinterface.h>

void UserMain(void *pd) {
    init();

    // Wait for network to be active
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Announce our presence to the network
    int iface = GetFirstInterface();
    IPADDR4 myIP = InterfaceIP4(iface);
    sendGratuitousArp(iface, myIP);

    iprintf("Sent gratuitous ARP for ");
    ShowIP(myIP);
    iprintf("\n");

    // Continue with application...
}
```

Example: IP Address Change

```
#include <arp.h>
#include <netinterface.h>

void changeIPAddress(IPADDR4 newIP) {
    int iface = GetFirstInterface();

    // Update the interface IP
    InterfaceBlock *ib = GetInterfaceBlock(iface);
    ib->ip4.cur_addr = newIP;
    ib->ip4.cur_mask = AsciiToIp4("255.255.255.0");

    iprintf("IP address changed to ");
    ShowIP(newIP);
    iprintf("\n");

    // Announce new IP to network
    sendGratuitousArp(iface, newIP);

    iprintf("Gratuitous ARP sent - network caches updated\n");
}
```

Example: Failover Scenario

```

void takeOverBackupIP(IPADDR4 backupIP) {
    int iface = GetFirstInterface();

    iprintf("Taking over backup IP ");
    ShowIP(backupIP);
    iprintf("...\n");

    // Configure our interface with the backup IP
    InterfaceBlock *ib = GetInterfaceBlock(iface);
    ib->ip4.cur_addr = backupIP;

    // Immediately announce ownership of this IP
    sendGratuitousArp(iface, backupIP);

    // Send multiple times for reliability
    for (int i = 0; i < 3; i++) {
        OSTimeDly(TICKS_PER_SECOND / 10);
        sendGratuitousArp(iface, backupIP);
    }

    iprintf("Failover complete - now responding on backup IP\n");
}

```

Example: Post-Conflict Resolution

```

void resolveIPConflict() {
    int iface = GetFirstInterface();
    IPADDR4 currentIP = InterfaceIP4(iface);

    // Detected conflict - change to different IP
    IPADDR4 newIP = AsciiToIp4("192.168.1.101");

    InterfaceBlock *ib = GetInterfaceBlock(iface);
    ib->ip4.cur_addr = newIP;

    iprintf("IP conflict resolved - changed to ");
    ShowIP(newIP);
    iprintf("\n");

    // Announce new IP to network
    sendGratuitousArp(iface, newIP);
}

```

Example: Multi-Interface System

```

void announceAllInterfaces() {
    int numInterfaces = GetNumInterfaces();

    for (int i = 0; i < numInterfaces; i++) {
        IPADDR4 ifaceIP = InterfaceIP4(i);
        if (ifaceIP != 0) { // Valid IP configured
            sendGratuitousArp(i, ifaceIP);
            iprintf("Interface %d announced\n", i);
        }
    }
}

```

See also

[IsMyAddressUsedArpDetect\(\)](#) To detect IP conflicts

[GetArpMacFromIp\(\)](#) To query ARP cache

[InterfaceIP4\(\)](#) To get interface IP address

14.2.11.6 ShowArp()

```

void ShowArp ()
#include <arp.h>

```

Display the current [ARP](#) cache contents to the standard output.

Displays all entries in the [ARP](#) cache to the stdio serial port (typically the debug serial port). Each entry shows the IP address, MAC address, interface number, and entry status.

This function is useful for debugging network connectivity issues, verifying [ARP](#) resolution, and monitoring which devices are in the cache.

Postcondition

ARP cache contents are written to stdio serial port

Note

Output format includes IP address, MAC address, interface, and flags

Output goes to the default stdio serial port (cannot be redirected)

For file or socket output, use [fShowArp\(\)](#) instead

Output Format Example:

IP Address	MAC Address	Iface	Flags
192.168.1.1	00:11:22:33:44:55	0	P
192.168.1.100	AA:BB:CC:DD:EE:FF	0	
192.168.1.50	12:34:56:78:9A:BC	0	S

Where flags may be:

- P: Permanent entry
- S: Static entry
- (blank): Dynamic entry

Example: Basic Debugging

```
void displayNetworkInfo() {
    iprintf("\n== Network ARP Cache ==\n");
    ShowArp();
    iprintf("=====\n\n");
}
```

Example: Periodic Cache Display

```
void monitorArpCache(void *pd) {
    while (1) {
        OSTimeDly(TICKS_PER_SECOND * 30); // Every 30 seconds
        iprintf("\n--- ARP Cache Update ---\n");
        ShowArp();
    }
}

void UserMain(void *pd) {
    init();
    OSSimpleTaskCreateName(monitorArpCache, MAIN_PRIO - 1, "ArpMonitor");
    // ...
}
```

Example: Troubleshooting Connectivity

```
void troubleshootConnection(IPADDR4 targetIP) {
    iprintf("Checking connection to ");
    ShowIP(targetIP);
    iprintf("\n");

    // Display current ARP cache
    ShowArp();

    // Try to ping device
    iprintf("\nSending ping...\n");
    Ping4(targetIP, 3, 2);

    // Display updated cache
    iprintf("\nUpdated ARP cache:\n");
    ShowArp();
}
```

See also

[fShowArp\(\)](#) For output to FILE pointer

[GetArpMacFromIp\(\)](#) To query specific ARP entries

14.2.12 Variable Documentation

14.2.12.1 hard_size

`uint8_t ARP::hard_size`

Hardware address length (6 for MAC)

Hardware address length (HLEN). For Ethernet (MAC) this is 6 bytes.

14.2.12.2 hard_Type

`beuint16_t ARP::hard_Type`

Hardware type (1 for Ethernet)

Hardware type (HTYPE). For Ethernet this is 1. Network byte order (big-endian).

14.2.12.3 op_code

`beuint16_t ARP::op_code`

Operation: 1=Request, 2=Reply.

Operation code. Network byte order (big-endian).

- 1 = [ARP](#) Request
- 2 = [ARP](#) Reply
- 3 = RARP Request
- 4 = RARP Reply

14.2.12.4 prot_size

`uint8_t ARP::prot_size`

Protocol address length (4 for IPv4)

Protocol address length (PLEN). For IPv4 this is 4 bytes.

14.2.12.5 prot_Type

`beuint16_t ARP::prot_Type`

Protocol type (0x0800 for IPv4)

Protocol type (PTYPE). For IPv4 this is 0x0800. Network byte order (big-endian).

14.2.12.6 sender_ip

`IPADDR4 ARP::sender_ip`

Sender IP address.

Sender protocol (IP) address. The IPv4 address of the device sending this [ARP](#) packet.

14.2.12.7 sender_phy

`MACADR ARP::sender_phy`

Sender MAC address.

Sender hardware (MAC) address. The MAC address of the device sending this [ARP](#) packet.

14.2.12.8 target_ip

`IPADDR4 ARP::target_ip`

Target IP address.

Target protocol (IP) address. The IPv4 address being queried (in requests) or resolved (in replies).

Note

This structure is packed to match exact network packet format

All multi-byte fields use network byte order (big-endian)

Do not change structure without careful consideration of network compatibility

Warning

Structure changes will break network protocol compatibility

Always add new fields at the end to maintain compatibility

Typical ARP Request Packet:

```

ARP request;
request.hard_Type = htons(1);           // Ethernet
request.prot_Type = htons(0x0800);      // IPv4
request.hard_size = 6;                  // MAC address length
request.prot_size = 4;                  // IPv4 address length
request.op_code = htons(1);             // ARP Request
// sender_phy = our MAC
// sender_ip = our IP
// target_phy = 00:00:00:00:00:00 (unknown)
// target_ip = IP we're looking for

```

See also

RFC 826 - Address Resolution Protocol specification

14.2.12.9 target_phy

`MACADDR ARP::target_phy`

Target MAC address.

Target hardware (MAC) address. In [ARP](#) requests this is typically all zeros. In replies, this contains the requested MAC address.

14.3 AT Commands

Functions for serial modem AT commands.

Macros

- `#define AT_COMMAND_REASON_START_AT (1)`
`AT_COMMAND_REASON_START_AT.`
- `#define AT_COMMAND_REASON_EXIT_AT (2)`
`AT_COMMAND_REASON_EXIT_AT.`
- `#define AT_COMMAND_REASON_NORMAL_COMMAND (3)`
`AT_COMMAND_REASON_NORMAL_COMMAND.`
- `#define AT_COMMAND_OK (1)`
`AT_COMMAND_OK.`
- `#define AT_COMMAND_EXIT (2)`
`AT_COMMAND_EXIT.`

Typedefs

- typedef int [ProcessATcommandFunc](#)(int uartnum, const char *pCommand, FILE *pResponseFile, int reason)
typedef for the AT command processing function

Functions

- void **EnableATCommands** (int port_number, [ProcessATcommandFunc](#) *pcmdf, int task_priority)
Enable AT command on the specific serial port, port must already be popen'd in interrupt mode.
- void **DisableATCommands** (int port_number)
Disable AT command.

14.3.1 Detailed Description

Functions for serial modem AT commands.

```
#include< atcommand.h>
```

14.3.2 Typedef Documentation

14.3.2.1 ProcessATcommandFunc

```
typedef int ProcessATcommandFunc(int uartnum, const char *pCommand, FILE *pResponseFile, int reason)
```

```
#include <atcommand.h>
```

typedef for the AT command processing function

Returns

AT_COMMAND_OK or AT_COMMAND_EXIT

14.4 Base64

Functions

- int [Base64Decode](#) (const char *input, uint8_t *outputbuf, const char *pEndMarker=NULL)
Decodes a null terminated Base64 string into bytes.
- int [Base64UrlDecode](#) (const char *input, uint8_t *outputbuf, const char *pEndMarker=NULL)
Decodes a null terminated Base64Url string into bytes.
- int [Base64Encode](#) (uint8_t *input, int inlen, char *outputbuf)
Encodes arbitrary binary bytes into Base64 string.
- int [Base64UrlEncode](#) (uint8_t *input, int inlen, char *outputbuf)
Encodes arbitrary binary bytes into Base64Url string.
- int [Base64StreamEncode](#) (int fd, uint8_t *input, int inlen, char *outBuf, int maxOutLen)
Encodes arbitrary binary bytes into Base64 string and writes them to a file descriptor.
- int [Base64UrlStreamEncode](#) (int fd, uint8_t *input, int inlen, char *outBuf, int maxOutLen)
Encodes arbitrary binary bytes into Base64Url string and writes them to a file descriptor.

14.4.1 Detailed Description

```
#include< base64.h>
```

Functions to encode/decode in Base64

14.4.2 Function Documentation

14.4.2.1 Base64Decode()

```
int Base64Decode (  
    const char * input,  
    uint8_t * outputbuf,  
    const char * pEndMarker = NULL)
```

```
#include <base64.h>
```

Decodes a null terminated Base64 string into bytes.

Notes:

- outputbuf should be at least the size of input divided by 4, rounded up
- Does not null terminate results

Parameters

<i>input</i>	Base64 string
<i>outputbuf</i>	Resulting decoded data
<i>pEndMarker</i>	Pointer to the end of the string (in the event it is not terminated by either a null char or a base64 terminator)

Returns

Number of bytes returned in outputbuf

See also

[Base64Encode\(\)](#)

[Base64UrlDecode\(\)](#)

14.4.2.2 Base64Encode()

```
int Base64Encode (  
    uint8_t * input,  
    int inlen,  
    char * outputbuf)
```

```
#include <base64.h>
```

Encodes arbitrary binary bytes into Base64 string.

Notes: Does not null terminate results but terminates with an '='

Parameters

<i>input</i>	Data
<i>inlen</i>	Length of data
<i>outputbuf</i>	Resulting encoded data

Returns

Number of bytes returned in outputbuf

See also

[Base64Decode\(\)](#)

[Base64StreamEncode\(\)](#)

[Base64UrlEncode\(\)](#)

14.4.2.3 Base64StreamEncode()

```
int Base64StreamEncode (  
    int fd,  
    uint8_t * input,  
    int inlen,  
    char * outBuf,  
    int maxOutLen)
```

```
#include <base64.h>
```

Encodes arbitrary binary bytes into Base64 string and writes them to a file descriptor.

Notes: Does not null terminate results but terminates with an '='

Parameters

<i>fd</i>	Target File Descriptor
<i>input</i>	Data to be encoded
<i>inlen</i>	Length of data
<i>outBuf</i>	Temporary buffer to accumulate encoded data
<i>maxOutLen</i>	Maximum buffer space to accumulate data

Returns

Number of bytes returned in outputbuf

See also

[Base64Encode](#)

[Base64UrlStreamEncode](#)

14.4.2.4 Base64UrlDecode()

```
int Base64UrlDecode (  
    const char * input,  
    uint8_t * outputbuf,  
    const char * pEndMarker = NULL)
```

```
#include <base64.h>
```

Decodes a null terminated Base64Url string into bytes.

Notes:

- outputbuf should be at least the size of input divided by 4, rounded up
- Does not null terminate results

Parameters

<i>input</i>	Base64Url string
<i>outputbuf</i>	Resulting decoded data
<i>pEndMarker</i>	Pointer to the end of the string (in the event it is not terminated by either a null char or a base64 terminator)

Returns

Number of bytes returned in outputbuf

See also

[Base64UrlEncode\(\)](#)

[Base64Decode\(\)](#)

14.4.2.5 Base64UrlEncode()

```
int Base64UrlEncode (
    uint8_t * input,
    int inlen,
    char * outputbuf)
```

```
#include <base64.h>
```

Encodes arbitrary binary bytes into Base64Url string.

Notes: Does not null terminate results but terminates with an '='

Parameters

<i>input</i>	Data
<i>inlen</i>	Length of data
<i>outputbuf</i>	Resulting encoded data

Returns

Number of bytes returned in outputbuf

See also

[Base64UrlDecode\(\)](#)

[Base64UrlStreamEncode\(\)](#)

[Base64Encode\(\)](#)

14.4.2.6 Base64UrlStreamEncode()

```
int Base64UrlStreamEncode (
    int fd,
    uint8_t * input,
    int inlen,
    char * outBuf,
    int maxOutLen)
```

```
#include <base64.h>
```

Encodes arbitrary binary bytes into Base64Url string and writes them to a file descriptor.

Notes: Does not null terminate results but terminates with an '='

Parameters

<i>fd</i>	Target File Descriptor
<i>input</i>	Data to be encoded
<i>inlen</i>	Length of data
<i>outBuf</i>	Temporary buffer to accumulate encoded data
<i>maxOutLen</i>	Maximum buffer space to accumulate data

Returns

Number of bytes returned in outputbuf

See also

[Base64UrlEncode](#)

[Base64StreamEncode](#)

14.5 Buffers - System Buffer Pool

Topics

- [Buffer Ownership States](#)
- [Buffer State Flags](#)

Classes

- struct [pool_buffer](#)
Main buffer structure for network and serial communication.
- class [buffer_list](#)
Doubly-linked list management for buffers.
- class [fifo_buffer_storage](#)
FIFO buffer storage using linked pool buffers.
- class [SMPoolPtr](#)
Smart pointer wrapper for automatic buffer management.

Functions

- [PoolPtr GetBuffer](#) ()
Allocate a software buffer for general use.
- [PoolPtr GetFastBuffer](#) ()
Allocate a software buffer, preferring the fast memory pool.
- void [FreeBuffer](#) ([PoolPtr](#) nbuf)
Free a buffer and return it to the unused pool.
- void [FreeBufferList](#) ([PoolPtr](#) nbuf)
Free a linked list of buffers.
- void [IncUsageCount](#) ([PoolPtr](#) pp)
Increment the usage count of a buffer.
- uint16_t [GetFreeCount](#) ()
Returns the number of free buffers in the system. Buffers are used for both serial and network interfaces.
- void [InitBuffers](#) ()
Initialize the buffer system.
- void [ShowBuffer](#) ([PoolPtr](#) p)
Display buffer contents and state information.

14.5.1 Detailed Description

```
#include< buffers.h>
```

Functions associated with the buffers allocated for use by network interfaces and serial ports. If a system runs out of buffers it will no longer be able to send or receive network data.

14.5.2 Function Documentation

14.5.2.1 FreeBuffer()

```
void FreeBuffer (
    PoolPtr nbuf)
#include <buffers.h>
```

Free a buffer and return it to the unused pool.

Decrements the buffer's usage count and returns it to the free pool when the count reaches zero. This allows safe sharing of buffers between multiple system components.

Parameters

<i>nbuf</i>	Buffer to free

See also

[GetBuffer\(\)](#), [IncUsageCount\(\)](#)

14.5.2.2 FreeBufferList()

```
void FreeBufferList (  
    PoolPtr nbuf)  
#include <buffers.h>
```

Free a linked list of buffers.

Frees a chain of buffers linked by their pNextFifo_EI pointers. Each buffer in the list will be returned to the unused pool.

Parameters

<i>nbuf</i>	First buffer in the linked list to free

See also

[FreeBuffer\(\)](#)

14.5.2.3 GetBuffer()

```
PoolPtr GetBuffer ()  
#include <buffers.h>
```

Allocate a software buffer for general use.

Allocates a buffer from the system buffer pool for use by application software. The buffer will be marked with ownership state BO_SOFTWARE.

Returns

Pointer to allocated buffer, or NULL if no buffers available

See also

[FreeBuffer\(\)](#), [GetFastBuffer\(\)](#)

14.5.2.4 GetFastBuffer()

```
PoolPtr GetFastBuffer ()  
#include <buffers.h>
```

Allocate a software buffer, preferring the fast memory pool.

Similar to [GetBuffer\(\)](#), but attempts to allocate from the fast buffer pool first before falling back to the standard pool. Fast buffers may provide better performance for time-critical operations.

Returns

Pointer to allocated buffer, or NULL if no buffers available

See also

[GetBuffer\(\)](#), [FreeBuffer\(\)](#)

14.5.2.5 GetFreeCount()

```
uint16_t GetFreeCount ()
#include <buffers.h>
```

Returns the number of free buffers in the system. Buffers are used for both serial and network interfaces.

This function returns the number of free pool buffers, which are used for network and serial communication. The maximum number of buffers is defined in `\nburn\nbrtos\include\constants.h`:

```
#define BUFFER_POOL_SIZE (128)
```

The size of each buffer is defined as:

```
#define ETHER_BUFFER_SIZE 1548
```

If the number of free buffers reaches zero, then no further network communication will be possible until some buffers are freed. This is a good function to use as a debug tool for detecting buffer leaks in your application.

Serial ports also use buffers, which are allocated in `constants.h`:

```
#define SERIAL_TX_BUFFERS (2)
```

```
#define SERIAL_RX_BUFFERS (2)
```

Each serial port buffer is equal in size to an Ethernet buffer.

Returns

The number of free buffers in the system.

14.5.2.6 IncUsageCount()

```
void IncUsageCount (
    PoolPtr pp)
#include <buffers.h>
```

Increment the usage count of a buffer.

Increases the reference count for shared buffer usage. The buffer will not be freed until `FreeBuffer()` is called once for each reference (usage count reaches zero).

Parameters

<i>pp</i>	Buffer to increment usage count

See also

[FreeBuffer\(\)](#)

14.5.2.7 InitBuffers()

```
void InitBuffers ()
#include <buffers.h>
```

Initialize the buffer system.

Sets up the buffer pools and free lists. This function is called during system initialization and should not be called by application code.

See also

[GetFreeCount\(\)](#)

14.5.2.8 ShowBuffer()

```
void ShowBuffer (
    PoolPtr p)
#include <buffers.h>
```

Display buffer contents and state information.

Prints detailed information about a buffer including its state, size, and data contents to stdout. Useful for debugging.

Parameters

<i>p</i>	Buffer to display

14.5.3 Buffer Ownership States

Macros

- **#define BO_UNUSED 0**
The buffer is free and available for allocation.
- **#define BO_SOFTWARE 1**
The buffer belongs to application software.
- **#define BO_PRX 2**
The buffer is assigned to receive function.
- **#define BO_PTXF 3**
The buffer is to be transmitted then freed.
- **#define BO_PTXS 4**
The buffer is to be transmitted then returned to software.

14.5.3.1 Detailed Description

```
#include< buffers.h>
```

Buffer ownership states define which system component currently owns a buffer as it moves through the network stack.

14.5.4 Buffer State Flags

Macros

- **#define BS_PHY_BCAST 1**
Physical layer broadcast packet.
- **#define BS_IP_BCAST 2**
Layer 3 (IP) broadcast packet.
- **#define BS_IP_LOCAL_NET 4**
Packet to/from local subnet.
- **#define BS_TCP_PUSH 8**
TCP PUSH flag was set on receive.
- **#define BS_PHY_802_3 0x10**
Packet requires 802.3 encapsulation.
- **#define BS_VLAN 0x20**
Packet requires VLAN tagging.
- **#define BS_DBG_LOG 0x80**
Debug log buffer - do not log operations.

14.5.4.1 Detailed Description

```
#include< buffers.h>
```

Buffer state flags provide additional networking context for packet processing.

14.6 CAN

Topics

- [FlexCAN SOMRT1061](#)
- [MOD5441x, NANO54415, SB800EX](#)
- [MODM7AE70](#)

14.6.1 Detailed Description

Controller Area Network (CAN) API for NetBurner Platforms

14.6.2 FlexCAN SOMRT1061

Classes

- [struct _flexcan_frame](#)
FlexCAN message frame structure.
- [struct _flexcan_timing_config](#)
FlexCAN protocol timing characteristic configuration structure.
- [struct _flexcan_config](#)
FlexCAN module configuration structure.
- [struct _flexcan_rx_mb_config](#)
FlexCAN Receive Message Buffer configuration structure.
- [struct _flexcan_rx_fifo_config](#)
FlexCAN Legacy Rx FIFO configuration structure.
- [struct _flexcan_mb_transfer](#)
FlexCAN Message Buffer transfer.
- [struct _flexcan_fifo_transfer](#)
FlexCAN Rx FIFO transfer.
- [struct _flexcan_handle](#)
FlexCAN handle structure.

Macros

- [#define FLEXCAN_ID_STD\(id\)](#) (((uint32_t)((uint32_t)(id)) << CAN_ID_STD_SHIFT)) & CAN_ID_STD_MASK)
FlexCAN frame length helper macro.
- [#define FLEXCAN_ID_EXT\(id\)](#)
- [#define FLEXCAN_RX_MB_STD_MASK\(id, rtr, ide\)](#)
FlexCAN Rx Message Buffer Mask helper macro.
- [#define FLEXCAN_RX_MB_EXT_MASK\(id, rtr, ide\)](#)
- [#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_A\(id, rtr, ide\)](#)
FlexCAN Legacy Rx FIFO Mask helper macro.
- [#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH\(id, rtr, ide\)](#)
- [#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW\(id, rtr, ide\)](#)
- [#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH\(id\)](#) (((uint32_t)(id)&0x7F8) << 21)
- [#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH\(id\)](#) (((uint32_t)(id)&0x7F8) << 13)
- [#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW\(id\)](#) (((uint32_t)(id)&0x7F8) << 5)
- [#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW\(id\)](#) (((uint32_t)(id)&0x7F8) >> 3)
- [#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A\(id, rtr, ide\)](#)
- [#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH\(id, rtr, ide\)](#)
- [#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW\(id, rtr, ide\)](#)
- [#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH\(id\)](#) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) << 3)

- `#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(id)`
- `#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(id)`
- `#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 21)`
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(id, rtr, ide) FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)`
FlexCAN Rx FIFO Filter helper macro.
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(id, rtr, ide)`
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(id, rtr, ide)`
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(id)`
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(id)`
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(id)`
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(id)`
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(id, rtr, ide) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)`
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(id, rtr, ide)`
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(id, rtr, ide)`
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(id)`
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(id)`
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(id)`
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)`
- `#define FLEXCAN_ERROR_AND_STATUS_INIT_FLAG`
FlexCAN interrupt/status flag helper macro.
- `#define FLEXCAN_MEMORY_ENHANCED_RX_FIFO_INIT_FLAG (0U)`
FlexCAN Enhanced Rx FIFO base address helper macro.
- `#define FLEXCAN_CALLBACK(x) void(x)(CAN_Type * base, flexcan_handle_t * handle, status_t status, uint32_t result, void *userData)`
FlexCAN transfer callback function.

Typedefs

- `typedef enum _flexcan_frame_format flexcan_frame_format_t`
FlexCAN frame format.
- `typedef enum _flexcan_frame_type flexcan_frame_type_t`
FlexCAN frame type.
- `typedef enum _flexcan_clock_source flexcan_clock_source_t`
FlexCAN clock source.
- `typedef enum _flexcan_wake_up_source flexcan_wake_up_source_t`
FlexCAN wake up source.
- `typedef enum _flexcan_rx_fifo_filter_type flexcan_rx_fifo_filter_type_t`
FlexCAN Rx Fifo Filter type.
- `typedef enum _flexcan_rx_fifo_priority flexcan_rx_fifo_priority_t`
FlexCAN Enhanced/Legacy Rx FIFO priority.
- `typedef struct _flexcan_frame flexcan_frame_t`
FlexCAN message frame structure.
- `typedef struct _flexcan_timing_config flexcan_timing_config_t`
FlexCAN protocol timing characteristic configuration structure.
- `typedef struct _flexcan_config flexcan_config_t`
FlexCAN module configuration structure.
- `typedef struct _flexcan_rx_mb_config flexcan_rx_mb_config_t`
FlexCAN Receive Message Buffer configuration structure.
- `typedef struct _flexcan_rx_fifo_config flexcan_rx_fifo_config_t`
FlexCAN Legacy Rx FIFO configuration structure.

- typedef struct `_flexcan_mb_transfer` `flexcan_mb_transfer_t`
FlexCAN Message Buffer transfer.
- typedef struct `_flexcan_fifo_transfer` `flexcan_fifo_transfer_t`
FlexCAN Rx FIFO transfer.
- typedef struct `_flexcan_handle` `flexcan_handle_t`
FlexCAN handle structure definition.

Enumerations

- enum {
`kStatus_FLEXCAN_TxBusy` = MAKE_STATUS(kStatusGroup_FLEXCAN, 0) , `kStatus_FLEXCAN_TxIdle` =
MAKE_STATUS(kStatusGroup_FLEXCAN, 1) , `kStatus_FLEXCAN_TxSwitchToRx` , `kStatus_FLEXCAN_RxBusy`
= MAKE_STATUS(kStatusGroup_FLEXCAN, 3) ,
`kStatus_FLEXCAN_RxIdle` = MAKE_STATUS(kStatusGroup_FLEXCAN, 4) , `kStatus_FLEXCAN_RxOverflow`
= MAKE_STATUS(kStatusGroup_FLEXCAN, 5) , `kStatus_FLEXCAN_RxFifoBusy` = MAKE_STATUS(k↵
StatusGroup_FLEXCAN, 6) , `kStatus_FLEXCAN_RxFifoIdle` = MAKE_STATUS(kStatusGroup_FLEXCAN, 7)
,
`kStatus_FLEXCAN_RxFifoOverflow` = MAKE_STATUS(kStatusGroup_FLEXCAN, 8) , `kStatus_FLEXCAN_RxFifoWarning`
= MAKE_STATUS(kStatusGroup_FLEXCAN, 9) , `kStatus_FLEXCAN_RxFifoDisabled` , `kStatus_FLEXCAN_ErrorStatus`
= MAKE_STATUS(kStatusGroup_FLEXCAN, 11) ,
`kStatus_FLEXCAN_WakeUp` = MAKE_STATUS(kStatusGroup_FLEXCAN, 12) , `kStatus_FLEXCAN_UnHandled`
= MAKE_STATUS(kStatusGroup_FLEXCAN, 13) , `kStatus_FLEXCAN_RxRemote` = MAKE_STATUS(k↵
StatusGroup_FLEXCAN, 14) }
FlexCAN transfer status.
- enum `_flexcan_frame_format` { `kFLEXCAN_FrameFormatStandard` = 0x0U , `kFLEXCAN_FrameFormatExtend`
= 0x1U }
FlexCAN frame format.
- enum `_flexcan_frame_type` { `kFLEXCAN_FrameTypeData` = 0x0U , `kFLEXCAN_FrameTypeRemote` = 0x1U
}
FlexCAN frame type.
- enum `_flexcan_clock_source` { `kFLEXCAN_ClkSrcOsc` = 0x0U , `kFLEXCAN_ClkSrcPeri` = 0x1U ,
`kFLEXCAN_ClkSrc0` = 0x0U , `kFLEXCAN_ClkSrc1` = 0x1U }
FlexCAN clock source.
- enum `_flexcan_wake_up_source` { `kFLEXCAN_WakeupSrcUnfiltered` = 0x0U , `kFLEXCAN_WakeupSrcFiltered`
= 0x1U }
FlexCAN wake up source.
- enum `_flexcan_rx_fifo_filter_type` { `kFLEXCAN_RxFifoFilterTypeA` = 0x0U , `kFLEXCAN_RxFifoFilterTypeB` ,
`kFLEXCAN_RxFifoFilterTypeC` , `kFLEXCAN_RxFifoFilterTypeD` = 0x3U }
FlexCAN Rx Fifo Filter type.
- enum `_flexcan_rx_fifo_priority` { `kFLEXCAN_RxFifoPrioLow` = 0x0U , `kFLEXCAN_RxFifoPrioHigh` = 0x1U }
FlexCAN Enhanced/Legacy Rx FIFO priority.
- enum `_flexcan_interrupt_enable` {
`kFLEXCAN_BusOffInterruptEnable` = CAN_CTRL1_BOFFMSK_MASK , `kFLEXCAN_ErrorInterruptEnable`
= CAN_CTRL1_ERRMSK_MASK , `kFLEXCAN_TxWarningInterruptEnable` = CAN_CTRL1_TWRNMSK_↵
MASK , `kFLEXCAN_RxWarningInterruptEnable` = CAN_CTRL1_RWRNMSK_MASK ,
`kFLEXCAN_WakeUpInterruptEnable` = CAN_MCR_WAKMSK_MASK }
FlexCAN interrupt enable enumerations.
- enum `_flexcan_flags` {
`kFLEXCAN_SynchFlag` = CAN_ESR1_SYNCH_MASK , `kFLEXCAN_TxWarningIntFlag` = CAN_ESR1_↵
TWRNINT_MASK , `kFLEXCAN_RxWarningIntFlag` = CAN_ESR1_RWRNINT_MASK , `kFLEXCAN_IdleFlag`
= CAN_ESR1_IDLE_MASK ,
`kFLEXCAN_FaultConfinementFlag` = CAN_ESR1_FLTCONF_MASK , `kFLEXCAN_TransmittingFlag` =
CAN_ESR1_TX_MASK , `kFLEXCAN_ReceivingFlag` = CAN_ESR1_RX_MASK , `kFLEXCAN_BusOffIntFlag`
= CAN_ESR1_BOFFINT_MASK ,
`kFLEXCAN_ErrorIntFlag` = CAN_ESR1_ERRINT_MASK , `kFLEXCAN_WakeUpIntFlag` = CAN_ESR1_↵
WAKINT_MASK , `kFLEXCAN_ErrorFlag` }

FlexCAN status flags.

- enum `_flexcan_error_flags` {
`kFLEXCAN_TxErrorWarningFlag` = `CAN_ESR1_TXWRN_MASK` , `kFLEXCAN_RxErrorWarningFlag` =
`CAN_ESR1_RXWRN_MASK` , `kFLEXCAN_StuffingError` = `CAN_ESR1_STFERR_MASK` , `kFLEXCAN_FormError`
= `CAN_ESR1_FRMERR_MASK` ,
`kFLEXCAN_CrcError` = `CAN_ESR1_CRCERR_MASK` , `kFLEXCAN_AckError` = `CAN_ESR1_ACKERR_`
`_MASK` , `kFLEXCAN_Bit0Error` = `CAN_ESR1_BIT0ERR_MASK` , `kFLEXCAN_Bit1Error` = `CAN_ESR1_`
`BIT1ERR_MASK` }

FlexCAN error status flags.

- enum { `kFLEXCAN_RxFifoOverflowFlag` = `CAN_IFLAG1_BUF7I_MASK` , `kFLEXCAN_RxFifoWarningFlag` =
`CAN_IFLAG1_BUF6I_MASK` , `kFLEXCAN_RxFifoFrameAvlFlag` = `CAN_IFLAG1_BUF5I_MASK` }

FlexCAN Legacy Rx FIFO status flags.

Initialization and deinitialization

- void `FLEXCAN_EnterFreezeMode` (`CAN_Type *base`)
Enter FlexCAN Freeze Mode.
- void `FLEXCAN_ExitFreezeMode` (`CAN_Type *base`)
Exit FlexCAN Freeze Mode.
- uint32_t `FLEXCAN_GetInstance` (`CAN_Type *base`)
Get the FlexCAN instance from peripheral base address.
- bool `FLEXCAN_CalculateImprovedTimingValues` (`CAN_Type *base`, uint32_t bitRate, uint32_t sourceClock_Hz, `flexcan_timing_config_t *pTimingConfig`)
Calculates the improved timing values by specific bit Rates for classical CAN.
- void `FLEXCAN_Init` (`CAN_Type *base`, const `flexcan_config_t *pConfig`, uint32_t sourceClock_Hz)
Initializes a FlexCAN instance.
- void `FLEXCAN_Deinit` (`CAN_Type *base`)
De-initializes a FlexCAN instance.
- void `FLEXCAN_GetDefaultConfig` (`flexcan_config_t *pConfig`)
Gets the default configuration structure.

Configuration.

- void `FLEXCAN_SetTimingConfig` (`CAN_Type *base`, const `flexcan_timing_config_t *pConfig`)
Sets the FlexCAN classical CAN protocol timing characteristic.
- status_t `FLEXCAN_SetBitRate` (`CAN_Type *base`, uint32_t sourceClock_Hz, uint32_t bitRate_Bps)
Set bit rate of FlexCAN classical CAN frame or CAN FD frame nominal phase.
- void `FLEXCAN_SetRxMbGlobalMask` (`CAN_Type *base`, uint32_t mask)
Sets the FlexCAN receive message buffer global mask.
- void `FLEXCAN_SetRxFifoGlobalMask` (`CAN_Type *base`, uint32_t mask)
Sets the FlexCAN receive FIFO global mask.
- void `FLEXCAN_SetRxIndividualMask` (`CAN_Type *base`, uint8_t maskIdx, uint32_t mask)
Sets the FlexCAN receive individual mask.
- void `FLEXCAN_SetTxMbConfig` (`CAN_Type *base`, uint8_t mbIdx, bool enable)
Configures a FlexCAN transmit message buffer.
- void `FLEXCAN_SetRxMbConfig` (`CAN_Type *base`, uint8_t mbIdx, const `flexcan_rx_mb_config_t *pRxMbConfig`, bool enable)
Configures a FlexCAN Receive Message Buffer.
- void `FLEXCAN_SetRxFifoConfig` (`CAN_Type *base`, const `flexcan_rx_fifo_config_t *pRxFifoConfig`, bool enable)
Configures the FlexCAN Legacy Rx FIFO.

Bus Operations

- status_t **FLEXCAN_WriteTxMb** (CAN_Type *base, uint8_t mbldx, const flexcan_frame_t *pTxFrame)
Writes a FlexCAN Message to the Transmit Message Buffer.
- status_t **FLEXCAN_ReadRxMb** (CAN_Type *base, uint8_t mbldx, flexcan_frame_t *pRxFrame)
Reads a FlexCAN Message from Receive Message Buffer.
- status_t **FLEXCAN_ReadRxFifo** (CAN_Type *base, flexcan_frame_t *pRxFrame)
Reads a FlexCAN Message from Legacy Rx FIFO.

Transactional

- status_t **FLEXCAN_TransferSendBlocking** (CAN_Type *base, uint8_t mbldx, flexcan_frame_t *pTxFrame)
Performs a polling send transaction on the CAN bus.
- status_t **FLEXCAN_TransferReceiveBlocking** (CAN_Type *base, uint8_t mbldx, flexcan_frame_t *pRxFrame)
Performs a polling receive transaction on the CAN bus.
- status_t **FLEXCAN_TransferReceiveFifoBlocking** (CAN_Type *base, flexcan_frame_t *pRxFrame)
Performs a polling receive transaction from Legacy Rx FIFO on the CAN bus.
- void **FLEXCAN_TransferCreateHandle** (CAN_Type *base, flexcan_handle_t *handle, flexcan_transfer_t callback_t callback, void *userData)
Initializes the FlexCAN handle.
- status_t **FLEXCAN_TransferSendNonBlocking** (CAN_Type *base, flexcan_handle_t *handle, flexcan_mb_transfer_t *pMbXfer)
Sends a message using IRQ.
- status_t **FLEXCAN_TransferReceiveNonBlocking** (CAN_Type *base, flexcan_handle_t *handle, flexcan_mb_transfer_t *pMbXfer)
Receives a message using IRQ.
- status_t **FLEXCAN_TransferReceiveFifoNonBlocking** (CAN_Type *base, flexcan_handle_t *handle, flexcan_fifo_transfer_t *pFifoXfer)
Receives a message from Rx FIFO using IRQ.
- status_t **FLEXCAN_TransferGetReceiveFifoCount** (CAN_Type *base, flexcan_handle_t *handle, size_t *count)
Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.
- uint32_t **FLEXCAN_GetTimeStamp** (flexcan_handle_t *handle, uint8_t mbldx)
Gets the detail index of Mailbox's Timestamp by handle.
- void **FLEXCAN_TransferAbortSend** (CAN_Type *base, flexcan_handle_t *handle, uint8_t mbldx)
Aborts the interrupt driven message send process.
- void **FLEXCAN_TransferAbortReceive** (CAN_Type *base, flexcan_handle_t *handle, uint8_t mbldx)
Aborts the interrupt driven message receive process.
- void **FLEXCAN_TransferAbortReceiveFifo** (CAN_Type *base, flexcan_handle_t *handle)
Aborts the interrupt driven message receive from Rx FIFO process.
- void **FLEXCAN_TransferHandleIRQ** (CAN_Type *base, flexcan_handle_t *handle)
FlexCAN IRQ handle function.

Driver version

- #define **FSL_FLEXCAN_DRIVER_VERSION** (MAKE_VERSION(2, 11, 6))
FlexCAN driver version.

14.6.2.1 Detailed Description

14.6.2.2 Macro Definition Documentation

14.6.2.2.1 FLEXCAN_CALLBACK

```
#define FLEXCAN_CALLBACK(
    x) void(x)(CAN_Type * base, flexcan_handle_t * handle, status_t status, uint32_t result, void *userData)
```

#include <fsl_flexcan.h>
FlexCAN transfer callback function.
The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to kStatus_FLEXCAN_ErrorStatus, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

14.6.2.2.2 FLEXCAN_ID_EXT

```
#define FLEXCAN_ID_EXT(
    id)
#include <fsl_flexcan.h>
```

Value:
(((uint32_t)((uint32_t)(id)) << CAN_ID_EXT_SHIFT)) & \
(CAN_ID_EXT_MASK | CAN_ID_STD_MASK))

Extend Frame ID helper macro.

14.6.2.2.3 FLEXCAN_ID_STD

```
#define FLEXCAN_ID_STD(
    id) (((uint32_t)((uint32_t)(id)) << CAN_ID_STD_SHIFT)) & CAN_ID_STD_MASK)
#include <fsl_flexcan.h>
```

FlexCAN frame length helper macro.
FlexCAN Frame ID helper macro. Standard Frame ID helper macro.

14.6.2.2.4 FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A

```
#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(
    id,
    rtr,
    ide) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)
#include <fsl_flexcan.h>
```

Extend Rx FIFO Filter helper macro Type A helper macro.

14.6.2.2.5 FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH

```
#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(
    id,
    rtr,
    ide)
#include <fsl_flexcan.h>
```

Value:
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(id, rtr, ide)

Extend Rx FIFO Filter helper macro Type B upper part helper macro.

14.6.2.2.6 FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW

```
#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(
    id,
    rtr,
    ide)
```

```
#include <fsl_flexcan.h>
```

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(          \
    id, rtr, ide)
```

Extend Rx FIFO Filter helper macro Type B lower part helper macro.

14.6.2.2.7 FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH

```
#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(
    id)
```

```
#include <fsl_flexcan.h>
```

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(          \
    id)
```

Extend Rx FIFO Filter helper macro Type C upper part helper macro.

14.6.2.2.8 FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW

```
#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(
    id) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)
```

```
#include <fsl_flexcan.h>
```

Extend Rx FIFO Filter helper macro Type C lower part helper macro.

14.6.2.2.9 FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH

```
#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(
    id)
```

```
#include <fsl_flexcan.h>
```

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(      \
    id)
```

Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.

14.6.2.2.10 FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW

```
#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(
    id)
```

```
#include <fsl_flexcan.h>
```

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(      \
    id)
```

Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.

14.6.2.2.11 FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A

```
#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(
    id,
    rtr,
    ide)
```

```
#include <fsl_flexcan.h>
```

Value:

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
(FLEXCAN_ID_EXT(id) << 1)
```

Extend Rx FIFO Mask helper macro Type A helper macro.

14.6.2.2.12 FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH

```
#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(
    id,
    rtr,
    ide)
```

```
#include <fsl_flexcan.h>
```

Value:

```
(
    ((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
    ((FLEXCAN_ID_EXT(id) & 0x1FFF8000)
     << 1))
\
```

Extend Rx FIFO Mask helper macro Type B upper part helper macro.

14.6.2.2.13 FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW

```
#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(
    id,
    rtr,
    ide)
#include <fsl_flexcan.h>
```

Value:

```
((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \
((FLEXCAN_ID_EXT(id) & 0x1FFF8000) >>
15))
\
```

Extend Rx FIFO Mask helper macro Type B lower part helper macro.

14.6.2.2.14 FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH

```
#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(
    id) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) << 3)
#include <fsl_flexcan.h>
```

Extend Rx FIFO Mask helper macro Type C upper part helper macro.

14.6.2.2.15 FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW

```
#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(
    id) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 21)
#include <fsl_flexcan.h>
```

Extend Rx FIFO Mask helper macro Type C lower part helper macro.

14.6.2.2.16 FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH

```
#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(
    id)
#include <fsl_flexcan.h>
```

Value:

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >>
5)
\
```

Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.

14.6.2.2.17 FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW

```
#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(
    id)
#include <fsl_flexcan.h>
```

Value:

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >>
13)
\
```

Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.

14.6.2.2.18 FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A

```
#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(
    id,
    rtr,
    ide) FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)
#include <fsl_flexcan.h>
```

FlexCAN Rx FIFO Filter helper macro.

Standard Rx FIFO Filter helper macro Type A helper macro.

14.6.2.2.19 FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH

```
#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(
    id,
    rtr,
    ide)
#include <fsl_flexcan.h>
```

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(          \
    id, rtr, ide)
```

Standard Rx FIFO Filter helper macro Type B upper part helper macro.

14.6.2.2.20 FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW

```
#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(
    id,
    rtr,
    ide)
#include <fsl_flexcan.h>
```

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(          \
    id, rtr, ide)
```

Standard Rx FIFO Filter helper macro Type B lower part helper macro.

14.6.2.2.21 FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH

```
#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(
    id)
#include <fsl_flexcan.h>
```

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(          \
    id)
```

Standard Rx FIFO Filter helper macro Type C upper part helper macro.

14.6.2.2.22 FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW

```
#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(
    id)
#include <fsl_flexcan.h>
```

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(          \
    id)
```

Standard Rx FIFO Filter helper macro Type C lower part helper macro.

14.6.2.2.23 FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH

```
#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(
    id)
#include <fsl_flexcan.h>
```

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(          \
    id)
```

Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.

14.6.2.2.24 FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW

```
#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(
    id)
#include <fsl_flexcan.h>
```

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(          \
    id)
```

Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.

14.6.2.2.25 FLEXCAN_RX_FIFO_STD_MASK_TYPE_A

```
#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(
    id,
    rtr,
    ide)
#include <fsl_flexcan.h>
```

Value:

```
((uint32_t)((uint32_t)(rtr) < 31) | (uint32_t)((uint32_t)(ide) < 30)) | \
(FLEXCAN_ID_STD(id) < 1))
```

FlexCAN Legacy Rx FIFO Mask helper macro.

Standard Rx FIFO Mask helper macro Type A helper macro.

14.6.2.2.26 FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH

```
#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(
    id,
    rtr,
    ide)
#include <fsl_flexcan.h>
```

Value:

```
((uint32_t)((uint32_t)(rtr) < 31) | (uint32_t)((uint32_t)(ide) < 30)) | \
((uint32_t)(id)&0x7FF) < 19))
```

Standard Rx FIFO Mask helper macro Type B upper part helper macro.

14.6.2.2.27 FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW

```
#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(
    id,
    rtr,
    ide)
#include <fsl_flexcan.h>
```

Value:

```
((uint32_t)((uint32_t)(rtr) < 15) | (uint32_t)((uint32_t)(ide) < 14)) | \
((uint32_t)(id)&0x7FF) < 3))
```

Standard Rx FIFO Mask helper macro Type B lower part helper macro.

14.6.2.2.28 FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH

```
#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(
    id) ((uint32_t)(id)&0x7F8) << 21)
#include <fsl_flexcan.h>
```

Standard Rx FIFO Mask helper macro Type C upper part helper macro.

14.6.2.2.29 FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW

```
#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(
    id) ((uint32_t)(id)&0x7F8) >> 3)
#include <fsl_flexcan.h>
```

Standard Rx FIFO Mask helper macro Type C lower part helper macro.

14.6.2.2.30 FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH

```
#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(
    id) ((uint32_t)(id)&0x7F8) << 13)
#include <fsl_flexcan.h>
```

Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.

14.6.2.2.31 FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW

```
#define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(
    id) ((uint32_t)(id)&0x7F8) << 5)
```



```
#include <fsl_flexcan.h>
```

Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.

14.6.2.2.32 FLEXCAN_RX_MB_EXT_MASK

```
#define FLEXCAN_RX_MB_EXT_MASK(
```

```
    id,
```

```
    rtr,
```

```
    ide)
```

```
#include <fsl_flexcan.h>
```

Value:

```
((uint32_t)((uint32_t)(rtr) < 31) | (uint32_t)((uint32_t)(ide) < 30)) | \
FLEXCAN_ID_EXT(id)
```

Extend Rx Message Buffer Mask helper macro.

14.6.2.2.33 FLEXCAN_RX_MB_STD_MASK

```
#define FLEXCAN_RX_MB_STD_MASK(
```

```
    id,
```

```
    rtr,
```

```
    ide)
```

```
#include <fsl_flexcan.h>
```

Value:

```
((uint32_t)((uint32_t)(rtr) < 31) | (uint32_t)((uint32_t)(ide) < 30)) | \
FLEXCAN_ID_STD(id)
```

FlexCAN Rx Message Buffer Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

14.6.2.3 Typedef Documentation

14.6.2.3.1 flexcan_clock_source_t

```
typedef enum _flexcan_clock_source flexcan_clock_source_t
```

```
#include <fsl_flexcan.h>
```

FlexCAN clock source.

Deprecated Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0
Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

14.6.2.3.2 flexcan_config_t

```
typedef struct _flexcan_config flexcan_config_t
```

```
#include <fsl_flexcan.h>
```

FlexCAN module configuration structure.

Deprecated Do not use the baudRate. It has been superceded bitRate
Do not use the baudRateFD. It has been superceded bitRateFD

14.6.2.3.3 flexcan_rx_fifo_priority_t

```
typedef enum _flexcan_rx_fifo_priority flexcan_rx_fifo_priority_t
```

```
#include <fsl_flexcan.h>
```

FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

14.6.2.3.4 flexcan_rx_mb_config_t

```
typedef struct _flexcan_rx_mb_config flexcan_rx_mb_config_t
#include <fsl_flexcan.h>
```

FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of [FLEXCAN_SetRxMbConfig\(\)](#) function. The [FLEXCAN_SetRxMbConfig\(\)](#) function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

14.6.2.4 Enumeration Type Documentation

14.6.2.4.1 anonymous enum

```
anonymous enum
#include <fsl_flexcan.h>
FlexCAN transfer status.
```

Enumerator

kStatus_FLEXCAN_TxBusy	Tx Message Buffer is Busy.
kStatus_FLEXCAN_TxIdle	Tx Message Buffer is Idle.
kStatus_FLEXCAN_TxSwitchToRx	Remote Message is send out and Message buffer changed to Receive one.
kStatus_FLEXCAN_RxBusy	Rx Message Buffer is Busy.
kStatus_FLEXCAN_RxIdle	Rx Message Buffer is Idle.
kStatus_FLEXCAN_RxOverflow	Rx Message Buffer is Overflowed.
kStatus_FLEXCAN_RxFifoBusy	Rx Message FIFO is Busy.
kStatus_FLEXCAN_RxFifoIdle	Rx Message FIFO is Idle.
kStatus_FLEXCAN_RxFifoOverflow	Rx Message FIFO is overflowed.
kStatus_FLEXCAN_RxFifoWarning	Rx Message FIFO is almost overflowed.
kStatus_FLEXCAN_RxFifoDisabled	Rx Message FIFO is disabled during reading.
kStatus_FLEXCAN_ErrorStatus	FlexCAN Module Error and Status.
kStatus_FLEXCAN_WakeUp	FlexCAN is waken up from STOP mode.
kStatus_FLEXCAN_UnHandled	UnHadled Interrupt asserted.
kStatus_FLEXCAN_RxRemote	Rx Remote Message Received in Mail box.

14.6.2.4.2 anonymous enum

```
anonymous enum
#include <fsl_flexcan.h>
FlexCAN Legacy Rx FIFO status flags.
```

The FlexCAN Legacy Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ~ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

Enumerator

kFLEXCAN_RxFifoOverflowFlag	Rx FIFO overflow flag.
kFLEXCAN_RxFifoWarningFlag	Rx FIFO almost full flag.
kFLEXCAN_RxFifoFrameAvlFlag	Frames available in Rx FIFO flag.

14.6.2.4.3 _flexcan_clock_source

```
enum _flexcan_clock_source
#include <fsl_flexcan.h>
FlexCAN clock source.
```

Deprecated Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0
Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

Enumerator

kFLEXCAN_ClkSrcOsc	FlexCAN Protocol Engine clock from Oscillator.
kFLEXCAN_ClkSrcPeri	FlexCAN Protocol Engine clock from Peripheral Clock.
kFLEXCAN_ClkSrc0	FlexCAN Protocol Engine clock selected by user as SRC == 0.
kFLEXCAN_ClkSrc1	FlexCAN Protocol Engine clock selected by user as SRC == 1.

14.6.2.4.4 _flexcan_error_flags

```
enum _flexcan_error_flags
#include <fsl_flexcan.h>
FlexCAN error status flags.
```

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with KFLEXCAN_ErrorFlag in [_flexcan_flags](#) enumerations to determine which error is generated.

Enumerator

kFLEXCAN_TxErrorWarningFlag	Tx Error Warning Status.
kFLEXCAN_RxErrorWarningFlag	Rx Error Warning Status.
kFLEXCAN_StuffingError	Stuffing Error.
kFLEXCAN_FormError	Form Error.
kFLEXCAN_CrcError	Cyclic Redundancy Check Error.
kFLEXCAN_AckError	Received no ACK on transmission.
kFLEXCAN_Bit0Error	Unable to send dominant bit.
kFLEXCAN_Bit1Error	Unable to send recessive bit.

14.6.2.4.5 _flexcan_flags

```
enum _flexcan_flags
#include <fsl_flexcan.h>
FlexCAN status flags.
```

This provides constants for the FlexCAN status flags for use in the FlexCAN functions.

Note

The CPU read action clears the bits corresponding to the FLEXCAN_ErrorFlag macro, therefore user need to read status flags and distinguish which error is occur using [_flexcan_error_flags](#) enumerations.

Enumerator

kFLEXCAN_SynchFlag	CAN Synchronization Status.
kFLEXCAN_TxWarningIntFlag	Tx Warning Interrupt Flag.
kFLEXCAN_RxWarningIntFlag	Rx Warning Interrupt Flag.
kFLEXCAN_IdleFlag	FlexCAN In IDLE Status.
kFLEXCAN_FaultConfinementFlag	FlexCAN Fault Confinement State.
kFLEXCAN_TransmittingFlag	FlexCAN In Transmission Status.
kFLEXCAN_ReceivingFlag	FlexCAN In Reception Status.
kFLEXCAN_BusOffIntFlag	Bus Off Interrupt Flag.
kFLEXCAN_ErrorIntFlag	CAN Error Interrupt Flag.
kFLEXCAN_WakeUpIntFlag	Self Wake-Up Interrupt Flag.

14.6.2.4.6 _flexcan_frame_format

```
enum _flexcan_frame_format
#include <fsl_flexcan.h>
FlexCAN frame format.
```

Enumerator

kFLEXCAN_FrameFormatStandard	Standard frame format attribute.
kFLEXCAN_FrameFormatExtend	Extend frame format attribute.

14.6.2.4.7 _flexcan_frame_type

```
enum _flexcan_frame_type
#include <fsl_flexcan.h>
FlexCAN frame type.
```

Enumerator

kFLEXCAN_FrameTypeData	Data frame type attribute.
kFLEXCAN_FrameTypeRemote	Remote frame type attribute.

14.6.2.4.8 _flexcan_interrupt_enable

```
enum _flexcan_interrupt_enable
#include <fsl_flexcan.h>
FlexCAN interrupt enable enumerations.
This provides constants for the FlexCAN interrupt enable enumerations for use in the FlexCAN functions.
```

Note

FlexCAN Message Buffers and Legacy Rx FIFO interrupts not included in.

Enumerator

kFLEXCAN_BusOffInterruptEnable	Bus Off interrupt, use bit 15.
kFLEXCAN_ErrorInterruptEnable	CAN Error interrupt, use bit 14.
kFLEXCAN_TxWarningInterruptEnable	Tx Warning interrupt, use bit 11.
kFLEXCAN_RxWarningInterruptEnable	Rx Warning interrupt, use bit 10.
kFLEXCAN_WakeUpInterruptEnable	Self Wake Up interrupt, use bit 26.

14.6.2.4.9 _flexcan_rx_fifo_filter_type

```
enum _flexcan_rx_fifo_filter_type
#include <fsl_flexcan.h>
FlexCAN Rx Fifo Filter type.
```

Enumerator

kFLEXCAN_RxFifoFilterTypeA	One full ID (standard and extended) per ID Filter element.
kFLEXCAN_RxFifoFilterTypeB	Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.
kFLEXCAN_RxFifoFilterTypeC	Four partial 8-bit Standard or extended ID slices per ID Filter Table element.
kFLEXCAN_RxFifoFilterTypeD	All frames rejected.

14.6.2.4.10 _flexcan_rx_fifo_priority

```
enum _flexcan_rx_fifo_priority
#include <fsl_flexcan.h>
```

FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

Enumerator

kFLEXCAN_RxFifoPrioLow	Matching process start from Rx Message Buffer first.
kFLEXCAN_RxFifoPrioHigh	Matching process start from Enhanced/Legacy Rx FIFO first.

14.6.2.4.11 _flexcan_wake_up_source

```
enum _flexcan_wake_up_source
#include <fsl_flexcan.h>
```

FlexCAN wake up source.

Enumerator

kFLEXCAN_WakeupSrcUnfiltered	FlexCAN uses unfiltered Rx input to detect edge.
kFLEXCAN_WakeupSrcFiltered	FlexCAN uses filtered Rx input to detect edge.

14.6.2.5 Function Documentation

14.6.2.5.1 FLEXCAN_CalculateImprovedTimingValues()

```
bool FLEXCAN_CalculateImprovedTimingValues (
    CAN_Type * base,
    uint32_t bitRate,
    uint32_t sourceClock_Hz,
    flexcan_timing_config_t * pTimingConfig)
#include <fsl_flexcan.h>
```

Calculates the improved timing values by specific bit Rates for classical CAN.

This function use to calculates the Classical CAN timing values according to the given bit rate. The Calculated timing values will be set in CTRL1/CBT/ENCBT register. The calculation is based on the recommendation of the CiA 301 v4.2.0 and previous version document.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>bitRate</i>	The classical CAN speed in bps defined by user, should be less than or equal to 1Mbps.
<i>sourceClock_Hz</i>	The Source clock frequency in Hz.
<i>pTimingConfig</i>	Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration.

14.6.2.5.2 FLEXCAN_Deinit()

```
void FLEXCAN_Deinit (  
    CAN_Type * base)  
#include <fsl_flexcan.h>
```

De-initializes a FlexCAN instance.

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

<i>base</i>	FlexCAN peripheral base address.

14.6.2.5.3 FLEXCAN_EnterFreezeMode()

```
void FLEXCAN_EnterFreezeMode (
    CAN_Type * base)
#include <fsl_flexcan.h>
```

Enter FlexCAN Freeze Mode.

This function makes the FlexCAN work under Freeze Mode.

Parameters

<i>base</i>	FlexCAN peripheral base address.

14.6.2.5.4 FLEXCAN_ExitFreezeMode()

```
void FLEXCAN_ExitFreezeMode (
    CAN_Type * base)
#include <fsl_flexcan.h>
```

Exit FlexCAN Freeze Mode.

This function makes the FlexCAN leave Freeze Mode.

Parameters

<i>base</i>	FlexCAN peripheral base address.

14.6.2.5.5 FLEXCAN_GetDefaultConfig()

```
void FLEXCAN_GetDefaultConfig (
    flexcan_config_t * pConfig)
#include <fsl_flexcan.h>
```

Gets the default configuration structure.

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. flexcanConfig->clkSrc = kFLEXCAN_ClkSrc0; flexcanConfig->bitRate = 1000000U; flexcanConfig->bitRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false; flexcanConfig->enableDoze = false; flexcanConfig->enableMemoryErrorControl = true; flexcanConfig->enableNonCorrectableErrorEnterFreeze = true; flexcanConfig.timingConfig = timingConfig;

Parameters

<i>pConfig</i>	Pointer to the FlexCAN configuration structure.

14.6.2.5.6 FLEXCAN_GetInstance()

```
uint32_t FLEXCAN_GetInstance (
    CAN_Type * base)
#include <fsl_flexcan.h>
```

Get the FlexCAN instance from peripheral base address.

Parameters

<i>base</i>	FlexCAN peripheral base address.

Returns

FlexCAN instance.

14.6.2.5.7 FLEXCAN_GetTimeStamp()

```
uint32_t FLEXCAN_GetTimeStamp (
    flexcan_handle_t * handle,
    uint8_t mbIdx)
#include <fsl_flexcan.h>
```

Gets the detail index of Mailbox's Timestamp by handle.

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN_TransferSendNonBlocking -FLEXCAN_TransferFDSendNonBlocking -FLEXCAN_TransferReceiveNonBlocking -FLEXCAN_TransferFDReceiveNonBlocking -FLEXCAN_TransferReceiveFifoNonBlocking

Parameters

<i>handle</i>	FlexCAN handle pointer.
<i>mbIdx</i>	The FlexCAN Message Buffer index.

Return values

<i>the</i>	index of mailbox 's timestamp stored in the handle.

14.6.2.5.8 FLEXCAN_Init()

```
void FLEXCAN_Init (
    CAN_Type * base,
    const flexcan_config_t * pConfig,
    uint32_t sourceClock_Hz)
#include <fsl_flexcan.h>
```

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_Init function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc      = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate     = 1000000U;
flexcanConfig.maxMbNum    = 16;
flexcanConfig.enableLoopBack = false;
flexcanConfig.enableSelfWakeup = false;
flexcanConfig.enableIndividMask = false;
flexcanConfig.enableDoze     = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.timingConfig  = timingConfig;
FLEXCAN_Init(CAN0, &flexcanConfig, 40000000UL);
```

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>pConfig</i>	Pointer to the user-defined configuration structure.
<i>sourceClock_Hz</i>	FlexCAN Protocol Engine clock source frequency in Hz.

14.6.2.5.9 FLEXCAN_ReadRxFifo()

```
status_t FLEXCAN_ReadRxFifo (
    CAN_Type * base,
    flexcan_frame_t * pRxFrame)
```

```
#include <fsl_flexcan.h>
```

Reads a FlexCAN Message from Legacy Rx FIFO.

This function reads a CAN message from the FlexCAN Legacy Rx FIFO.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>pRxFrame</i>	Pointer to CAN message frame structure for reception.

Return values

<i>kStatus_Success</i>	- Read Message from Rx FIFO successfully.
<i>kStatus_Fail</i>	- Rx FIFO is not enabled.

14.6.2.5.10 FLEXCAN_ReadRxMb()

```
status_t FLEXCAN_ReadRxMb (
    CAN_Type * base,
    uint8_t mbIdx,
    flexcan_frame_t * pRxFrame)
```

```
#include <fsl_flexcan.h>
```

Reads a FlexCAN Message from Receive Message Buffer.

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>mbIdx</i>	The FlexCAN Message Buffer index.
<i>pRxFrame</i>	Pointer to CAN message frame structure for reception.

Return values

<i>kStatus_Success</i>	- Rx Message Buffer is full and has been read successfully.
<i>kStatus_FLEXCAN_RxOverflow</i>	- Rx Message Buffer is already overflowed and has been read successfully.
<i>kStatus_Fail</i>	- Rx Message Buffer is empty.

14.6.2.5.11 FLEXCAN_SetBitRate()

```
status_t FLEXCAN_SetBitRate (
    CAN_Type * base,
    uint32_t sourceClock_Hz,
    uint32_t bitRate_Bps)
```

```
#include <fsl_flexcan.h>
```

Set bit rate of FlexCAN classical CAN frame or CAN FD frame nominal phase.

This function set the bit rate of classical CAN frame or CAN FD frame nominal phase base on [FLEXCAN_CalculateImprovedTimingValue](#) API calculated timing values.

Note

Calling [FLEXCAN_SetBitRate\(\)](#) overrides the bit rate set in [FLEXCAN_Init\(\)](#).

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>sourceClock_Hz</i>	Source Clock in Hz.
<i>bitRate_Bps</i>	Bit rate in Bps.

Returns

kStatus_Success - Set CAN baud rate (only Nominal phase) successfully.

14.6.2.5.12 FLEXCAN_SetRxFifoConfig()

```
void FLEXCAN_SetRxFifoConfig (
    CAN_Type * base,
    const flexcan_rx_fifo_config_t * pRxFifoConfig,
    bool enable)
```

```
#include <fsl_flexcan.h>
```

Configures the FlexCAN Legacy Rx FIFO.

This function configures the FlexCAN Rx FIFO with given configuration.

Note

Legacy Rx FIFO only can receive classic CAN message.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>pRxFifoConfig</i>	Pointer to the FlexCAN Legacy Rx FIFO configuration structure. Can be NULL when enable parameter is false.
<i>enable</i>	Enable/disable Legacy Rx FIFO. • true: Enable Legacy Rx FIFO. • false: Disable Legacy Rx FIFO.

14.6.2.5.13 FLEXCAN_SetRxFifoGlobalMask()

```
void FLEXCAN_SetRxFifoGlobalMask (
    CAN_Type * base,
    uint32_t mask)
```

```
#include <fsl_flexcan.h>
```

Sets the FlexCAN receive FIFO global mask.

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>mask</i>	Rx Fifo Global Mask value.

14.6.2.5.14 FLEXCAN_SetRxIndividualMask()

```
void FLEXCAN_SetRxIndividualMask (
    CAN_Type * base,
    uint8_t maskIdx,
    uint32_t mask)
#include <fsl_flexcan.h>
```

Sets the FlexCAN receive individual mask.

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the [FLEXCAN_Init\(\)](#). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>maskIdx</i>	The Index of individual Mask.
<i>mask</i>	Rx Individual Mask value.

14.6.2.5.15 FLEXCAN_SetRxMbConfig()

```
void FLEXCAN_SetRxMbConfig (
    CAN_Type * base,
    uint8_t mbIdx,
    const flexcan_rx_mb_config_t * pRxMbConfig,
    bool enable)
#include <fsl_flexcan.h>
```

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>mbIdx</i>	The Message Buffer index.
<i>pRxMbConfig</i>	Pointer to the FlexCAN Message Buffer configuration structure.
<i>enable</i>	Enable/disable Rx Message Buffer. • true: Enable Rx Message Buffer. • false: Disable Rx Message Buffer.

14.6.2.5.16 FLEXCAN_SetRxMbGlobalMask()

```
void FLEXCAN_SetRxMbGlobalMask (
    CAN_Type * base,
    uint32_t mask)
#include <fsl_flexcan.h>
```

Sets the FlexCAN receive message buffer global mask.

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the [FLEXCAN_Init\(\)](#).

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>mask</i>	Rx Message Buffer Global Mask value.

14.6.2.5.17 FLEXCAN_SetTimingConfig()

```
void FLEXCAN_SetTimingConfig (
    CAN_Type * base,
    const flexcan_timing_config_t * pConfig)
```

```
#include <fsl_flexcan.h>
```

Sets the FlexCAN classical CAN protocol timing characteristic.

This function gives user settings to classical CAN or CAN FD nominal phase timing characteristic. The function is for an experienced user. For less experienced users, call the [FLEXCAN_SetBitRate\(\)](#) instead.

Note

Calling [FLEXCAN_SetTimingConfig\(\)](#) overrides the bit rate set in [FLEXCAN_Init\(\)](#) or [FLEXCAN_SetBitRate\(\)](#).

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>pConfig</i>	Pointer to the timing configuration structure.

14.6.2.5.18 FLEXCAN_SetTxMbConfig()

```
void FLEXCAN_SetTxMbConfig (
    CAN_Type * base,
    uint8_t mbIdx,
    bool enable)
```

```
#include <fsl_flexcan.h>
```

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>mbIdx</i>	The Message Buffer index.
<i>enable</i>	Enable/disable Tx Message Buffer. • true: Enable Tx Message Buffer. • false: Disable Tx Message Buffer.

14.6.2.5.19 FLEXCAN_TransferAbortReceive()

```
void FLEXCAN_TransferAbortReceive (
    CAN_Type * base,
    flexcan_handle_t * handle,
    uint8_t mbIdx)
```

```
#include <fsl_flexcan.h>
```

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	FlexCAN handle pointer.
<i>mbIdx</i>	The FlexCAN Message Buffer index.

14.6.2.5.20 FLEXCAN_TransferAbortReceiveFifo()

```
void FLEXCAN_TransferAbortReceiveFifo (
    CAN_Type * base,
    flexcan_handle_t * handle)
```

```
#include <fsl_flexcan.h>
```

Aborts the interrupt driven message receive from Rx FIFO process.

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	FlexCAN handle pointer.

14.6.2.5.21 FLEXCAN_TransferAbortSend()

```
void FLEXCAN_TransferAbortSend (
    CAN_Type * base,
    flexcan_handle_t * handle,
    uint8_t mbIdx)
```

```
#include <fsl_flexcan.h>
```

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	FlexCAN handle pointer.
<i>mbIdx</i>	The FlexCAN Message Buffer index.

14.6.2.5.22 FLEXCAN_TransferCreateHandle()

```
void FLEXCAN_TransferCreateHandle (
    CAN_Type * base,
    flexcan_handle_t * handle,
    flexcan_transfer_callback_t callback,
    void * userData)
```

```
#include <fsl_flexcan.h>
```

Initializes the FlexCAN handle.

This function initializes the FlexCAN handle, which can be used for other FlexCAN transactional APIs. Usually, for a specified FlexCAN instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	FlexCAN handle pointer.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.

14.6.2.5.23 FLEXCAN_TransferGetReceiveFifoCount()

```
status_t FLEXCAN_TransferGetReceiveFifoCount (
    CAN_Type * base,
    flexcan_handle_t * handle,
    size_t * count)
```

```
#include <fsl_flexcan.h>
```

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	FlexCAN handle pointer.
<i>count</i>	Number of CAN messages receive so far by the non-blocking transaction.

Return values

<i>kStatus_InvalidArgument</i>	count is Invalid.
<i>kStatus_Success</i>	Successfully return the count.

14.6.2.5.24 FLEXCAN_TransferHandleIRQ()

```
void FLEXCAN_TransferHandleIRQ (
    CAN_Type * base,
    flexcan_handle_t * handle)
```

```
#include <fsl_flexcan.h>
```

FlexCAN IRQ handle function.

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	FlexCAN handle pointer.

14.6.2.5.25 FLEXCAN_TransferReceiveBlocking()

```
status_t FLEXCAN_TransferReceiveBlocking (
    CAN_Type * base,
    uint8_t mbIdx,
    flexcan_frame_t * pRxFrame)
```

```
#include <fsl_flexcan.h>
```

Performs a polling receive transaction on the CAN bus.

Note

A transfer handle does not need to be created before calling this API.

Parameters

<i>base</i>	FlexCAN peripheral base pointer.
<i>mbIdx</i>	The FlexCAN Message Buffer index.
<i>pRxFrame</i>	Pointer to CAN message frame structure for reception.

Return values

<i>kStatus_Success</i>	- Rx Message Buffer is full and has been read successfully.
<i>kStatus_FLEXCAN_RxOverflow</i>	- Rx Message Buffer is already overflowed and has been read successfully.
<i>kStatus_Fail</i>	- Rx Message Buffer is empty.

14.6.2.5.26 FLEXCAN_TransferReceiveFifoBlocking()

```
status_t FLEXCAN_TransferReceiveFifoBlocking (
    CAN_Type * base,
    flexcan_frame_t * pRxFrame)
```

```
#include <fsl_flexcan.h>
```

Performs a polling receive transaction from Legacy Rx FIFO on the CAN bus.

Note

A transfer handle does not need to be created before calling this API.

Parameters

<i>base</i>	FlexCAN peripheral base pointer.
<i>pRxFrame</i>	Pointer to CAN message frame structure for reception.

Return values

<i>kStatus_Success</i>	- Read Message from Rx FIFO successfully.
<i>kStatus_Fail</i>	- Rx FIFO is not enabled.

14.6.2.5.27 FLEXCAN_TransferReceiveFifoNonBlocking()

```
status_t FLEXCAN_TransferReceiveFifoNonBlocking (
    CAN_Type * base,
    flexcan_handle_t * handle,
    flexcan_fifo_transfer_t * pFifoXfer)
```

```
#include <fsl_flexcan.h>
```

Receives a message from Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	FlexCAN handle pointer.
<i>pFifoXfer</i>	FlexCAN Rx FIFO transfer structure. See the flexcan_fifo_transfer_t .

Return values

<i>kStatus_Success</i>	- Start Rx FIFO receiving process successfully.
<i>kStatus_FLEXCAN_RxFifoBusy</i>	- Rx FIFO is currently in use.

14.6.2.5.28 FLEXCAN_TransferReceiveNonBlocking()

```
status_t FLEXCAN_TransferReceiveNonBlocking (
    CAN_Type * base,
    flexcan_handle_t * handle,
    flexcan_mb_transfer_t * pMbXfer)
```

```
#include <fsl_flexcan.h>
```

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	FlexCAN handle pointer.
<i>pMbXfer</i>	FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t .

Return values

<i>kStatus_Success</i>	- Start Rx Message Buffer receiving process successfully.
<i>kStatus_FLEXCAN_RxBusy</i>	- Rx Message Buffer is in use.

14.6.2.5.29 FLEXCAN_TransferSendBlocking()

```
status_t FLEXCAN_TransferSendBlocking (
    CAN_Type * base,
    uint8_t mbIdx,
    flexcan_frame_t * pTxFrame)
```

```
#include <fsl_flexcan.h>
```

Performs a polling send transaction on the CAN bus.

Note

A transfer handle does not need to be created before calling this API.

Parameters

<i>base</i>	FlexCAN peripheral base pointer.
<i>mbIdx</i>	The FlexCAN Message Buffer index.
<i>pTxFrame</i>	Pointer to CAN message frame to be sent.

Return values

<i>kStatus_Success</i>	- Write Tx Message Buffer Successfully.
<i>kStatus_Fail</i>	- Tx Message Buffer is currently in use.

14.6.2.5.30 FLEXCAN_TransferSendNonBlocking()

```
status_t FLEXCAN_TransferSendNonBlocking (
    CAN_Type * base,
    flexcan_handle_t * handle,
    flexcan_mb_transfer_t * pMbXfer)
```

```
#include <fsl_flexcan.h>
```

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	FlexCAN handle pointer.
<i>pMbXfer</i>	FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t .

Return values

<i>kStatus_Success</i>	Start Tx Message Buffer sending process successfully.
<i>kStatus_Fail</i>	Write Tx Message Buffer failed.
<i>kStatus_FLEXCAN_TxBusy</i>	Tx Message Buffer is in use.

14.6.2.5.31 FLEXCAN_WriteTxMb()

```
status_t FLEXCAN_WriteTxMb (
    CAN_Type * base,
    uint8_t mbIdx,
    const flexcan_frame_t * pTxFrame)
```

```
#include <fsl_flexcan.h>
```

Writes a FlexCAN Message to the Transmit Message Buffer.

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>mbIdx</i>	The FlexCAN Message Buffer index.
<i>pTxFrame</i>	Pointer to CAN message frame to be sent.

Return values

<i>kStatus_Success</i>	- Write Tx Message Buffer Successfully.
<i>kStatus_Fail</i>	- Tx Message Buffer is currently in use.

14.6.3 MOD5441x, NANO54415, SB800EX

Classes

- class [canMCF5441x::CanRxMessage](#)
Class to hold received CAN messages.

14.6.3.1 Detailed Description

- `#include< multican.h>`

Control Area Network (CAN) for ColdFire MCF5441x platforms including MOD54415, MOD54417, NANO54415 and SB800EX.

14.6.4 MODM7AE70

Topics

- [MCAN Driver Internals](#)

Classes

- class [mcanMODM7AE70::mcan_module](#)
MCAN Module Class.
- class [mcanMODM7AE70::mcan_config](#)

MCAN configuration structure.

- class [mcanMODM7AE70::CanRxMessage](#)

Class to hold received CAN messages.

MCAN Constants

- const uint32_t **mcanMODM7AE70::CONF_MCAN_RX_FIFO_0_NUM** = 32
Range: 1..64.
- const uint32_t **mcanMODM7AE70::CONF_MCAN_RX_FIFO_1_NUM** = 1
Range: 1..64.
- const uint32_t **mcanMODM7AE70::CONF_MCAN_RX_BUFFER_NUM** = 1
Range: 1..64.
- const uint32_t **mcanMODM7AE70::CONF_MCAN_TX_BUFFER_NUM** = 8
Range: 1..16.
- const uint32_t **mcanMODM7AE70::CONF_MCAN_TX_FIFO_QUEUE_NUM** = 1
Range: 1..16.
- const uint32_t **mcanMODM7AE70::CONF_MCAN_TX_EVENT_FIFO** = 8
Range: 1..32.
- const uint32_t **mcanMODM7AE70::CONF_MCAN_RX_STANDARD_ID_FILTER_NUM** = 32
Range: 1..128.
- const uint32_t **mcanMODM7AE70::CONF_MCAN_RX_EXTENDED_ID_FILTER_NUM** = 32
Range: 1..64.
- #define **CAN_DATA_STORE_SIZE** (512)
Receive OS_FIFO Buffer Size.

Helper Macros

- #define **CAN_EXTENDED_ID_BIT** (0x20000000)
The single bit used by this API to indicate an extended ID.
- #define **ExtToNbId**(id) (id | CAN_EXTENDED_ID_BIT)
Convert an ID to an extended ID.
- #define **NormToNbId**(id) (id & 0x7ff)
Convert a normal ID in the range of 0 to 2048 to the range of 0 to 1024.
- #define **IsNbIdExt**(id) ((id & CAN_EXTENDED_ID_BIT) != 0)
Check for an extended MCAN ID.
- #define **NbToExtId**(id) (id & 0x1FFFFFFF)
Remove the API extended flage from the ID.
- #define **NbToNormId**(id) (id & 0x7FF)
Shift a Normal ID so it has a value from 0 to 1023.

14.6.4.1 Detailed Description

```
#include< mcan.h>
```

Control Area Network (MCAN) Low Level Driver for MODM7AE70

The Controller Area Network (MCAN) performs communication according to ISO 11898-1:2015 and to Bosch CANFD specification. Additional transceiver hardware is required for connection to the physical layer. All functions concerning the handling of messages are implemented by the Rx Handler and the Tx Handler. The Rx Handler manages message acceptance filtering, the transfer of received messages from the CAN core to the Message RAM, as well as providing receive message status information. The Tx Handler is responsible for the transfer of transmit messages from the Message RAM to the CAN core, as well as providing transmit status information.

Acceptance filtering is implemented by a combination of up to 128 filter elements, where each element can be configured as a range, as a bit mask, or as a dedicated ID filter.

There are two types of CAN identifiers that define different formats of the message frame, with the main difference being the identifier length. The two CAN protocol versions are:

- Version 2.0A, Normal or Standard CAN, supporting messages with 11-bit identifiers.
- Version 2.0B, Extended CAN, supporting messages with 29-bit identifiers consisting of an 11-bit base identifier for compatibility with Version 2.0A and an 18-bit extension identifier.

Note

Detailed information on the MCAN operation and features is located in the Microchip SAME70 Family Data Sheet located in your \nburn\docs\Microchip folder, chapter 49.

14.6.4.2 Macro Definition Documentation

14.6.4.2.1 IsNBIdExt

```
#define IsNBIdExt(  
    id) ( (id & CAN_EXTENDED_ID_BIT) !=0)
```

#include <mcan.h>
Check for an extended MCAN ID.
Returns non zero if the ID is an extended ID

14.6.4.2.2 NbToNormId

```
#define NbToNormId(  
    id) (id & 0x7FF)
```

#include <mcan.h>
Shift a Normal ID so it has a value from 0 to 1023.
Some CAN systems will treat normal ID's as an integer from 0 to 2048. Other systems may treat normal ID's as 28 bit values where the bottom 17 bits are zero. This macro will convert the NetBurner normal ID format to the 0 to 2048 format.

14.6.4.3 MCAN Driver Internals

Module Setting

- enum `mcan_timeout_mode` { `MCAN_TIMEOUT_CONTINUOUS` = `MCAN_TOCC_TOS_CONTINUOUS` , `MCAN_TIMEOUT_TX_EVEN_FIFO` = `MCAN_TOCC_TOS_TX_EV_TIMEOUT` , `MCAN_TIMEOUT_RX_FIFO_0` = `MCAN_TOCC_TOS_RX0_EV_TIMEOUT` , `MCAN_TIMEOUT_RX_FIFO_1` = `MCAN_TOCC_TOS_RX1_EV_TIMEOUT` }
- *Can time out modes.*
- enum `mcan_nonmatching_frames_action` { `MCAN_NONMATCHING_FRAMES_FIFO_0` , `MCAN_NONMATCHING_FRAMES_REJECT` }
- *Can non-matching frames action.*
- enum `mcan_interrupt_source` { `MCAN_RX_FIFO_0_NEW_MESSAGE` = `MCAN_IE_RF0NE` , `MCAN_RX_FIFO_0_WATERMARK` = `MCAN_IE_RF0WE` , `MCAN_RX_FIFO_0_FULL` = `MCAN_IE_RF0FE` , `MCAN_RX_FIFO_0_LOST_MESSAGE` = `MCAN_IE_RF0LE` , `MCAN_RX_FIFO_1_NEW_MESSAGE` = `MCAN_IE_RF1NE` , `MCAN_RX_FIFO_1_WATERMARK` = `MCAN_IE_RF1WE` , `MCAN_RX_FIFO_1_FULL` = `MCAN_IE_RF1FE` , `MCAN_RX_FIFO_1_MESSAGE_LOST` = `MCAN_IE_RF1LE` , `MCAN_RX_HIGH_PRIORITY_MESSAGE` = `MCAN_IE_HPME` , `MCAN_TIMESTAMP_COMPLETE` = `MCAN_IE_TCE` , `MCAN_TX_CANCELLATION_FINISH` = `MCAN_IE_TCFE` , `MCAN_TX_FIFO_EMPTY` = `MCAN_IE_TFEE` , `MCAN_TX_EVENT_FIFO_NEW_ENTRY` = `MCAN_IE_TEFNE` , `MCAN_TX_EVENT_FIFO_WATERMARK` = `MCAN_IE_TEFWE` , `MCAN_TX_EVENT_FIFO_FULL` = `MCAN_IE_TEFWE` , `MCAN_TX_EVENT_FIFO_ELEMENT_LOST` = `MCAN_IE_TEFLE` , `MCAN_TIMESTAMP_WRAPAROUND` = `MCAN_IE_TSWE` , `MCAN_MESSAGE_RAM_ACCESS_FAILURE` = `MCAN_IE_MRAFE` , `MCAN_TIMEOUT_OCCURRED` = `MCAN_IE_TOOE` , `MCAN_RX_BUFFER_NEW_MESSAGE` = `MCAN_IE_DRXE` , `MCAN_ERROR_LOGGING_OVERFLOW` = `MCAN_IE_ELOE` , `MCAN_ERROR_PASSIVE` = `MCAN_IE_EPE` , `MCAN_WARNING_STATUS` = `MCAN_IE_EWE` , `MCAN_BUS_OFF` = `MCAN_IE_BOE` ,

```
MCAN_WATCHDOG = MCAN_IE_WDIE , MCAN_CRC_ERROR = MCAN_IE_CRCEE , MCAN_BIT_ERROR
= MCAN_IE_BEE , MCAN_ACKNOWLEDGE_ERROR = MCAN_IE_ACKEE ,
MCAN_FORMAT_ERROR = MCAN_IE_FOEE , MCAN_STUFF_ERROR = MCAN_IE_STEE }
```

Can module interrupt source.

14.6.4.3.1 Detailed Description

```
#include< mcan_internal.h>
```

This driver for Atmel® | SMART SAM devices provides an low level interface for the configuration and management of the device's Control Area Network functionality.

Note

Since "The Control Area Network (CAN) performs communication according to ISO 11898-1 (Bosch CAN specification 2.0 part A,B) and to Bosch CAN FD specification V1.0", the driver is focus on the MAC layer and try to offer the APIs which can be used by upper application layer.

For storage of Rx/Tx messages and for storage of the filter configuration, a message RAM is needed to the CAN module. In this driver, the message RAM is static allocated, the related setting is defined and can be changed in the module configuration file "conf_mcan.h".

The following peripherals are used by this module:

- CAN (Control Area Network)

The following devices can use this module:

- SAMV71

The outline of this documentation is as follows:

- [Prerequisites](#)
- [Module Overview](#)
- [Special Considerations](#)
- [Extra Information](#)
- [Examples](#)
- [API Overview](#)

14.6.4.3.2 Prerequisites

There are no prerequisites for this module.

14.6.4.3.3 Module Overview

This driver provides an interface for the Control Area Network Controller functions on the device.

14.6.4.3.4 Special Considerations

There are no special considerations for this module.

14.6.4.3.5 Extra Information

For extra information see asfdoc_sam_mcan_extra. This includes:

- asfdoc_sam_mcan_extra_acronyms
- asfdoc_sam_mcan_extra_dependencies
- asfdoc_sam_mcan_extra_errata
- asfdoc_sam_mcan_extra_history

14.6.4.3.6 Examples

For a list of examples related to this driver, see asfdoc_sam_mcan_exqsg.

14.6.4.3.7 API Overview

14.6.4.3.8 Enumeration Type Documentation

14.6.4.3.8.1 mcan_interrupt_source

```
enum mcan_interrupt_source
#include <mcan_internal.h>
Can module interrupt source.
Enum for the interrupt source.
```

Enumerator

MCAN_RX_FIFO_0_NEW_MESSAGE	Rx FIFO 0 New Message Interrupt Enable.
MCAN_RX_FIFO_0_WATERMARK	Rx FIFO 0 Watermark Reached Interrupt Enable.
MCAN_RX_FIFO_0_FULL	Rx FIFO 0 Full Interrupt Enable.
MCAN_RX_FIFO_0_LOST_MESSAGE	Rx FIFO 0 Message Lost Interrupt Enable.
MCAN_RX_FIFO_1_NEW_MESSAGE	Rx FIFO 1 New Message Interrupt Enable.
MCAN_RX_FIFO_1_WATERMARK	Rx FIFO 1 Watermark Reached Interrupt Enable.
MCAN_RX_FIFO_1_FULL	Rx FIFO 1 Full Interrupt Enable.
MCAN_RX_FIFO_1_MESSAGE_LOST	Rx FIFO 1 Message Lost Interrupt Enable.
MCAN_RX_HIGH_PRIORITY_MESSAGE	High Priority Message Interrupt Enable.
MCAN_TIMESTAMP_COMPLETE	Transmission Completed Interrupt Enable.
MCAN_TX_CANCELLATION_FINISH	Transmission Cancellation Finished Interrupt Enable.
MCAN_TX_FIFO_EMPTY	Tx FIFO Empty Interrupt Enable.
MCAN_TX_EVENT_FIFO_NEW_ENTRY	Tx Event FIFO New Entry Interrupt Enable.
MCAN_TX_EVENT_FIFO_WATERMARK	Tx Event FIFO Watermark Reached Interrupt Enable.
MCAN_TX_EVENT_FIFO_FULL	Tx Event FIFO Full Interrupt Enable.
MCAN_TX_EVENT_FIFO_ELEMENT_LOST	Tx Event FIFO Element Lost Interrupt Enable.
MCAN_TIMESTAMP_WRAPAROUND	Timestamp Wraparound Interrupt Enable.
MCAN_MESSAGE_RAM_ACCESS_FAILURE	Message RAM Access Failure Interrupt Enable.
MCAN_TIMEOUT_OCCURRED	Timeout Occurred Interrupt Enable.
MCAN_RX_BUFFER_NEW_MESSAGE	Message stored to Dedicated Rx Buffer Interrupt Enable.
MCAN_ERROR_LOGGING_OVERFLOW	Error Logging Overflow Interrupt Enable.
MCAN_ERROR_PASSIVE	Error Passive Interrupt Enable.
MCAN_WARNING_STATUS	Warning Status Interrupt Enable.
MCAN_BUS_OFF	Bus_Off Status Interrupt Enable.
MCAN_WATCHDOG	Watchdog Interrupt Enable.
MCAN_CRC_ERROR	CRC Error Interrupt Enable
MCAN_BIT_ERROR	Bit Error Interrupt Enable.
MCAN_ACKNOWLEDGE_ERROR	Acknowledge Error Interrupt Enable .
MCAN_FORMAT_ERROR	Format Error Interrupt Enable
MCAN_STUFF_ERROR	Stuff Error Interrupt Enable

14.6.4.3.8.2 mcan_nonmatching_frames_action

```
enum mcan_nonmatching_frames_action
#include <mcan_internal.h>
```

Can non-matching frames action.

Enumerator

MCAN_NONMATCHING_FRAMES_FIFO_0	Accept in Rx FIFO 0.
MCAN_NONMATCHING_FRAMES_FIFO_1	Accept in Rx FIFO 1.
MCAN_NONMATCHING_FRAMES_REJECT	Reject.

14.6.4.3.8.3 mcan_timeout_mode

```
enum mcan_timeout_mode
#include <mcan_internal.h>
Can time out modes.
```

Enumerator

MCAN_TIMEOUT_CONTINUES	Continuous operation.
MCAN_TIMEOUT_TX_EVEN_FIFO	Timeout controlled by TX Event FIFO.
MCAN_TIMEOUT_RX_FIFO_0	Timeout controlled by Rx FIFO 0.
MCAN_TIMEOUT_RX_FIFO_1	Timeout controlled by Rx FIFO 1.

14.7 Command Processor

Topics

- [Command Processor Disconnect Causes](#)
- [Command Processor Listen Channels](#)
- [Command Processor Response Codes](#)

Functions

- int [CmdStartCommandProcessor](#) (int priority)
Start the command processor.
- int [CmdAddCommandFd](#) (int fd, bool require_auth, bool time_out_conn, bool local_echo=true)
Add an established FD connection to the list of fd's managed by the command processor.
- int [CmdRemoveCommandFd](#) (int fd)
Remove an established FD (either a TCP session, a serial connection, or an SSH session).
- int [CmdListenOnTcpPort](#) (uint16_t port, int do_telnet_processing, int max_connections)
Start listening for a connection over TCP.
- int [CmdListenQuietOnTcpPort](#) (uint16_t port, int do_telnet_processing, int max_connections)
Start listening for a connection over TCP, but without the siggnon or password.
- int [CmdListenOnSshPort](#) (uint16_t port, int max_connections)
Start listening for a connection over SSH.
- int [CmdListenQuietOnSshPort](#) (uint16_t port, int max_connections)
Start listening for a connection over SSH, but without the siggnon or password.
- int [CmdStopListeningOnTcpPort](#) (uint16_t port)
Stop Listening for connections on the specified port. Also closes all open connections that were based on that port.
- int [CmdStopListeningOnSshPort](#) (uint16_t port)
Stop Listening for connections on the specified port. Also closes all open connections that were based on that port.
- void [SendToAll](#) (const char *buffer, int len, bool include_serial_ports)
Send a message to all connected sockets, excluding "Listening sockets".

Variables

- `int(* CmdAuthenticateFunc )(const char *name, const char *passwd)`
External Authentication function CALLBACK for TCP connections, used to verify username and password. If this function pointer is not NULL then each new Telnet session will be asked to authenticate.
- `int(* CmdAuthenticateSshFunc )(const char *name, const char *authVal, AuthType authType)`
External Authentication function CALLBACK for SSH connections, used to verify username and password. If this function pointer is not NULL then each new SSH session will be asked to authenticate. If this function pointer is NULL, it tries to use CmdAuthenticateFunc instead.
- `int(* CmdCmd_func )(const char *command, FILE *fRespondto, void *pData)`
The command processing callback function for handling string commands.
- `int(* CmdChar_func )(char command, FILE *fRespondto, void *pData)`
The command processing callback function for handling single character commands. If this is implemented does not do echo or line editing this is the responsibility of the application programmer.
- `void (* CmdConnect_func )(FILE *fRespondto)`
Connect callback function. If this function is not NULL, then the system will call this function every time a new session is started.
- `void(* CmdPrompt_func )(FILE *fRespondto, void *pData)`
Prompt callback function. If this function is not NULL, then the system will call this function every time a new prompt line needs to be displayed.
- `void(* CmdDisConnect_func )(FILE *fRespondto, int cause, void *pData)`
Dis-Connect callback function, if this function is not NULL then the system will call this function every time a session is terminated.
- `int CmdIdleTimeout`
- `const char * Cmdlogin_prompt`
If this is defined, then it will be sent to the socket on connection before Authentication is tried.

14.7.1 Detailed Description

```
#include< command.h>
```

This is NetBurner's Command Processor Library. It operates as a task and reads input one line at a time from multiple sources. When it has gathered a whole line, it sends it to a command processor that is defined by the user.

14.7.2 Function Documentation

14.7.2.1 CmdAddCommandFd()

```
int CmdAddCommandFd (
    int fd,
    bool require_auth,
    bool time_out_conn,
    bool local_echo = true)
#include <command.h>
```

Add an established FD connection to the list of fd's managed by the command processor.

Parameters

<i>fd</i>	The file descriptor.
<i>require_auth</i>	Do we Authenticate the connection on this FD?
<i>time_out_conn</i>	Do we time out the connection on this FD?
<i>local_echo</i>	Do we provide a local echo for this FD (default is true)?

Return values

<i>CMD_OK</i>	Command processor task started properly.
<i>CMD_FAIL</i>	Command processor task failed to start.
<i>CMD_TO_MANY_FDS</i>	Too many file descriptors are currently in use.

14.7.2.2 CmdListenOnSshPort()

```
int CmdListenOnSshPort (
    uint16_t port,
    int max_connections)
#include <command.h>
Start listening for a connection over SSH.
```

Parameters

<i>port</i>	The port number to listen on.
<i>max_connections</i>	What are the max number of connections we should allow on this port?

Return values

<i>CMD_OK</i>	Successfully removed the file descriptor.
<i>CMD_TO_MANY_FDS</i>	Too many file descriptors are currently in use.
<i>CMD_FAILED_SSH_KEY_GEN</i>	Unable to generate the key needed for SSH communication.

14.7.2.3 CmdListenOnTcpPort()

```
int CmdListenOnTcpPort (
    uint16_t port,
    int do_telnet_processing,
    int max_connections)
#include <command.h>
Start listening for a connection over TCP.
```

Parameters

<i>port</i>	The port number to listen on.
<i>do_telnet_processing</i>	Should we treat the port as telnet and process telnet negotiations?
<i>max_connections</i>	What are the max number of connections we should allow on this port?

Return values

<i>CMD_OK</i>	Successfully removed the file descriptor.
<i>CMD_TO_MANY_FDS</i>	Too many file descriptors are currently in use.

14.7.2.4 CmdListenQuietOnSshPort()

```
int CmdListenQuietOnSshPort (
    uint16_t port,
    int max_connections)
#include <command.h>
Start listening for a connection over SSH, but without the siggnon or password.
```

Parameters

<i>port</i>	The port number to listen on.
<i>max_connections</i>	What are the max number of connections we should allow on this port?

Return values

<i>CMD_OK</i>	Successfully removed the file descriptor.
<i>CMD_TO_MANY_FDS</i>	Too many file descriptors are currently in use.
<i>CMD_FAILED_SSH_KEY_GEN</i>	Unable to generate the key needed for SSH communication.

14.7.2.5 CmdListenQuietOnTcpPort()

```
int CmdListenQuietOnTcpPort (
    uint16_t port,
    int do_telnet_processing,
    int max_connections)
```

```
#include <command.h>
```

Start listening for a connection over TCP, but without the siggnon or password.

Parameters

<i>port</i>	The port number to listen on.
<i>do_telnet_processing</i>	Should we treat the port as telnet and process telnet negotiations?
<i>max_connections</i>	What are the max number of connections we should allow on this port?

Return values

<i>CMD_OK</i>	Successfully removed the file descriptor.
<i>CMD_TO_MANY_FDS</i>	Too many file descriptors are currently in use.

14.7.2.6 CmdRemoveCommandFd()

```
int CmdRemoveCommandFd (
    int fd)
```

```
#include <command.h>
```

Remove an established FD (either a TCP session, a serial connection, or an SSH session).

Parameters

<i>fd</i>	The file descriptor to remove.

Return values

<i>CMD_OK</i>	Successfully removed the file descriptor.
<i>CMD_FAIL</i>	Failed to remove the file descriptor.

14.7.2.7 CmdStartCommandProcessor()

```
int CmdStartCommandProcessor (
    int priority)
#include <command.h>
Start the command processor.
```

Parameters

<i>priority</i>	The NBRTOS task priority to run at.

Return values

<i>CMD_OK</i>	Command processor task started properly.
<i>CMD_FAIL</i>	Command processor task failed to start.

14.7.2.8 CmdStopListeningOnSshPort()

```
int CmdStopListeningOnSshPort (
    uint16_t port)
```

```
#include <command.h>
```

Stop Listening for connections on the specified port. Also closes all open connections that were based on that port.

Parameters

<i>port</i>	The port number to listen on.

Return values

<i>CMD_OK</i>	Successfully removed the file descriptor.
<i>CMD_FAIL</i>	Port requested wasn't listening for a connection.

14.7.2.9 CmdStopListeningOnTcpPort()

```
int CmdStopListeningOnTcpPort (
    uint16_t port)
```

```
#include <command.h>
```

Stop Listening for connections on the specified port. Also closes all open connections that were based on that port.

Parameters

<i>port</i>	The port number to listen on.

Return values

<i>CMD_OK</i>	Successfully removed the file descriptor.
<i>CMD_FAIL</i>	Port requested wasn't listening for a connection.

14.7.2.10 SendToAll()

```
void SendToAll (
    const char * buffer,
    int len,
    bool include_serial_ports)
```

```
#include <command.h>
```

Send a message to all connected sockets, excluding "Listening sockets".

Parameters

<i>buffer</i>	A buffer holding the message to send.
<i>len</i>	The length of the buffer to send.
<i>include_serial_ports</i>	Also send to serial ports?

14.7.3 Variable Documentation

14.7.3.1 CmdAuthenticateFunc

```
int(* CmdAuthenticateFunc) (const char *name, const char *passwd) (
    const char * name,
    const char * passwd) [extern]
```

```
#include <command.h>
```

External Authentication function CALLBACK for TCP connections, used to verify username and password. If this function pointer is not NULL then each new Telnet session will be asked to authenticate.

Parameters

<i>name</i>	The user name trying to authenticate.
<i>passwd</i>	The password provided for the user.

Return values

<i>CMD_OK</i>	If the authentication was successful.
<i>CMD_CLOSE</i>	If the authentication was not successful.

14.7.3.2 CmdAuthenticateSshFunc

```
int(* CmdAuthenticateSshFunc) (const char *name, const char *authVal, AuthType authType) (
    const char * name,
    const char * authVal,
    AuthType authType) [extern]
```

```
#include <command.h>
```

External Authentication function CALLBACK for SSH connections, used to verify username and password. If this function pointer is not NULL then each new SSH session will be asked to authenticate. If this function pointer is NULL, it tries to use CmdAuthenticateFunc instead.

Parameters

<i>name</i>	The user name trying to authenticate.
<i>authVal</i>	The authorization value provided for the user.
<i>authType</i>	The type of authorization value passed in.

Return values

<i>CMD_OK</i>	If the authentication was successful.
<i>CMD_CLOSE</i>	If the authentication was not successful.

14.7.3.3 CmdChar_func

```
int (* CmdChar_func) (char command, FILE *fRespondto, void *pData) (
    char command,
    FILE * fRespondto,
    void * pData) [extern]
```

```
#include <command.h>
```

The command processing callback function for handling single character commands. If this is implemented does not do echo or line editing this is the responsibility of the application programmer.

Parameters

<i>command</i>	The single command character.
<i>fRespondto</i>	The FILE * all responses should be set to (use fprintf).
<i>pData</i>	Any data that is associated from the session. This is returned by CmdConnect_func.

Return values

<i>CMD_OK</i>	If the command was okay.
<i>CMD_CLOSE</i>	If the command causes the session to terminate (Close).

See also

[CmdCmd_func](#)

[CmdConnect_func](#)

14.7.3.4 CmdCmd_func

```
int (* CmdCmd_func) (const char *command, FILE *fRespondto, void *pData) (
    const char * command,
    FILE * fRespondto,
    void * pData) [extern]
```

```
#include <command.h>
```

The command processing callback function for handling string commands.

Parameters

<i>command</i>	A pointer to the null terminated ASCII text of the command.
<i>fRespondto</i>	The FILE * all responses should be set to (use fprintf).
<i>pData</i>	Any data that is associated from the session and is returned by CmdConnect_func.

Return values

<i>CMD_OK</i>	If the command was okay.
<i>CMD_CLOSE</i>	If the command causes the session to terminate (Close).

See also

[CmdChar_func](#)

[CmdConnect_func](#)

14.7.3.5 CmdConnect_func

```
void (* CmdConnect_func) (FILE *fRespondto) (
    FILE * fRespondto) [extern]
```

```
#include <command.h>
```

Connect callback function. If this function is not NULL, then the system will call this function every time a new session is started.

Parameters

<i>fRespondto</i>	The FILE * all responses should be set to (use fprintf).

Return values

<i>Returns</i>	an arbitrary void* data item to be associated with the session. This data item is the same pData item used in other callback functions such as CmdPrompt_func.

See also

[CmdCmd_func](#)

[CmdChar_func](#)

14.7.3.6 CmdDisconnect_func

```
void(* CmdDisconnect_func) (FILE *fRespondto, int cause, void *pData) (
    FILE * fRespondto,
    int cause,
    void * pData) [extern]
```

```
#include <command.h>
```

Dis-Connect callback function, if this function is not NULL then the system will call this function every time a session is terminated.

Parameters

<i>fRespondto</i>	The FILE * all responses should be set to (use fprintf).
<i>cause</i>	Why it disconnected. Can be any value from Command Processor Disconnect Causes .
<i>pData</i>	Any data that is associated from the session. This is returned by CmdConnect_func.

See also

[CmdConnect_func](#)

14.7.3.7 CmdIdleTimeout

```
int CmdIdleTimeout [extern]
#include <command.h>
```

User defined value that specifies the number of seconds a connection is idle before it is terminated due to inactivity.

14.7.3.8 CmdPrompt_func

```
void(* CmdPrompt_func) (FILE *fRespondto, void *pData) (
    FILE * fRespondto,
    void * pData) [extern]
```

```
#include <command.h>
```

Prompt callback function. If this function is not NULL, then the system will call this function every time a new prompt line needs to be displayed.

Parameters

<i>fRespondto</i>	The FILE * all responses should be set to (use fprintf).
<i>pData</i>	Any data that is associated from the session. This is returned by CmdConnect_func.

See also

[CmdConnect_func](#)

14.7.4 Command Processor Disconnect Causes

Macros

- #define **CMD_DIS_CAUSE_TIMEOUT** (1)
Connection timed out.
- #define **CMD_DIS_CAUSE_CLOSED** (2)
Connection closed.
- #define **CMD_DIS_SOCKET_CLOSED** (3)
Socket closed. Don't send a response for this case.
- #define **CMD_DIS_AUTH_FAILED** (4)
Authorization failed. Don't send a response for this case.

14.7.4.1 Detailed Description

Possible causes given for disconnects.

14.7.5 Command Processor Listen Channels

Enumerations

- enum [ListenOn](#) : int8_t { [eListenOnTcp](#) = 1 , [eListenOnSsh](#) = 2 }
- The various remote channels that the command processor can listen to.*

14.7.5.1 Detailed Description

14.7.5.2 Enumeration Type Documentation

14.7.5.2.1 ListenOn

```
enum ListenOn : int8_t
#include <command.h>
```

The various remote channels that the command processor can listen to.

Enumerator

eListenOnTcp	Listen on TCP.
eListenOnSsh	Listen on SSH.

14.7.6 Command Processor Response Codes

Macros

- #define **CMD_OK** (0)
Function was executed properly.
- #define **CMD_CLOSE** (1)
Connection closed.
- #define **CMD_FAIL** (2)

- General function failed.*
- #define **CMD_AUTH_FAIL** (3)
Authorization failed.
- #define **CMD_TO_MANY_FDS** (4)
Too many file descriptors currently in use.
- #define **CMD_FAILED_SSH_KEY_GEN** (5)
Unable to correctly generate key needed for SSH connections.

14.7.6.1 Detailed Description

Valid return codes that can be returned by the NetBurner Command Processor functions.

14.8 Configuration System

Configuration System Variables.

Topics

- [Configuration Network Objects](#)
- [Configuration Server](#)
- [Configuration Time Objects](#)
- [Configuration Variables](#)

Configuration System Variables.

14.8.1 Detailed Description

Configuration System Variables.

```
#include< config_fs_bulkstore.h>
```

```
#include< config_obj.h>
```

14.8.2 Configuration Network Objects

Classes

- class [config_localname](#)
Configure device network name.
- class [CUR_IPADDR4](#)
- class [I4Record](#)
- class [Dynamic_IPADDR](#)
Dynamic_IPADDR.
- class [I6Record](#)

14.8.2.1 Detailed Description

```
#include< config_netobj.h>
```

14.8.3 Configuration Server

Classes

- class [WebConfigServlet](#)
Class to run web configurator.

Functions

- void [EnableConfigMirror](#) ()
Enable the configuration mirror.
- void [SaveConfigToStorage](#) ()
Write all pending data to flash memory.
- size_t [ConfigSize](#) ()
Returns the number of bytes currently in use by configuration flash.
- size_t [ConfigMaxSize](#) ()
Returns the number of bytes available in configuration flash.

14.8.3.1 Detailed Description

```
#include< config_server.h>
```

The configuration server manages the configuration variables, which can be used internally, or optionally presented on a configuration server web page. Please refer to the configuration object section for more details.

There is a default factory web interface that is normally running on the device as a user interface to set configuration options. Enabling the Configuration Mirror provides a method to replace the factory interface with your own custom interface. The is an easy way to use your company brand on the device.

Any time a configuration variable is modified, it is marked as pending. Modified values will not be written to flash memory until a call to [SaveConfigToStorage\(\)](#) is executed. Note that if a user clicks on the Update button on the configuration web page, values will be written to flash.

14.8.3.2 Function Documentation

14.8.3.2.1 ConfigMaxSize()

```
size_t ConfigMaxSize ()
#include <config_server.h>
```

Returns the number of bytes available in configuration flash.

Returns

The maximum number of bytes available. This function is implemented in hal.cpp

14.8.3.2.2 ConfigSize()

```
size_t ConfigSize ()
#include <config_server.h>
```

Returns the number of bytes currently in use by configuration flash.

Returns

Number of bytes in use

14.8.3.2.3 EnableConfigMirror()

```
void EnableConfigMirror ()
#include <config_server.h>
```

Enable the configuration mirror.

When enabled, the application configuration web interface will replace the default system web interface. In this way you can easily customize and brand the configuration web interface for your company.

Note

This is an empty function. To enable the config mirror, it needs to be called within the application code, which in turn forces the function to be linked during compilation. The linking is what enables the config mirror.

14.8.3.2.4 SaveConfigToStorage()

```
void SaveConfigToStorage ()
```

```
#include <config_server.h>
```

Write all pending data to flash memory.

Checks the pending flag of all configuration system objects and writes updates to flash memory

14.8.4 Configuration Time Objects**Classes**

- class [config_time_t](#)
Class to hold time configuration.

14.8.4.1 Detailed Description

```
#include< config_time.h>
```

14.8.5 Configuration Variables

Configuration System Variables.

Topics

- [Configuration Variable Flags](#)

Classes

- class [config_obj](#)
Base class used to create configuration objects.
- class [config_value](#)
Base class used to create a configuration value.
- class [config_uint](#)
Unsigned 32-bit Integer Configuration Variable.
- class [config_int](#)
Signed 32-bit Integer Configuration Variable.
- class [config_double](#)
Double Float Configuration Variable.
- class [config_bool](#)
Boolean Configuration Variable.
- class [config_string](#)
String Configuration Variable.
- class [config_pass](#)
Password string Configuration Variable.
- class [config_IPADDR4](#)
Configuration Variable for [IPADDR4](#) (IPv4) object types.
- class [config_IPADDR](#)
Configuration Variable for [IPADDR](#) (IPv6) object type.

- class [config_MACADR](#)
Configuration Variable for [MACADR](#) object type.
- class [config_chooser](#)
Chooser Configuration Variable - Select From a List of Items.
- class [config_int_limit](#)
A [config_int](#) with minimum and/or maximum values Attempting to set a value lower than the minimum or greater than the maximum will fail with an error both programmatically and in the Config Web UI.
- class [config_uint_limit](#)
A [config_uint](#) with minimum and/or maximum values Attempting to set a value lower than the minimum or greater than the maximum will fail with an error both programmatically and in the Config Web UI.
- class [config_double_limit](#)
A [config_double](#) with minimum and/or maximum values Attempting to set a value lower than the minimum or greater than the maximum will fail with an error both programmatically and in the Config Web UI.
- class [config_string_limit](#)
A [config_string](#) with minimum and/or maximum lengths Attempting to set a value shorter than the minimum or longer than the maximum will fail with an error both programmatically and in the Config Web UI.
- class [config_pass_limit](#)
A [config_pass](#) with minimum and/or maximum lengths Attempting to set a value shorter than the minimum or longer than the maximum will fail with an error both programmatically and in the Config Web UI.

Functions

- void **SaveConfigToStorage** ()
Save configuration to flash storage.
- void **OverrideConfig** ()
Create this function to override config settings.

14.8.5.1 Detailed Description

Configuration System Variables.

```
#include< config_obj.h>
```

Configuration variable objects are part of the NetBurner 3.x configuration system. The use of configuration objects facilitates a much easier method to create and store non-volatile information for both system configuration and for user application configuration and storage. They use a JSON interface so the information can be easily downloaded to save a configuration, or uploaded to configure the entire device at one time, such as in production bring-up. This includes system configuration such as IP addresses, and also any settings specific to your application.

The JSON value types include: integer, unsigned integer, float, boolean and string. There are also objects for IPv4 IP, IPv6 IP, MAC addresses, passwords, and selection lists options. In JSON everything is represented as a string, so there are object member functions to convert between values. These functions make use of the [NBString](#) class. For customers using the 2.x tools, the User Parameter flash sector method is still available, but we do recommend converting to the new configuration system. In 3.x you have the ability to access the data through a web configuration page and also through JSON, but you can certainly choose not to use those features and treat the configuration system as internal storage, and any configuration can be done through your application just as with the 2.x tool set.

Note

All modifications to configuration objects are marked as pending. A call to [SaveConfigToStorage](#) is required to save changes to flash memory.

14.8.5.2 Function Documentation

14.8.5.2.1 OverrideConfig()

```
void OverrideConfig ()
#include <config_obj.h>
```

Create this function to override config settings.

This function exists as a weak reference. So you just need to create it and then force set any config variables you want to control. This will be called after config is loaded before any config is used by anything. It will also be called when anyone calls `revert` on the config system.

14.8.5.3 Configuration Variable Flags

Variables

- `const uint32_t fConfigValueLeaf = 0x01`
Value is a leaf.
- `const uint32_t fConfigReadOnly = 0x02`
Variable is read-only.
- `const uint32_t fConfigModified = 0x04`
Variable has been modified, but not yet saved.
- `const uint32_t fConfigHidden = 0x08`
Not visible to configuration web server display.
- `const uint32_t fConfigNoSave = 0x10`
Do not save to flash memory when save functions are called.
- `const uint32_t fConfigNoObscure = 0x20`
Do not obscure the value.
- `const uint32_t fConfigNeedReboot = 0x40`
System reboot required for changes to take effect.
- `const uint32_t fConfigNoReload = 0x80`
Disable reloading value from flash during a call to `ReloadFromFlash`.
- `const uint32_t fConfigDetached = 0x100`
Disable reloading value from flash during a call to `ReloadFromFlash`.
- `const uint32_t fConfigIsDefault = 0x200`
Value is unchanged from the default, i.e. was never set.
- `const uint32_t fMaskDoingSave = 0x40000000`
We're currently executing a config save to non-volatile storage.

14.8.5.3.1 Detailed Description

Specifies the characteristics of a configuration variable, such as read-only, hidden, etc.

14.9 Constants (constants.h)

NetBurner System Constants.

Macros

- `#define TICK_IRQ_LEVEL (5)`
System clock IRQ level.
- `#define SERIAL_IRQ_LEVEL (3)`
- `#define SERIAL_VECTOR_BASE (64)`
- `#define TICKS_PER_SECOND (20)`
System clock ticks per second.
- `#define OS_MAX_TASKS (32)`
Max number of system tasks.
- `#define OS_MAX_PRIOS (64)`

Maximum number of system priorities (NBRTOS limit: 256)

- **#define SSH_TASK_PRIORITY** (56)
ief SSH server must be lower than main for progress displays

Ethernet buffer defines

OS Maximum Interrupt Level

OS_MAX_IRQ_MASK - Optional, Maximum IRQ level that the RTOS is allowed to mask. RTOS objects may not be used in ISRs above this level.

- **#define ETHER_BUFFER_SIZE** (1548)
- **#define ETH_MAX_PAYLOAD** (1500)
- **#define ETH_MAX_SIZE** (1522)
- **#define ETH_MIN_SIZE** (46)
- **#define IP_HEADER_SIZE** (20)
- **#define UDP_HEADER_SIZE** (8)
- **#define MAX_UDPDATA** (ETH_MAX_PAYLOAD - (IP_HEADER_SIZE + UDP_HEADER_SIZE))
- **#define SERIAL_TX_BUFFERS** (2)
ETHERN_BUFFER_SIZE = bytes of serial TX fifo.
- **#define SERIAL_RX_BUFFERS** (2)
ETHERN_BUFFER_SIZE = bytes of serial RX fifo.
- **#define stdin_buffer_size** (200)

System task priorities

Restating the NBRTOS RTOS Library document (NBRTosLibrary.pdf) Lowest priority is OS_MAX_PRIOS-1, 1 is the highest. There can only be one task at each priority level. Idle task is created at priority 63.

UserMain is created at priority 10. It is recommended and is supported by examples and default projects that a call to OSChangePrio(MAIN_PRIO); be made to lower the priority to the range recommended for the main application. Factory applications use priorities 46 through 56.

Netburner Runtime library support and driver tasks use 36 through 45.

Care should be taken in use of priorities it can affect reliability and performance.

- **#define MAIN_PRIO** (50)
Recommend UserMain priority.

Runtime library driver and support task priorities

- **#define USB_HW_PRIO** (45)
- **#define PPP_PRIO** (44)
- **#define SECURITY_TASK_PRIO** (43)
- **#define WIFI_STATION_TASK_PRIO** (42)
- **#define WIFI_TASK_PRIO** (41)

- `#define CONFIG_SERVER_PRIO` (40)
- `#define HTTP_PRIO` (39)
- `#define ETHER_SEND_PRIO` (38)
- `#define MQTT_TASK_PRIO` ([MAIN_PRIO](#)-1)

14.9.1 Detailed Description

NetBurner System Constants.

```
#include< constants.h>
```

14.9.2 Macro Definition Documentation

14.9.2.1 TICKS_PER_SECOND

```
#define TICKS_PER_SECOND (20)
```

```
#include <constants.h>
```

System clock ticks per second.

PLEASE READ THIS BEFORE MODIFYING TICKS_PER_SECOND

Before you change this value understand what changing it does. Making it faster only slows things down. It does not speed up task switches, If only changes the granularity of time delays and timeouts. Task switches happen much much faster than the tick interval. They happen as soon as one task blocks or an interrupt /or task causes a higher priority task to be unblocked. The Time tick has NOTHING to do with task switches.

If you change it to the maximum 200 you have increased the RTOS overhead by a factor of 10 and have actually slowed your system.

14.10 DHCP

Topics

- [DHCP Server](#)
- [DHCPv4 Client](#)
- [DHCPv6 Client](#)

14.10.1 Detailed Description

[DHCP](#) Client functions. Refer to DHCPv4 or DHCPv6 documents for IPv4 or IPv6.

The [DHCP](#) client services are automatically controlled by the configuration option when the [init\(\)](#) function is called in an application. Options are:

- [DHCP](#)
- [DHCP](#) with Static Fallback
- Static

You may also create your own [DHCP](#) objects.

14.10.2 DHCP Server

Namespaces

- namespace [DHCP](#)
[DHCP](#) Namespace.

Macros

- `#define DHCP_SERV_MAX_INTF (4)`
- `#define DHCP_OFFER_DURATION (2 * TICKS\_PER\_SECOND)`
- `#define DHCP_SERV_MAX_HOSTNAME_LEN (32)`
- `#define LEASE_POOL_SIZE 150`
- `#define DHCPD_STARTING_ADDRESS 0xC0A80184`

Functions

- bool [AddStandardDHCPServer](#) (int intf=0, [IPADDR4](#) startAddr=[IPADDR4::NullIP\(\)](#))
Starts a standard allocator [DHCP](#) server.

14.10.2.1 Detailed Description

```
#include< dhcpd.h>
```

Adding a [DHCP](#) server will enable your NetBurner device to serve IPv4 addresses. This can be useful in circumstances such as direct connections between 2 devices.

14.10.2.2 Function Documentation

14.10.2.2.1 AddStandardDHCPServer()

```
bool AddStandardDHCPServer (
    int intf = 0,
    IPADDR4 startAddr = IPADDR4::NullIP\(\))
```

```
#include <dhcpd.h>
```

Starts a standard allocator [DHCP](#) server.

Also checks for existing [DHCP](#) servers on the specified network interface.

Parameters

<i>intf</i>	The interface to use. If no parameter is specified the default value is 0 which will use the first system interface.
<i>startAddr</i>	The starting IP address. If no parameter is specified the default range of 192.168.1.100 to 192.168.1.249 will be used.

Return values

<i>true</i>	Success
<i>false</i>	A server already exists or error starting the server

14.10.3 DHCPv4 Client

Topics

- [DHCP State](#)

Classes

- class [DhcpObject](#)
DHCP client class.

Functions

- `int32_t GetInterfaceDHCPState (int interface=0)`
Returns current state of the [DHCP](#) lease, with optional interface parameter.
- `int32_t WaitForDHCPInterface (int interface=0, uint16_t TicksToWait=10 * TICKS\_PER\_SECOND)`
Wait until a [DHCP](#) lease is obtained, or the timeout occurs.

14.10.3.1 Detailed Description

```
#include< dhcpclient.h>
```

14.10.3.2 Function Documentation

14.10.3.2.1 GetInterfaceDHCPState()

```
int32_t GetInterfaceDHCPState (
    int interface = 0)
#include <dhcpclient.h>
```

Returns current state of the [DHCP](#) lease, with optional interface parameter.

The [DhcpObject::GetDHCPState\(\)](#) function is a member function of the [DHCP](#) client class.

Returns

[DHCP State](#)

See also

[DhcpObject::GetDHCPState](#)

14.10.3.2.2 WaitForDHCPInterface()

```
int32_t WaitForDHCPInterface (
    int interface = 0,
    uint16_t TicksToWait = 10 * TICKS\_PER\_SECOND)
#include <dhcpclient.h>
```

Wait until a [DHCP](#) lease is obtained, or the timeout occurs.

Parameters

<i>interface</i>	The network interface to wait on.
<i>TicksToWait</i>	The number of system Time Ticks to wait. The TICKS_PER_SECOND constant can be used to more easily specify a number of seconds. If not specified, the default timeout is 10 seconds.

Returns

[DHCP State](#)

See also

[DhcpObject::GetDHCPState](#)

14.10.3.3 DHCP State

Macros

- **#define SDHCP_NOTSTARTED 0**
The System has not been initialized.
- **#define SDHCP_DISCOVER 1**
The system is discovering the [DHCP](#) servers.
- **#define SDHCP_OFFER 2**
The system has responded to an OFFER.
- **#define SDHCP_ACK 3**
The System has Acknowledged the OFFER.
- **#define SDHCP_INIT 4**
The System is reinitializing.
- **#define SDHCP_CMPL 5**
The System has obtained a valid [DHCP](#) lease.
- **#define SDHCP_RENEW 6**
The System is in the process of renewing.
- **#define SDHCP_REBIND 7**
The System has failed the Renew and is trying to Rebind.
- **#define SDHCP_RELEASE 8**
The System is trying to release the Lease.
- **#define SBOOTP_TRANSMITTING 9**
Trying BOOTP.
- **#define SBOOTP_DONE 10**
BOOTP complete.
- **#define SDHCP_FAILED 11**
[DHCP](#) attempt failed - could not obtain a [DHCP](#) lease.

14.10.3.3.1 Detailed Description

14.10.4 DHCPv6 Client

Functions

- void [InitIPv6](#) (int ifc=0)
- void [ShowAllV6Info](#) ()
- void [StartDHCPv6_Solicit](#) (int ifnum=-1)
Manually starts the DHCPv6 Client in Full Solicitation mode.
- void [StartDHCPv6_InfoReq](#) (int ifnum=-1)
Manually starts the DHCPv6 Client in Full Solicitation mode.
- void [StartDHCPv6](#) (int ifnum=-1)
Manually starts the DHCPv6 Client in Information Request mode.
- bool [AddStaticIPv6Address](#) (const [IPADDR6](#) &ip, int ifnum=-1)
Add a static IPv6 address to an interface.
- bool [RemoveStaticIPv6Address](#) (const [IPADDR6](#) &ip, int ifnum=-1)
Add a static IPv6 address to an interface.

14.10.4.1 Detailed Description

```
#include< ipv6_intf.h>
```

14.10.4.2 Function Documentation

14.10.4.2.1 AddStaticIPv6Address()

```
bool AddStaticIPv6Address (
    const IPADDR6 & ip,
    int ifnum = -1)
#include <ipv6_intf.h>
Add a static IPv6 address to an interface.
```

Parameters

<i>ip</i>	The IP address to add.
<i>ifnum</i>	The desired IPv6Interface to operate on. A negative value will use the first interface.

Return values

<i>true</i>	IP address successfully added
<i>false</i>	Error

See also

[RemoveStaticIPv6Address\(\)](#)

14.10.4.2.2 InitIPv6()

```
void InitIPv6 (
    int ifc = 0)
#include <ipv6_intf.h>
```

14.10.4.2.3 RemoveStaticIPv6Address()

```
bool RemoveStaticIPv6Address (
    const IPADDR6 & ip,
    int ifnum = -1)
#include <ipv6_intf.h>
Add a static IPv6 address to an interface.
```

Parameters

<i>ip</i>	The IP address to remove.
<i>ifnum</i>	The desired IPv6Interface to operate on. A negative value will use the first interface.

Return values

<i>true</i>	IP address successfully removed
<i>false</i>	Error

See also

[AddStaticIPv6Address\(\)](#)

14.10.4.2.4 ShowAllV6Info()

```
void ShowAllV6Info ()
#include <ipv6_intf.h>
```

14.10.4.2.5 StartDHCPv6()

```
void StartDHCPv6 (
    int ifnum = -1) [inline]
#include <ipv6_intf.h>
```

Manually starts the DHCPv6 Client in Information Request mode.

Parameters

<i>ifnum</i>	The desired IPv6Interface to operate on. A negative value indicates all interfaces

See also

[StartDHCPv6_Solicit\(\)](#)

14.10.4.2.6 StartDHCPv6_InfoReq()

```
void StartDHCPv6_InfoReq (
    int ifnum = -1)
#include <ipv6_intf.h>
```

Manually starts the DHCPv6 Client in Full Solicitation mode.

Parameters

<i>ifnum</i>	The desired IPv6Interface to operate on. A negative value indicates all interfaces

14.10.4.2.7 StartDHCPv6_Solicit()

```
void StartDHCPv6_Solicit (
    int ifnum = -1)
#include <ipv6_intf.h>
```

Manually starts the DHCPv6 Client in Full Solicitation mode.

Parameters

<i>ifnum</i>	The desired IPv6Interface to operate on. A negative value indicates all interfaces

See also

[StartDHCPv6\(\)](#)

14.11 DNS - Domain Name System**Topics**

- [DNS Record Types](#)
- [DNS Return Codes](#)

Functions

- bool [IsNameIPAddress](#) (const char *name)
Determine if the name is a valid IP Address and does not need to be looked up.
- int [fd_dns_part1](#) (const char *name, const [IPADDR](#) &dns_server, uint16_t TYPE=[DNS_A](#), uint16_t TYPE2=0, int ifn=-1)
Open a UDP socket and initiate a DNS lookup.
- bool [fd_dns_processresult](#) (int fd, const char *name, [IPADDR](#) &addr_out, uint16_t TYPE=[DNS_A](#), uint16_t TYPE2=[DNS_AAAA](#), uint32_t *ttl=0)
Process any responses on the UDP socket opened for DNS.
- int [fd_outstanding_Responses](#) (int fd)
Check to see if there are any outstanding DNS requests.
- bool [AnyDNSInterFaceActive](#) ()
Determine if we have an active DNS route; DNS server is set for an active interface.
- int [GetHostByName](#) (const char *name, [IPADDR](#) *plpaddr, const [IPADDR](#) &dns_server, const TickTimeout tout, uint16_t TYPE1=[DNS_A](#), uint16_t TYPE2=[extra_dns_t](#), uint32_t *ttl=NULL)
Get the IP address associated with the specified domain name.
- int [GetHostByNameVialfNum](#) (const char *name, [IPADDR](#) *plpaddr, const [IPADDR](#) &dns_server, int ifn, const TickTimeout &tout, uint16_t TYPE1=[DNS_A](#), uint16_t TYPE2=[extra_dns_t](#), uint32_t *ttl=NULL)
Get the IP address associated with the specified domain name on a specific interface.
- bool [CheckDnsCache](#) (const char *name, [IPADDR](#) &ia, uint32_t *ttl, uint16_t rtype)
Check the DNS cache for a stored response.

14.11.1 Detailed Description

```
#include< dns.h>
```

The current DNS system is agnostic about the use of [IPADDR](#), [IPADDR6](#) or [IPADDR4](#). The capability of C++ is such that you pass in any of those types and the compiler will select the correct function for that type. This is a change from tools versions 3.4.0 and earlier in which explicit IPv4 and IPv6 functions were used.

Note

While we are pointing out the automatic functionality in this DNS API, the same is true for most functions, structures and classes throughout the NetBurner API libraries.

The DNS system can be accessed as a blocking or synchronous operation with: [GetHostByName\(\)](#). If you want to do asynchronous non-blocking lookups, please refer to the functions:

- [fd_dns_part1\(\)](#)
- [fd_dns_processresult\(\)](#)

14.11.2 Function Documentation

14.11.2.1 AnyDNSInterFaceActive()

```
bool AnyDNSInterFaceActive ()
#include <dns.h>
```

Determine if we have an active DNS route; DNS server is set for an active interface.

Returns

true if active DNS is available somewhere

See also

[GetHostByName\(\)](#)
[InterfaceLinkActive\(int interface\);](#)

14.11.2.2 CheckDnsCache()

```
bool CheckDnsCache (
    const char * name,
    IPADDR & ia,
    uint32_t * ttl,
    uint16_t rtype)
#include <dns.h>
Check the DNS cache for a stored response.
```

Parameters

<i>*name</i>	Pointer to domain name to resolve
<i>ia</i>	Variable of type IPADDR to store resultant IP address
<i>ttl</i>	Optional pointer to return variable for TimeToLive value for DNS record
<i>rtype</i>	Request type: DNS Record Types

Returns

true if the cache had a valid answer for this query

14.11.2.3 fd_dns_part1()

```
int fd_dns_part1 (
    const char * name,
    const IPADDR & dns_server,
    uint16_t TYPE = DNS_A,
    uint16_t TYPE2 = 0,
    int ifn = -1)
#include <dns.h>
Open a UDP socket and initiate a DNS lookup.
This socket can be put in a select and used for asynchronous DNS lookups. Opens a UDP receive FD and sends
DNS queries.
```

Parameters

<i>name</i>	Pointer to domain name to resolve
<i>dns_server</i>	Reference to the DNS server to use. Pass INADDR_ANY to use the DNS server associated with the interface.
<i>TYPE</i>	Optional DNS record type. If no type is specified the default is IPv4 DNS_A
<i>TYPE2</i>	Optional 2nd DNS record type. If no type is specified the default is no look up
<i>ifn</i>	Which interface to use -1 uses the default active DNS interface

Returns

fd The file descriptor of the udp socket. [DNS Return Codes](#)

See also

[fd_dns_processresult\(\)](#), [fd_outstanding_Responses\(\)](#), [GetHostByName\(\)](#),

14.11.2.4 fd_dns_processresult()

```
bool fd_dns_processresult (
    int fd,
    const char * name,
```

```

    IPADDR & addr_out,
    uint16_t TYPE = DNS_A,
    uint16_t TYPE2 = DNS_AAAA,
    uint32_t * ttl = 0)

```

```
#include <dns.h>
```

Process any responses on the UDP socket opened for DNS.

This should only be called when the socket has data, otherwise it will block.

Parameters

<i>fd</i>	The file descriptor opened with <code>fd_dns_part1</code>
<i>name</i>	Pointer to domain name to resolve
<i>addr_out</i>	Reference to IPADDR variable to hold the result
<i>TYPE</i>	DNS record type. If no type is specified the default is IPv4 DNS_A
<i>TYPE2</i>	2nd DNS record type. If no type is specified the default is no look up
<i>ttl</i>	Optional pointer to return variable for TimeToLive value for DNS record

Returns

true if resolution was successful

See also

[fd_dns_part1\(\)](#), [fd_outstanding_Responses\(\)](#), [GetHostByName\(\)](#),

14.11.2.5 fd_outstanding_Responses()

```

int fd_outstanding_Responses (
    int fd)

```

```
#include <dns.h>
```

Check to see if there are any outstanding DNS requests.

If this returns 0 then all DNS responses have been processed and one can assume there is no such name.

Parameters

<i>fd</i>	The file descriptor opened with <code>fd_dns_part1</code>

Returns

The number of outstanding DNS requests that have not been answered.

See also

[fd_dns_part1\(\)](#), [fd_dns_processresult\(\)](#), [GetHostByName\(\)](#),

14.11.2.6 GetHostByName()

```

int GetHostByName (
    const char * name,
    IPADDR * pIpaddr,
    const IPADDR & dns_server,
    const TickTimeout tout,
    uint16_t TYPE1 = DNS_A,
    uint16_t TYPE2 = extra_dns_t,
    uint32_t * ttl = NULL) [inline]
#include <dns.h>

```

Get the IP address associated with the specified domain name.

Calling [GetHostByName\(\)](#) in dual stack mode will automatically call the correct IPv4 or IPv6 function.

The function will attempt to retrieve record type 1 first. If successful the function returns. If the attempt using type 1 fails, the function will attempt to get a record using the type 2 parameter. For example, let's say you wish your code to use IPv6, but it's OK to fall back to IPv4. You can set type 1 to DNS_AAAA and type2 to DNS_A.

Parameters

<i>*name</i>	Pointer to domain name to resolve
<i>*pIpaddr</i>	Pointer to variable of type IPADDR (or IPADDR4) to store resultant IP address.
<i>&dns_server</i>	Specified the DNS server to use. Pass INADDR_ANY to use the DNS server associated with the interface.
<i>tout</i>	Number of time ticks to wait of type TickTimeout
<i>TYPE1</i>	Optional record type. If not specified will default to IPv4 DNS_A
<i>TYPE2</i>	Optional record type. If not specified will default to IPv6 DNS_AAAA
<i>tTl</i>	Optional pointer to return variable for TimeToLive value for DNS record

Returns

[DNS Return Codes](#)

See also

[GetHostByName4\(\)](#)

[GetHostByName6\(\)](#)

14.11.2.7 GetHostByNameViaIfNum()

```
int GetHostByNameViaIfNum (
    const char * name,
    IPADDR * pIpaddr,
    const IPADDR & dns_server,
    int ifn,
    const TickTimeout & tout,
    uint16_t TYPE1 = DNS_A,
    uint16_t TYPE2 = extra_dns_t,
    uint32_t * ttl = NULL) [inline]

#include <dns.h>
```

Get the IP address associated with the specified domain name on a specific interface.

Calling [GetHostByName\(\)](#) in dual stack mode will automatically call the correct IPv4 or IPv6 function.

The function will attempt to retrieve record type 1 first. If successful the function returns. If the attempt using type 1 fails, the function will attempt to get a record using the type 2 parameter. For example, let's say you wish your code to use IPv6, but it's OK to fall back to IPv4. You can set type 1 to DNS_AAAA and type2 to DNS_A.

Parameters

<i>*name</i>	Pointer to domain name to resolve
<i>*pIpaddr</i>	Pointer to variable of type IPADDR to store resultant IP address
<i>&dns_server</i>	Specified the DNS server to use. Pass INADDR_ANY to use the DNS server associated with the interface.
<i>tout</i>	A reference to the number of time ticks to wait of type TickTimeout
<i>TYPE1</i>	Optional record type. If not specified will default to IPv4 DNS_A
<i>TYPE2</i>	Optional record type. If not specified will default to IPv6 DNS_AAAA
<i>ifn</i>	Interface number
<i>ttl</i>	Optional pointer to return variable for TimeToLive value for DNS record

Returns[DNS Return Codes](#)**See also**[GetHostByName4VialfNum\(\)](#)[GetHostByNameVialfNum\(\)](#)**14.11.2.8 IsNameIPAddress()**

```
bool IsNameIPAddress (
    const char * name)
```

```
#include <dns.h>
```

Determine if the name is a valid IP Address and does not need to be looked up.

Parameters

<i>*name</i>	Pointer to domain name to check

Returns

true If it is a valid IP address

See also[GetHostByName\(\)](#), [GetHostByName4VialfNum\(\)](#), [GetHostByName6\(\)](#)**14.11.3 DNS Record Types****Macros**

- **#define DNS_A 1**
32-bit IPv4 address
- **#define DNS_CNAME 5**
Canonical name record.
- **#define DNS_MB 7**
Mailing list subscriber list.
- **#define DNS_MG 8**
Mailing list subscriber list.
- **#define DNS_MX 15**
Mail exchange record.
- **#define DNS_AAAA 28**
128-bit IPv6 address

14.11.3.1 Detailed Description

DNS record types from RFC1035

14.11.4 DNS Return Codes**Macros**

- **#define DNS_OK (0)**
Success.
- **#define DNS_TIMEOUT (1)**
Request timed out.
- **#define DNS_NOSUCHNAME (2)**
Name not found.
- **#define DNS_ERR (3)**
Other error.

14.11.4.1 Detailed Description

14.12 Device Interrupts

Macros

- #define **DEVICE_INTERRUPT_MAXIMUM** (8)
Fixed interrupt sources.

Typedefs

- typedef enum [_DeviceInterruptTrigger](#) **DeviceInterruptTrigger**
DeviceSetupIsr interrupt trigger.
- typedef void **DeviceIsrFn**()
ISR function.

Enumerations

- enum [_DeviceInterruptTrigger](#) {
 [InterruptTriggerNone](#) , [InterruptTriggerSensitive](#) , [InterruptTriggerRisingEdge](#) , [InterruptTriggerFallingEdge](#) ,
 [InterruptTriggerBothEdge](#) }
DeviceSetupIsr interrupt trigger.

Functions

- BOOL [DevicesValid](#) (int irq)
Verify availability and support for device. (Only verifies IRQ is valid)
- BOOL [DeviceSetupIsr](#) (int irq, [DeviceInterruptTrigger](#) trigger, DeviceIsrFnPtr isrPtr)
Set interrupt controller and attach function to interrupt.
- void [DeviceAckInterrupt](#) (int irq)
Acknowledge interrupt.
- void [DeviceDisableInterrupt](#) (int irq)
Disable interrupt.
- void [DeviceEnableInterrupt](#) (int irq)
Enable interrupt.
- OS_SEM * [DeviceGetQspiSem](#) (void)
Acquire QSPI semaphore for sharing access to SPI devices.
- OS_SEM * [DeviceGetI2cSem](#) (void)
Acquire I2C semaphore for sharing access to I2C devices.

14.12.1 Detailed Description

```
#include< device.h>
```

14.12.2 Enumeration Type Documentation

14.12.2.1 _DeviceInterruptTrigger

```
enum \_DeviceInterruptTrigger
#include <device.h>
DeviceSetupIsr interrupt trigger.
```

Enumerator

InterruptTriggerNone	No interrupt level.
InterruptTriggerSensitive	Trigger-sensitive interrupt.
InterruptTriggerRisingEdge	Rising edge triggered interrupt.
InterruptTriggerFallingEdge	Falling edge triggered interrupt.
InterruptTriggerBothEdge	Both rising and falling edge triggered.

14.12.3 Function Documentation**14.12.3.1 DeviceAckInterrupt()**

```
void DeviceAckInterrupt (
    int irq)
#include <device.h>
Acknowledge interrupt.
```

Parameters

<i>irq</i>	Fixed level interrupt source

14.12.3.2 DeviceDisableInterrupt()

```
void DeviceDisableInterrupt (
    int irq)
#include <device.h>
Disable interrupt.
```

Parameters

<i>irq</i>	Fixed level interrupt source

14.12.3.3 DeviceEnableInterrupt()

```
void DeviceEnableInterrupt (
    int irq)
#include <device.h>
Enable interrupt.
```

Parameters

<i>irq</i>	Fixed level interrupt source

14.12.3.4 DeviceGetI2cSem()

```
OS_SEM * DeviceGetI2cSem (
    void )
#include <device.h>
Acquire I2C semaphore for sharing access to I2C devices.
```

Returns

Semaphore

Return values

<i>NULL</i>	for non-existent device

14.12.3.5 DeviceGetQspiSem()

```
OS_SEM * DeviceGetQspiSem (
    void )
```

```
#include <device.h>
```

Acquire QSPI semaphore for sharing access to SPI devices.

Returns

Semaphore

Return values

<i>NULL</i>	for non-existent device

14.12.3.6 DeviceIsValid()

```
BOOL DeviceIsValid (
    int irq)
```

```
#include <device.h>
```

Verify availability and support for device. (Only verifies IRQ is valid)

Parameters

<i>irq</i>	Fixed level interrupt source

Return values

<i>TRUE</i>	Support and available on module
<i>FALSE</i>	Option not supported

14.12.3.7 DeviceSetupIsr()

```
BOOL DeviceSetupIsr (
    int irq,
    DeviceInterruptTrigger trigger,
    DeviceIsrFnPtr isrPtr)
```

```
#include <device.h>
```

Set interrupt controller and attach function to interrupt.

Parameters

<i>irq</i>	Fixed level interrupt source
<i>trigger</i>	Triggering edge level
<i>isrPtr</i>	ISR function

Return values

<i>TRUE</i>	Set
<i>FALSE</i>	Not valid interrupt source

14.13 Diagnostics (DIAG)

Classes

- class [DiagItemClass](#)
Base class for all diagnostic items.
- class [DiagCallBack](#)
Callback-based diagnostic item that calls a custom function to generate diagnostic content.
- class [DiagVarMon](#)
Monitor and report the value of numeric variables.
- class [DiagStrMon](#)
Monitor and report string values.
- class [DiagPtrMon](#)
Monitor and report pointer values.

Typedefs

- typedef void [diagcallback](#)(int sock)
Callback function type for custom diagnostic reports.

Functions

- bool [SendDiagReport](#) (const char *pUrl, const char *pUser=0, const char *pPass=0)
Send the complete diagnostic report to an external URL via HTTP POST.
- bool [SendDiagReport](#) ([ParsedURI](#) &uri, const char *pUser=0, const char *pPass=0)
Send the complete diagnostic report to an external URL via HTTP POST.
- void [InitPlatformDiag](#) ()
Initialize platform-specific diagnostics.

14.13.1 Detailed Description

```
#include< diagnostics.h>
```

The functions in this header are used to add diagnostic reports to the diagnostics tab under the config page when [diagnostics.h](#) is included and [EnableSystemDiagnostics\(\)](#) is called.

The resulting JSON blob can be accessed directly (replace 10.1.1.100 with your IP or dns name) <http://10.1.1.100:20034/DIAG>

You can also access subgroups directly from the diagnostics: <http://10.1.1.100:20034/DIAG/Buffers> or <http://10.1.1.100:20034/DIAG/Tasks>

Examples

If I have a `uint32_t` called `myVar` I want to monitor:

```
DiagVarMon MyVarMon("MyVarName", myVar);
```

If I have char buffer holding the name of the current mode:

```
DiagStrMon MyStrMon("CurMode", cur_mode);
```

If I want to output a complex JSON object for a custom report, I can write a routine that outputs this object to an fd:

```
void MyCustomReport(int fd);
DiagCallBack MyReporter("MyReportName", MyCustomReport);
```

14.13.2 Typedef Documentation

14.13.2.1 diagcallback

```
typedef void diagcallback(int sock)
#include <diagnostics.h>
```

Callback function type for custom diagnostic reports.

This function type is used to create custom diagnostic output functions that can be called when the diagnostic system needs to output data for a specific diagnostic item.

Parameters

<i>sock</i>	File descriptor to write diagnostic output to

14.13.3 Function Documentation

14.13.3.1 InitPlatformDiag()

```
void InitPlatformDiag ()
#include <diagnostics.h>
```

Initialize platform-specific diagnostics.

This function is called by [EnableSystemDiagnostics\(\)](#) to allow platforms to register their own diagnostic items. The default implementation is weak-linked and does nothing. Platforms can override this by providing their own implementation without weak linkage.

14.13.3.2 SendDiagReport() [1/2]

```
bool SendDiagReport (
    const char * pUrl,
    const char * pUser = 0,
    const char * pPass = 0)
#include <diagnostics.h>
```

Send the complete diagnostic report to an external URL via HTTP POST.

This function collects all diagnostic data and sends it as a JSON payload via HTTP POST to the specified URL. This is useful for remote monitoring and logging of device diagnostics.

Parameters

<i>pUrl</i>	Target URL to send the diagnostic data to
<i>pUser</i>	Optional username for HTTP authentication
<i>pPass</i>	Optional password for HTTP authentication

Returns

true if the report was sent successfully, false otherwise

See also

[SendDiagReport\(ParsedURI&, const char*, const char*\)](#)

14.13.3.3 SendDiagReport() [2/2]

```
bool SendDiagReport (
    ParsedURI & uri,
    const char * pUser = 0,
    const char * pPass = 0)
#include <diagnostics.h>
```

Send the complete diagnostic report to an external URL via HTTP POST.

This function collects all diagnostic data and sends it as a JSON payload via HTTP POST to the specified URL using a [ParsedURI](#) object.

Parameters

<i>uri</i>	ParsedURI object containing the target URL
<i>pUser</i>	Optional username for HTTP authentication
<i>pPass</i>	Optional password for HTTP authentication

Returns

true if the report was sent successfully, false otherwise

See also

[SendDiagReport\(const char*, const char*, const char*\)](#)

14.14 EFFS - Embedded Flash File System

Topics

- [EFFS-STD Flash File System](#)

The Standard Embedded Flash File System, EFFS-STD
`#include< fsf.h>`

- [FAT File System](#)

The FAT32 Embedded Flash File System for Flash Cards, EFFS-FAT
`#include< api_f.h>`

14.14.1 Detailed Description

Two Embedded Flash File Systems are supported:

- FAT32 File System (EFFS-FAT), a FAT32 file system used on external flash memory cards and ram drives.
- Standard File System (EFFS-STD), a power-fail safe file system that resides in the flash on the module.

14.14.2 EFFS-STD Flash File System

The Standard Embedded Flash File System, EFFS-STD
`#include< fsf.h>`

Topics

- [STD File System Seek Codes](#)

Macros

- `#define fs\_getfreespace(drivenum, space) fsm_getfreespace(drivenum, space)`
Provides information about the drive space usage.
- `#define fs\_mkdir(dirname) fsm_mkdir(dirname)`
Makes a new directory.
- `#define fs\_chdir(dirname) fsm_chdir(dirname)`
Change the directory.
- `#define fs\_rmdir(dirname) fsm_rmdir(dirname)`
Removes a directory.
- `#define fs\_delete(filename) fsm_delete(filename)`
Deletes a file.

- `#define fs_findfirst(filename, find) fsm_findfirst(filename, find)`
Find the first file or subdirectory in a specified directory.
- `#define fs_findnext(find) fsm_findnext(find)`
Finds the next file or subdirectory in a specified directory after a previous call to `fs_findfirst()` or `fs_findnext()`.
- `#define fs_open(filename, mode) fsm_open(filename, mode)`
Opens a file in the file system.
- `#define fs_close(filehandle) fsm_close(filehandle)`
Closes an opened file.
- `#define fs_write(buf, size, size_st, filehandle) fsm_write(buf, size, size_st, filehandle)`
Write data to the file at the current position.
- `#define fs_read(buf, size, size_st, filehandle) fsm_read(buf, size, size_st, filehandle)`
Read data from the current position in a file.
- `#define fs_seek(filehandle, offset, whence) fsm_seek(filehandle, offset, whence)`
Move the stream position of an open file.
- `#define fs_eof(filehandle) fsm_eof(filehandle)`
Check whether the current position in the open target file is the end of the file.
- `#define fs_rewind(filehandle) fsm_rewind(filehandle)`
Sets the file position in the open target file to the start of the file.
- `#define fs_settimedate(filename, ctime, cdate) fsm_settimedate(filename, ctime, cdate)`
Set the time and date of a file or directory.
- `#define fs_gettimedate(filename, pctime, pdate) fsm_gettimedate(filename, pctime, pdate)`
Get the time and date of a file or directory.

14.14.2.1 Detailed Description

The Standard Embedded Flash File System, EFFS-STD

```
#include< fsf.h>
```

14.14.2.2 Macro Definition Documentation

14.14.2.2.1 fs_chdir

```
#define fs_chdir(  
    dirname) fsm_chdir(dirname)  
#include <fsf.h>  
Change the directory.
```

Parameters

<i>dirname</i>	The directory to change to.

Return values

<i>FS_NOERR</i>	If able to change to the directory
<i>IFS_NOERR</i>	If unable to change to the directory

See also

[fs_mkdir\(\)](#)

[fs_rmdir\(\)](#)

14.14.2.2.2 fs_close

```
#define fs_close(  
    filehandle) fsm_close(filehandle)
```

```
#include <fsf.h>
```

Closes an opened file.

Parameters

<i>filehandle</i>	A handle to the file to close.

Return values

<i>FS_NOERR</i>	If able to close the file
<i>IFS_NOERR</i>	If unable to close the file

14.14.2.2.3 fs_delete

```
#define fs_delete(  
    filename) fsm_delete(filename)
```

```
#include <fsf.h>
```

Deletes a file.

A read-only or open file cannot be deleted.

Parameters

<i>filename</i>	The name of the file to be deleted.

Return values

<i>FS_NOERR</i>	If able to delete the file
<i>IFS_NOERR</i>	If unable to delete the file

14.14.2.2.4 fs_eof

```
#define fs_eof(  
    filehandle) fsm_eof(filehandle)
```

```
#include <fsf.h>
```

Check whether the current position in the open target file is the end of the file.

Parameters

<i>filehandle</i>	A handle to the file to check.

Return values

<i>0</i>	Nothing at the end of the file
<i>!0</i>	The end of the file, or an error

14.14.2.2.5 fs_findfirst

```
#define fs_findfirst(  
    filename,  
    find) fsm_findfirst(filename, find)  
#include <fsf.h>
```

Find the first file or subdirectory in a specified directory.

Note: If this is called with "*" and this is not the root directory, the first entry found will be ".", which is the current directory.

Parameters

<i>filename</i>	The name of the file to find.
<i>find</i>	Where to store the found information.

Return values

<i>FS_NOERR</i>	If the file was found
<i>IFS_NOERR</i>	If the file wasn't found or there are errors

See also

[fs_findnext\(\)](#)

14.14.2.2.6 fs_findnext

```
#define fs_findnext(  
    find) fsm_findnext(find)  
#include <fsf.h>
```

Finds the next file or subdirectory in a specified directory after a previous call to [fs_findfirst\(\)](#) or [fs_findnext\(\)](#).

Note: If this is called with "*" and this is not the root directory, the first entry found will be ".", which is the current directory.

Parameters

<i>find</i>	Where to store the found information.

Return values

<i>FS_NOERR</i>	If the file was found
<i>IFS_NOERR</i>	If the file wasn't found or there are errors

See also

[fs_findfirst\(\)](#)

14.14.2.2.7 fs_getfreespace

```
#define fs_getfreespace(  
    drivenum,  
    space) fsm_getfreespace(drivenum, space)  
#include <fsf.h>
```

Provides information about the drive space usage.

Parameters

<i>drivenum</i>	The drive number to get space usage for.
<i>space</i>	The structure to store the usage information in.

Return values

<i>FS_NOERR</i>	If able to get the drive space usage
<i>IFS_NOERR</i>	If unable to get the drive space usage

14.14.2.2.8 fs_gettimedate

```
#define fs_gettimedate(  
    filename,  
    pctime,  
    pcdte) fsm_gettimedate(filename, pctime, pcdte)
```

```
#include <fsf.h>
```

Get the time and date of a file or directory.

Parameters

<i>filename</i>	The name of the file or directory to modify.
<i>pctime</i>	A pointer to store the creation time of the file or directory.
<i>pcdate</i>	A pointer to store the creation date of the file or directory.

Return values

<i>FS_NOERR</i>	If able to get time and date
<i>IFS_NOERR</i>	If unable to get time and date

14.14.2.2.9 fs_mkdir

```
#define fs_mkdir(  
    dirname) fsm_mkdir(dirname)
```

```
#include <fsf.h>
```

Makes a new directory.

Parameters

<i>dirname</i>	The name to use for the new directory.

Return values

<i>FS_NOERR</i>	If able to make the directory
<i>IFS_NOERR</i>	If unable to make the directory

See also

[fs_chdir\(\)](#)

[fs_rmdir\(\)](#)

14.14.2.2.10 fs_open

```
#define fs_open(
    filename,
    mode) fsm_open(filename, mode)
#include <fsf.h>
Opens a file in the file system.
```

Parameters

<i>filename</i>	The name of the file to open.
<i>mode</i>	The mode to open the file in. The following options are available: "r": Reading. The stream is positioned at the beginning of the file. "r+": Reading and writing. The stream is positioned at the beginning of the file. "w": Truncate file to 0 length or create for writing. The stream is positioned at the beginning of the file. "w+": Reading and writing. Create if it doesn't exist, or truncate if it does. The stream is positioned at the beginning of the file. "a": Append, and create if it doesn't exist. The stream is positioned at the end of the file. "a+": Reading and appending, and create if it doesn't exist. The stream is positioned at the end of the file.

Return values

<i>FS_FILE*</i>	Pointer to the file if it could be opened
<i>0</i>	If it a file could not be opened

See also

[fs_close\(\)](#)

14.14.2.2.11 fs_read

```
#define fs_read(
    buf,
    size,
    size_st,
    filehandle) fsm_read(buf, size, size_st, filehandle)
#include <fsf.h>
Read data from the current position in a file.
```

Parameters

<i>buf</i>	A pointer where the data should be read.
<i>size</i>	The size of the items to be read.
<i>size_st</i>	The number of items to be read.
<i>filehandle</i>	A handle to the file to read to.

Returns

The number of items read.

14.14.2.2.12 fs_rewind

```
#define fs_rewind(  
    filehandle) fsm_rewind(filehandle)  
#include <fsf.h>  
Sets the file position in the open target file to the start of the file.
```

Parameters

<i>filehandle</i>	A handle to the file operate on.

Return values

<i>FS_NOERR</i>	If successful
<i>IFS_NOERR</i>	If there was an error

14.14.2.2.13 fs_rmdir

```
#define fs_rmdir(  
    dirname) fsm_rmdir(dirname)  
#include <fsf.h>  
Removes a directory.  
Note: The target directory must be empty, otherwise an error code is returned.
```

Parameters

<i>dirname</i>	The name of the directory to remove.

Return values

<i>FS_NOERR</i>	If able to get the drive space usage
<i>IFS_NOERR</i>	If unable to get the drive space usage

See also

[fs_chdir\(\)](#)
[fs_mkdir\(\)](#)

14.14.2.2.14 fs_seek

```
#define fs_seek(  
    filehandle,  
    offset,  
    whence) fsm_seek(filehandle, offset, whence)  
#include <fsf.h>  
Move the stream position of an open file.
```

Parameters

<i>filehandle</i>	A handle to the file to move the position of.
<i>offset</i>	The relative byte position according to whence.
<i>whence</i>	Where to calculate the offset from. Can be a value from fSeekCodeGroup.

Return values

<i>FS_NOERR</i>	If the seek operation was successful
<i>IFS_NOERR</i>	If there was an error

14.14.2.2.15 fs_settimedate

```
#define fs_settimedate(
    filename,
    ctime,
    cdate) fsm_settimedate(filename, ctime, cdate)
```

```
#include <fsf.h>
```

Set the time and date of a file or directory.

Parameters

<i>filename</i>	The name of the file or directory to modify.
<i>ctime</i>	The creation time of the file or directory.
<i>cdate</i>	The creation date of the file or directory.

Return values

<i>FS_NOERR</i>	If able to set time and date
<i>IFS_NOERR</i>	If unable to set time and date

14.14.2.2.16 fs_write

```
#define fs_write(
    buf,
    size,
    size_st,
    filehandle) fsm_write(buf, size, size_st, filehandle)
```

```
#include <fsf.h>
```

Write data to the file at the current position.

Parameters

<i>buf</i>	A pointer to the data to write.
<i>size</i>	The size of the items to write.
<i>size_st</i>	The number of items to write.
<i>filehandle</i>	A handle to the file to write to.

Returns

The number of items written.

14.14.2.3 STD File System Seek Codes

Macros

- **#define FS_SEEK_SET 0**
Beginning of file.
- **#define FS_SEEK_CUR 1**
Current position of the file pointer.
- **#define FS_SEEK_END 2**
End of file.

14.14.2.3.1 Detailed Description

```
#include< fsf.h>
```

The codes used when calling `f_seek()`.

14.14.3 FAT File System

The FAT32 Embedded Flash File System for Flash Cards, EFFS-FAT

```
#include< api_f.h>
```

Topics

- [FAT File System Seek Codes](#)

Macros

- `#define f_delvolume(drvnumber) fm_delvolume(drvnumber)`
Un-mounts a flash card.
- `#define f_getfreespace(drivenum, pspace) fm_getfreespace(drivenum, pspace)`
Provides information about the drive space usage.
- `#define f_chdir(dirname) fm_chdir(dirname)`
Change the directory.
- `#define f_mkdir(dirname) fm_mkdir(dirname)`
Makes a new directory.
- `#define f_rmdir(dirname) fm_rmdir(dirname)`
Removes a directory.
- `#define f_findfirst(filename, find) fm_findfirst(filename, find)`
Find the first file or subdirectory in a specified directory.
- `#define f_findnext(find) fm_findnext(find)`
Finds the next file or subdirectory in a specified directory after a previous call to `f_findfirst()` or `f_findnext()`.
- `#define f_close(filehandle) fm_close(filehandle)`
Closes an opened file.
- `#define f_open(filename, mode) fm_open(filename, mode)`
Opens a file in the file system.
- `#define f_read(buf, size, size_st, filehandle) fm_read(buf, size, size_st, filehandle)`
Read data from the current position in a file.
- `#define f_write(buf, size, size_st, filehandle) fm_write(buf, size, size_st, filehandle)`
Write data to the file at the current position.
- `#define f_seek(filehandle, offset, whence) fm_seek(filehandle, offset, whence)`
Move the stream position of an open file.
- `#define f_rewind(filehandle) fm_rewind(filehandle)`
Sets the file position in the open target file to the start of the file.
- `#define f_eof(filehandle) fm_eof(filehandle)`
Check whether the current position in the open target file is the end of the file.
- `#define f_gettimedate(filename, pctime, pctime) fm_gettimedate(filename, pctime, pctime)`
Get the time and date of a file or directory.
- `#define f_settimedate(filename, ctime, cdate) fm_settimedate(filename, ctime, cdate)`
Set the time and date of a file or directory.
- `#define f_delete(filename) fm_delete(filename)`
Deletes a file.

Functions

- int [f_enterFS](#) (void)
Adds a new task priority to the task list used by the file system.
- void [f_releaseFS](#) (void)
Removes a task priority from the task list used by the file system.

14.14.3.1 Detailed Description

The FAT32 Embedded Flash File System for Flash Cards, EFFS-FAT
`#include < api_f.h>`

14.14.3.2 Macro Definition Documentation

14.14.3.2.1 [f_chdir](#)

```
#define f_chdir(  
    dirname) fm_chdir(dirname)  
#include <api_f.h>  
Change the directory.
```

Parameters

<i>dirname</i>	The directory to change to.

Return values

<i>F_NO_ERROR</i>	If able to change to the directory
<i>!F_NO_ERROR</i>	If unable to change to the directory

See also

[f_mkdir\(\)](#)
[f_rmdir\(\)](#)

14.14.3.2.2 [f_close](#)

```
#define f_close(  
    filehandle) fm_close(filehandle)  
#include <api_f.h>  
Closes an opened file.
```

Parameters

<i>filehandle</i>	A handle to the file to close.

Return values

<i>F_NO_ERROR</i>	If able to close the file
<i>!F_NO_ERROR</i>	If unable to close the file

14.14.3.2.3 f_delete

```
#define f_delete(  
    filename) fm_delete(filename)
```

```
#include <api_f.h>
```

Deletes a file.

A read-only or open file cannot be deleted.

Parameters

<i>filename</i>	The name of the file to be deleted.

Return values

<i>F_NO_ERROR</i>	If able to delete the file
<i>!F_NO_ERROR</i>	If unable to delete the file

14.14.3.2.4 f_delvolume

```
#define f_delvolume(  
    drvnumber) fm_delvolume(drvnumber)
```

```
#include <api_f.h>
```

Un-mounts a flash card.

Parameters

<i>drvnumber</i>	The drive to be un-mount.

14.14.3.2.5 f_eof

```
#define f_eof(  
    filehandle) fm_eof(filehandle)
```

```
#include <api_f.h>
```

Check whether the current position in the open target file is the end of the file.

Parameters

<i>filehandle</i>	A handle to the file to check.

Return values

<i>0</i>	Nothing at the end of the file
<i>!0</i>	The end of the file, or an error

14.14.3.2.6 f_findfirst

```
#define f_findfirst(  
    filename,  
    find) fm_findfirst(filename, find)
```

```
#include <api_f.h>
```

Find the first file or subdirectory in a specified directory.

Note: If this is called with "*. *" and this is not the root directory, the first entry found will be ".", which is the current directory.

Parameters

<i>filename</i>	The name of the file to find.
<i>find</i>	Where to store the found information.

Return values

<i>F_NO_ERROR</i>	If the file was found
<i>!F_NO_ERROR</i>	If the file wasn't found or there are errors

See also

[f_findnext\(\)](#)**14.14.3.2.7 f_findnext**

```
#define f_findnext(  
    find) fm_findnext(find)  
#include <api_f.h>
```

Finds the next file or subdirectory in a specified directory after a previous call to [f_findfirst\(\)](#) or [f_findnext\(\)](#).

Note: If this is called with " *.* " and this is not the root directory, the first entry found will be ".", which is the current directory.

Parameters

<i>find</i>	Where to store the found information.

Return values

<i>F_NO_ERROR</i>	If the file was found
<i>!F_NO_ERROR</i>	If the file wasn't found or there are errors

See also

[f_findfirst\(\)](#)**14.14.3.2.8 f_getfreespace**

```
#define f_getfreespace(  
    drivenum,  
    pspace) fm_getfreespace(drivenum, pspace)  
#include <api_f.h>
```

Provides information about the drive space usage.

Parameters

<i>drivenum</i>	The drive number to get space usage for.
<i>pspace</i>	The structure to store the usage information in.

Return values

<i>F_NO_ERROR</i>	If able to get the drive space usage
<i>!F_NO_ERROR</i>	If unable to get the drive space usage

14.14.3.2.9 f_gettimedate

```
#define f_gettimedate(
    filename,
    pctime,
    pctime) fm_gettimedate(filename, pctime, pctime)
#include <api_f.h>
```

Get the time and date of a file or directory.

Parameters

	<i>filename</i>	The name of the file or directory to modify.
out	<i>pctime</i>	A pointer to store the creation time of the file or directory.
out	<i>pctime</i>	A pointer to store the creation date of the file or directory.

Return values

<i>F_NO_ERROR</i>	If able to get time and date
<i>IF_NO_ERROR</i>	If unable to get time and date

14.14.3.2.10 f_mkdir

```
#define f_mkdir(
    dirname) fm_mkdir(dirname)
#include <api_f.h>
```

Makes a new directory.

Parameters

<i>dirname</i>	The name to use for the new directory.

Return values

<i>F_NO_ERROR</i>	If able to make the directory
<i>IF_NO_ERROR</i>	If unable to make the directory

See also

[f_chdir\(\)](#)

[f_rmdir\(\)](#)

14.14.3.2.11 f_open

```
#define f_open(
    filename,
    mode) fm_open(filename, mode)
#include <api_f.h>
```

Opens a file in the file system.

Parameters

<i>filename</i>	The name of the file to open.

Parameters

<i>mode</i>	The mode to open the file in. The following options are available: • "r": Reading. The stream is positioned at the beginning of the file. • "r+": Reading and writing. The stream is positioned at the beginning of the file. • "w": Truncate file to 0 length or create for writing. The stream is positioned at the beginning of the file. • "w+": Reading and writing. Create if it doesn't exist, or truncate if it does. The stream is positioned at the beginning of the file. • "a": Append, and create if it doesn't exist. The stream is positioned at the end of the file. • "a+": Reading and appending, and create if it doesn't exist. The stream is positioned at the end of the file.

Return values

<i>F_FILE*</i>	Pointer to the file if it could be opened
<i>0</i>	If it a file could not be opened

See also

[f_close\(\)](#)**14.14.3.2.12 f_read**

```
#define f_read(  
    buf,  
    size,  
    size_st,  
    filehandle) fm_read(buf, size, size_st, filehandle)  
#include <api_f.h>  
Read data from the current position in a file.
```

Parameters

<i>buf</i>	A pointer where the data should be read.
<i>size</i>	The size of the items to be read.
<i>size_st</i>	The number of items to be read.
<i>filehandle</i>	A handle to the file to read to.

Returns

The number of items read.

14.14.3.2.13 f_rewind

```
#define f_rewind(  
    filehandle) fm_rewind(filehandle)  
#include <api_f.h>  
Sets the file position in the open target file to the start of the file.
```

Parameters

<i>filehandle</i>	A handle to the file operate on.

Return values

<i>F_NO_ERROR</i>	If successful
<i>!F_NO_ERROR</i>	If there was an error

14.14.3.2.14 f_rmdir

```
#define f_rmdir(  
    dirname) fm_rmdir(dirname)  
#include <api_f.h>
```

Removes a directory.

Note: The target directory must be empty, otherwise an error code is returned.

Parameters

<i>dirname</i>	The name of the directory to remove.

Return values

<i>F_NO_ERROR</i>	If able to remove the directory
<i>!F_NO_ERROR</i>	If unable to remove the directory

See also

[f_chdir\(\)](#)

[f_mkdir\(\)](#)

14.14.3.2.15 f_seek

```
#define f_seek(  
    filehandle,  
    offset,  
    whence) fm_seek(filehandle, offset, whence)  
#include <api_f.h>
```

Move the stream position of an open file.

Parameters

<i>filehandle</i>	A handle to the file to move the position of.
<i>offset</i>	The relative byte position according to whence.
<i>whence</i>	Where to calculate the offset from. Can be a value from FAT File System Seek Codes .

Return values

<i>F_NO_ERROR</i>	If the seek operation was successful
<i>!F_NO_ERROR</i>	If there was an error

14.14.3.2.16 f_settimedate

```
#define f_settimedate(
    filename,
    ctime,
    cdate) fm_settimedate(filename, ctime, cdate)
```

```
#include <api_f.h>
```

Set the time and date of a file or directory.

Parameters

<i>filename</i>	The name of the file or directory to modify.
<i>ctime</i>	The creation time of the file or directory.
<i>cdate</i>	The creation date of the file or directory.

Return values

<i>F_NO_ERROR</i>	If able to set time and date
<i>!F_NO_ERROR</i>	If unable to set time and date

14.14.3.2.17 f_write

```
#define f_write(
    buf,
    size,
    size_st,
    filehandle) fm_write(buf, size, size_st, filehandle)
```

```
#include <api_f.h>
```

Write data to the file at the current position.

Parameters

<i>buf</i>	A pointer to the data to write.
<i>size</i>	The size of the items to write.
<i>size_st</i>	The number of items to write.
<i>filehandle</i>	A handle to the file to write to.

Returns

The number of items written.

14.14.3.3 Function Documentation

14.14.3.3.1 f_enterFS()

```
int f_enterFS (
    void ) [extern]
```

```
#include <api_f.h>
```

Adds a new task priority to the task list used by the file system.

This function needs to be called when a new task wants to use the file system, before any other calls to other file system functions. Up to 10 tasks can be assigned to use the file system at any given time.

Note: The task priority level is associated with the task list, not the task itself. This means that a task switched priority levels and still wants to use the file system, it will need to be called again.

Return values

0	If successful

<i>!O</i>	If there were errors

See also

[f_releaseFS\(\)](#)

14.14.3.3.2 f_releaseFS()

```
void f_releaseFS (
    void ) [extern]
#include <api_f.h>
```

Removes a task priority from the task list used by the file system.

When a task no longer needs to use the file system, or prior to task changing priorities, this should be called. In the case of a task changing priorities, if access to the file system is still desired, another call to [f_enterFS\(\)](#) should be made.

See also

[f_releaseFS\(\)](#)

14.14.3.4 FAT File System Seek Codes

Macros

- `#define F_SEEK_SET FN_SEEK_SET`
Beginning of file.
- `#define F_SEEK_END FN_SEEK_END`
End of file.
- `#define F_SEEK_CUR FN_SEEK_CUR`
Current position of the file pointer.

14.14.3.4.1 Detailed Description

```
#include< api_f.h>
```

The codes used when calling [f_seek\(\)](#).

14.15 Ethernet

Topics

- [Ethernet I/O Counters](#)

Classes

- struct [_EthernetFrameHeader](#)
Ethernet Type II Frame Header.

Macros

- `#define NO_AUTOMATIC_2ND_ETHERNET` extern const bool bAutomatic2ndEther = false;
Disable automatic initialization of second Ethernet interface.

Typedefs

- typedef struct [_EthernetFrameHeader](#) **EthernetFrameHeader**
Ethernet Type II Frame Header.

Functions

- void [AddEthernetInterfaces](#) ()
Add an Ethernet interface.
- void [ManualEthernetConfig](#) (int interface, BOOL speed100Mbit, BOOL fullDuplex, BOOL autoNegotiate)
Manually configure Ethernet speed and duplex settings.
- void [DisablePHY](#) (int ifn)
Disable the specified Ethernet PHY.
- void [EnablePHY](#) (int ifn)
Enable the specified Ethernet PHY.

Ethernet Interface Types

EtherType Field (Ethernet Version II)

- #define **ETHERNET_ETHERTYPE_IPV4** (uint16_t)(0x0800)
Internet Protocol, Version 4 (IPv4)
- #define **ETHERNET_ETHERTYPE_ARP** (uint16_t)(0x0806)
Address Resolution Protocol ([ARP](#))
- #define **ETHERNET_ETHERTYPE_IPV6** (uint16_t)(0x86DD)
Internet Protocol, Version 6 (IPv6)
- #define **ETHERNET_ETHERTYPE_AARP** (uint16_t)(0x80F3)
AppleTalk Address Resolution Prot. (AARP)
- #define **ETHERNET_ETHERTYPE_IPX** (uint16_t)(0x8137)
Novell Internet Packet Exchange (IPX) (alt.)
- #define **ETHERNET_ETHERTYPE_EAPOL** (uint16_t)(0x888E)
Extensible Authorization Protocol (EAP) over LAN.
- #define **ETHERNET_ETHERTYPE_VLAN** (uint16_t)(0x8100)
Virtual Private Network (VLAN)

14.15.1 Detailed Description

```
#include< ethernet.h>
```

Functions to add, configure and disable Ethernet interfaces

14.15.2 Macro Definition Documentation

14.15.2.1 NO_AUTOMATIC_2ND_ETHERNET

```
#define NO_AUTOMATIC_2ND_ETHERNET extern const bool bAutomatic2ndEther = false;
#include <ethernet.h>
```

Disable automatic initialization of second Ethernet interface.

Include this macro in UserMain to prevent automatic system addition and initialization of the second Ethernet interface on dual Ethernet devices.

14.15.3 Function Documentation

14.15.3.1 AddEthernetInterfaces()

```
void AddEthernetInterfaces ()
#include <ethernet.h>
```

Add an Ethernet interface.

The default system behavior is all Ethernet interfaces will automatically be added and initialized. Interface numbers range from 1 to (MAX_INTERFACES - 1)

14.15.3.2 DisablePHY()

```
void DisablePHY (
    int ifn)
#include <ethernet.h>
Disable the specified Ethernet PHY.
Typically used for low power mode.
```

Parameters

<i>ifn</i>	Interface number to disable

See also

[EnablePHY](#)

14.15.3.3 EnablePHY()

```
void EnablePHY (
    int ifn)
#include <ethernet.h>
Disable the specified Ethernet PHY.
Enable an Ethernet PHY previously disabled by DisablePHY\(\)
```

Parameters

<i>ifn</i>	Interface number to enable

See also

[DisablePHY\(\)](#)

14.15.3.4 ManualEthernetConfig()

```
void ManualEthernetConfig (
    int interface,
    BOOL speed100Mbit,
    BOOL fullDuplex,
    BOOL autoNegotiate)
#include <ethernet.h>
```

Manually configure Ethernet speed and duplex settings.

The default setting to establish an Ethernet link is autonegotiate. This function can be used to disable autonegotiate and set the Ethernet link speed and duplex.

Warning

With autonegotiate disabled, the other Ethernet host must also be configured manually with the same settings. If the other host is set to autonegotiate, the link will have undefined behavior and data loss.

Parameters

<i>interface</i>	The network interface to modify. The first interface number is 1
<i>speed100Mbit</i>	True = 100Mbps, False = 10Mbps
<i>fullDuplex</i>	True = full duplex, False = half duplex
<i>autoNegotiate</i>	True = enabled, False = disabled. If enabled, will override manual settings.

14.15.4 Ethernet I/O Counters

Data counters kept by the TCP/IP system

- volatile uint32_t **frames_tx**
"tx:%u", "tx:%d",
- volatile uint32_t **frames_rx**
"rx:%u", "rx:%d",
- volatile uint32_t **frames_tx_err**
"tx_err:%u", "tx_err:%d",
- volatile uint32_t **frames_rx_err**
"rx_err:%u", "rx_err:%d",
- volatile uint32_t **frames_rx_discard**
"rx_err:%u", "rx_err:%d",
- volatile uint32_t **frames_rx_arp**
"rx_arp:%u", "rx_arp:%d",
- volatile uint32_t **frames_tx_arp**
"tx_arp:%u", "tx_arp:%d",
- volatile uint32_t **frames_tx_udp**
"tx_udp:%u", "tx_udp:%d",
- volatile uint32_t **frames_rx_udp**
"rx_udp:%u", "rx_udp:%d",
- volatile uint32_t **frames_tx_tcp**
"tx_tcp:%u", "tx_tcp:%d",
- volatile uint32_t **frames_rx_tcp**
"rx_tcp:%u", "rx_tcp:%d",
- volatile uint32_t **frames_tx_icmp**
"tx_icmp:%u", "tx_icmp:%d",
- volatile uint32_t **frames_rx_icmp**
"rx_icmp:%u", "rx_icmp:%d",
- volatile uint32_t **frames_ip_errors**
"ip_err:%u", "ip_err:%d",
- volatile uint32_t **frames_ip_discard**
"ip_discard:%u", "ip_discard:%d",
- volatile uint32_t **frames_udp_errors**
"udp_err:%u", "udp_err:%d",
- volatile uint32_t **frames_tcp_errors**
"tcp_err:%u", "tcp_err:%d",
- volatile uint32_t **enet_last_errhw**
"last_err:%u""last_err:%d"
- volatile uint32_t **enet_last_errlw**
"last_err:%u""last_err:%d"
- volatile uint32_t **ip_last_err**
"last_err:%u""last_err:%d"
- volatile uint32_t **enet_isr**
"enet_isr:%u""enet_isr:%d"
- volatile uint32_t **frames_rx_unknown**
"frames_rx_unknown:%u""frames_rx_unknown%d"
- volatile uint32_t **frames_rx_ppp_errors**
- volatile uint32_t **frames_rx_fragment**

14.15.4.1 Detailed Description

```
#include< counters.h>
```

14.16 Extra File Descriptors

Functions

- int [GetExtraFD](#) (void *extra_data, struct IoExpandStruct *pFuncs)
Returns a file descriptor for the structure passed as the IoExpandStruct. [FreeExtraFd\(\)](#) will release the fd back to the pool of available fds.
- IoExpandStruct * [GetIoExpandStruct](#) (int fd)
Returns the pointer to the IoExpandStruct associated with the file descriptor.
- void * [GetExtraData](#) (int fd)
Returns the extra structure value from IoExpandStruct associated with the file descriptor.
- void [FreeExtraFd](#) (int fd)
Free a file descriptor and associated resources.
- int [GetFreeExtraFDCount](#) ()
Returns the number of free file descriptors.
- int [GetFreeSocketCount](#) (void)
Returns the number of free sockets.

14.16.1 Detailed Description

```
#include< iointernal.h>
```

The NetBurner environment integrates the RTOS, TCP/IP stack, and other peripherals with a file I/O system based on file descriptors. A file descriptor can be described as a handle to a network socket, serial port, system peripheral, or any other object that can be read or written to. Most of the API functions pass a file descriptor as a parameter to such an object. These function are available if you wish to use the file descriptor system capabilities for your own application purposes. Please refer to the File Descriptor section in the Programmers Guide.

14.16.2 Function Documentation

14.16.2.1 FreeExtraFd()

```
void FreeExtraFd (
    int fd)
#include <iointernal.h>
Free a file descriptor and associated resources.
```

Parameters

<i>fd</i>	File descriptor

14.16.2.2 GetExtraData()

```
void * GetExtraData (
    int fd)
#include <iointernal.h>
Returns the extra structure value from IoExpandStruct associated with the file descriptor.
```

Parameters

<i>fd</i>	File descriptor

Return values

<i>Value</i>	of the extra member of the structure

14.16.2.3 GetExtraFD()

```
int GetExtraFD (
    void * extra_data,
    struct IoExpandStruct * pFuncs)
#include <iointernal.h>
```

Returns a file descriptor for the structure passed as the `IoExpandStruct`. [FreeExtraFd\(\)](#) will release the fd back to the pool of available fds.

Parameters

<i>extra_data</i>	The optional extra_data void pointer can be used to pass data into the file descriptor
<i>*pFuncs</i>	Structure containing callback functions for such things as read, write, close, and the extra_data variable

Return values

<i>Value</i>	greater than 0, representing the file descriptor on success
<i>-1</i>	on failure

14.16.2.4 GetFreeExtraFDCount()

```
int GetFreeExtraFDCount ()
#include <iointernal.h>
```

Returns the number of free file descriptors.

Return values

<i>Number</i>	of free extra file descriptors

14.16.2.5 GetFreeSocketCount()

```
int GetFreeSocketCount (
    void )
#include <iointernal.h>
```

Returns the number of free sockets.

Return values

<i>Number</i>	of free sockets

14.16.2.6 GetIoExpandStruct()

```
IoExpandStruct * GetIoExpandStruct (  
    int fd)
```

```
#include <iointernal.h>
```

Returns the pointer to the `IoExpandStruct` associated with the file descriptor.

Parameters

<i>fd</i>	File descriptor

Return values

<i>Pointer</i>	to the the <code>IoExpandStruct</code> associated with the file descriptor.

14.17 FTP Client

Functions

- int [FTP_InitializeSession](#) (IPADDR4 server_address, uint16_t port, PCSTR UserName, PCSTR Password, uint32_t time_out)
Initialize a FTP session with a FTP server.
- int [FTP_CloseSession](#) (int session)
Close a FTP session.
- int [FTPGetDir](#) (int ftp_Session, char *dir_buf, int nbytes, uint16_t timeout)
Get the current working directory.
- int [FTPSetDir](#) (int ftp_Session, const char *new_dir, uint16_t timeout)
Set the current working directory.
- int [FTPDeleteDir](#) (int ftp_Session, const char *dir_to_delete, uint16_t timeout)
Delete a directory.
- int [FTPMakeDir](#) (int ftp_Session, const char *dir_to_make, uint16_t timeout)
Create a new directory.
- int [FTPUpDir](#) (int ftp_Session, uint16_t timeout)
Move up one directory level.
- int [FTPDeleteFile](#) (int ftp_Session, const char *file_name, uint16_t timeout)
Delete a file.
- int [FTPRenameFile](#) (int ftp_Session, const char *old_file_name, const char *new_file_name, uint16_t time-out)
Rename a file.
- int [FTPSendFile](#) (int ftp_Session, const char *full_file_name, BOOL bBinaryMode, uint16_t timeout)
Initialize the process to send a file to s FTP server.
- int [FTPGetFile](#) (int ftp_Session, const char *full_file_name, BOOL bBinaryMode, uint16_t timeout)
Initialize the process to get a file from a FTP server.
- int [FTPGetList](#) (int ftp_Session, const char *full_dir_name, uint16_t timeout)
Initialize the process to receive a directory listing from a FTP server.
- int [FTPGetFileNames](#) (int ftp_Session, const char *full_dir_name, uint16_t timeout)
Initialize the process to receive just the file names in a directory from a FTP server.
- int [FTPRawCommand](#) (int ftp_Session, const char *cmd, char *cmd_buf, int nbytes, uint16_t timeout)
Send a FTP command to the FTP server.
- int [FTPGetCommandResult](#) (int ftp_Session, char *cmd_buf, int nbytes, uint16_t timeout)
Returns the result of the last FTP operation.
- int [FTPRawStreamCommand](#) (int ftp_Session, const char *cmd, int *pResult, char *cmd_buf, int nbytes, uint16_t timeout)
Send a command and receive a response over a stream connection.
- void [FTPActiveMode](#) (int ftp_Session)
Set mode to active.
- void [FTPPassiveMode](#) (int ftp_Session)
Set mode to passive.

FTP Client Return Codes

- `#define FTP_OK (0)`
OK.
- `#define FTP_TIMEOUT (-1)`
Timeout.
- `#define FTP_PASSWORDERROR (-2)`
Password error.
- `#define FTP_CONNECTFAIL (-3)`
Connection failed.
- `#define FTP_COMMANDFAIL (-4)`
Command failed.
- `#define FTP_COMMANDERROR (-4)`
Command error.
- `#define FTP_BADSESSION (-5)`
Bad session.
- `#define FTP_NETWORKERROR (-6)`
Network error.

14.17.1 Detailed Description

```
#include< ftp.h>
```

The NetBurner FTP Client Group

14.17.2 Function Documentation**14.17.2.1 FTP_CloseSession()**

```
int FTP_CloseSession (
    int session)
```

```
#include <ftp.h>
```

Close a FTP session.

Parameters

<i>session</i>	FTP session number to close

Returns

[ftpClientReturnCodes](#)

See also

[FTP_InitializeSession\(\)](#)

14.17.2.2 FTP_InitializeSession()

```
int FTP_InitializeSession (
    IPADDR4 server_address,
    uint16_t port,
    PCSTR UserName,
    PCSTR Password,
    uint32_t time_out)
```

```
#include <ftp.h>
```

Initialize a FTP session with a FTP server.

Open a connection to a FTP server and log in with the specified username and password. The session number returned from this call can be used with the file and directory functions.

Parameters

<i>server_address</i>	The IP address of the FTP server
<i>port</i>	The listen port number of the FTP server. Typically 21.
<i>UserName</i>	Log-in username
<i>Password</i>	Log-in password
<i>time_out</i>	Timeout in system Time Ticks

Returns

A value greater than 0 is a FTP session handle, otherwise [ftpClientReturnCodes](#)

See also

[FTP_CloseSession\(\)](#)

14.17.2.3 FTPActiveMode()

```
void FTPActiveMode (
    int ftp_Session)
#include <ftp.h>
Set mode to active.
```

Parameters

<i>ftp_Session</i>	FTP session number

14.17.2.4 FTPDeleteDir()

```
int FTPDeleteDir (
    int ftp_Session,
    const char * dir_to_delete,
    uint16_t timeout)
#include <ftp.h>
Delete a directory.
```

Parameters

<i>ftp_Session</i>	FTP session number
<i>dir_to_delete</i>	Directory to delete
<i>timeout</i>	Timeout in system Time Ticks

Returns

[ftpClientReturnCodes](#)

14.17.2.5 FTPDeleteFile()

```
int FTPDeleteFile (
    int ftp_Session,
    const char * file_name,
    uint16_t timeout)
#include <ftp.h>
Delete a file.
```


Parameters

<i>ftp_Session</i>	FTP session number
<i>file_name</i>	Name of file to delete
<i>timeout</i>	Timeout in system Time Ticks

Returns

[ftpClientReturnCodes](#)

14.17.2.6 FTPGetCommandResult()

```
int FTPGetCommandResult (
    int ftp_Session,
    char * cmd_buf,
    int nbytes,
    uint16_t timeout)
```

```
#include <ftp.h>
```

Returns the result of the last FTP operation.

Queries the FTP server for the last FTP operation status.

Must be called after the following functions are used:

- [FTPGetList\(\)](#)
- [FTPFileNames\(\)](#)
- [FTPGetFile\(\)](#)
- [FTPSendFile\(\)](#)

Parameters

<i>ftp_Session</i>	FTP session number
<i>cmd_buf</i>	The buffer to hold the ASCII FTP server response. See FTP Server return codes for details.
<i>nbytes</i>	The maximum number of bytes cmd_buf can hold
<i>timeout</i>	Timeout in system Time Ticks

Returns

The FTP server response number on success, otherwise [ftpClientReturnCodes](#)

14.17.2.7 FTPGetDir()

```
int FTPGetDir (
    int ftp_Session,
    char * dir_buf,
    int nbytes,
    uint16_t timeout)
```

```
#include <ftp.h>
```

Get the current working directory.

Parameters

<i>ftp_Session</i>	FTP session number
<i>dir_buf</i>	The buffer to copy the directory information into
<i>nbytes</i>	Maximum number of bytes in the dir_buf
<i>timeout</i>	Timeout in system Time Ticks

Returns

[ftpClientReturnCodes](#)

14.17.2.8 FTPGetFile()

```
int FTPGetFile (
    int ftp_Session,
    const char * full_file_name,
    BOOL bBinaryMode,
    uint16_t timeout)
```

```
#include <ftp.h>
```

Initialize the process to get a file from a FTP server.

This function opens a TCP connection to a FTP server so that a file may be read. it will return a file descriptor that can then be used to read the file data using file I/O functions such as: [read\(\)](#) and [ReadWithTimeout\(\)](#).

IMPORTANT After reading the file data you must do 2 things:

- The TCP connection must be closed with the [close\(\)](#) function.
- [FTPGetCommandResult\(\)](#) must be called to check the result of the operation.

Parameters

<i>ftp_Session</i>	FTP session number
<i>full_file_name</i>	Full file name being read, including the path
<i>bBinaryMode</i>	True = read file as binary data, False = read file as ASCII data
<i>timeout</i>	Timeout in system Time Ticks

Returns

TCP file descriptor if greater than 0, otherwise [ftpClientReturnCodes](#)

```
int fd = FTPGetFile(ftp, "testfile.txt", FALSE, 5 * TICKS_PER_SECOND);    // Send file in ASCII mode

if (fd > 0)
{
    // Code to read all data from the file descriptor goes here
    close(fd);

    int rv = FTPGetCommandResult(ftpSession, resultBuffer, 80, 5 * TICKS_PER*SECOND );

    if ( rv != 226)    // Return code 226 = Closing data connection. Requested file action successful.
        iprintf("Read error result = %d %s\r\n", rv, resultBuffer);
}
else
{
    iprintf("Failed to read file\r\n");
}
```

14.17.2.9 FTPGetFileNames()

```
int FTPGetFileNames (
    int ftp_Session,
    const char * full_dir_name,
    uint16_t timeout)
```

```
#include <ftp.h>
```

Initialize the process to receive just the file names in a directory from a FTP server.

This function opens a TCP connection to a FTP server so that a list of file names may be read. it will return a file descriptor that can then be used to read the file data using file I/O functions such as: [read\(\)](#) and [ReadWithTimeout\(\)](#).

IMPORTANT After reading the file data you must do 2 things:

- The TCP connection must be closed with the [close\(\)](#) function.
- [FTPGetCommandResult\(\)](#) must be called to check the result of the operation.

Parameters

<i>ftp_Session</i>	FTP session number
<i>full_dir_name</i>	Full directory name to read, including the path
<i>timeout</i>	Timeout in system Time Ticks

Returns

TCP file descriptor if greater than 0, otherwise ftpClientReturnCodes

14.17.2.10 FTPGetList()

```
int FTPGetList (
    int ftp_Session,
    const char * full_dir_name,
    uint16_t timeout)
#include <ftp.h>
```

Initialize the process to receive a directory listing from a FTP server.

This function opens a TCP connection to a FTP server so that a directory list may be read. it will return a file descriptor that can then be used to read the file data using file I/O functions such as: [read\(\)](#) and [ReadWithTimeout\(\)](#). IMPORTANT After reading the file data you must do 2 things:

- The TCP connection must be closed with the [close\(\)](#) function.
- [FTPGetCommandResult\(\)](#) must be called to check the result of the operation.

Parameters

<i>ftp_Session</i>	FTP session number
<i>full_dir_name</i>	Full directory name to read, including the path
<i>timeout</i>	Timeout in system Time Ticks

Returns

TCP file descriptor if greater than 0, otherwise ftpClientReturnCodes

14.17.2.11 FTPMakeDir()

```
int FTPMakeDir (
    int ftp_Session,
    const char * dir_to_make,
    uint16_t timeout)
#include <ftp.h>
```

Create a new directory.

Parameters

<i>ftp_Session</i>	FTP session number
<i>dir_to_make</i>	Directory to create
<i>timeout</i>	Timeout in system Time Ticks

Returns

[ftpClientReturnCodes](#)

14.17.2.12 FTPPassiveMode()

```
void FTPPassiveMode (
    int ftp_Session)
#include <ftp.h>
Set mode to passive.
```

Parameters

<i>ftp_Session</i>	FTP session number

14.17.2.13 FTPRawCommand()

```
int FTPRawCommand (
    int ftp_Session,
    const char * cmd,
    char * cmd_buf,
    int nbytes,
    uint16_t timeout)
#include <ftp.h>
Send a FTP command to the FTP server.
```

Parameters

<i>ftp_Session</i>	FTP session number.
<i>cmd</i>	The command to send. Do not include termination such as \r\n.
<i>cmd_buf</i>	The buffer to hold the FTP server response, in ASCII.
<i>nbytes</i>	The maximum number of bytes the cmd_buf can hold.
<i>timeout</i>	Timeout in system Time Ticks.

Returns

The FTP server response code if greater than 0, otherwise ftpClientReturnCodes

14.17.2.14 FTPRawStreamCommand()

```
int FTPRawStreamCommand (
    int ftp_Session,
    const char * cmd,
    int * pResult,
    char * cmd_buf,
    int nbytes,
    uint16_t timeout)
#include <ftp.h>
```

Send a command and receive a response over a stream connection.

The FTP server will close the connection after all the data has been read. This function is the basis for such functions as [FTPGetList\(\)](#) and [FTPGetFiles\(\)](#).

IMPORTANT After reading the file data you must call [FTPGetCommandResult\(\)](#) to check the result of the operation.

Parameters

<i>ftp_Session</i>	FTP session number
<i>cmd</i>	Command to send
<i>pResult</i>	The server response code integer value
<i>cmd_buf</i>	The server response code ASCII value (including \r\n)
<i>nbytes</i>	Maximum number of bytes cmd_buf can hold

<i>timeout</i>	Timeout in system Time Ticks

Returns

TCP file descriptor if greater than 0, otherwise ftpClientReturnCodes

14.17.2.15 FTPRenameFile()

```
int FTPRenameFile (
    int ftp_Session,
    const char * old_file_name,
    const char * new_file_name,
    uint16_t timeout)
#include <ftp.h>
```

Rename a file.

Parameters

<i>ftp_Session</i>	FTP session number
<i>old_file_name</i>	Name of file to rename
<i>new_file_name</i>	New name of file
<i>timeout</i>	Timeout in system Time Ticks

Returns

[ftpClientReturnCodes](#)

14.17.2.16 FTPSendFile()

```
int FTPSendFile (
    int ftp_Session,
    const char * full_file_name,
    BOOL bBinaryMode,
    uint16_t timeout)
#include <ftp.h>
```

Initialize the process to send a file to s FTP server.

This function opens a TCP connection to a FTP server so that a file may be sent. it will return a file descriptor that can then be used to send the file data using file I/O write functions such as: [write\(\)](#), [writeall\(\)](#), [writestring\(\)](#), etc.

IMPORTANT After sending the file data you must do 2 things:

- The TCP connection must be closed with the [close\(\)](#) function.
- [FTPGetCommandResult\(\)](#) must be called to check the result of the operation.

Parameters

<i>ftp_Session</i>	FTP session number
<i>full_file_name</i>	Name of the file being sent
<i>bBinaryMode</i>	True = send file as binary data, False = send file as ASCII data
<i>timeout</i>	Timeout in system Time Ticks

Returns

TCP file descriptor if greater than 0, otherwise [ftpClientReturnCodes](#)

```
int fd = FTPSendFile(ftp, "testfile.txt", FALSE, 5 * TICKS_PER_SECOND);    // Send file in ASCII mode

if (fd > 0)
{
    writestring(fd, "This is a test file\r\n");
    writestring(fd, "This is line 2 of the test file\r\n");
    writestring(fd, "Last Line\r\n");
    close(fd);

    int rv = FTPGetCommandResult(ftpSession, resultBuffer, 80, 5 * TICKS_PER*SECOND );

    if ( rv != 226)    // Return code 226 = Closing data connection. Requested file action successful.
        iprintf("Write error result = %d %s\r\n", rv, resultBuffer);
}
else
{
    iprintf("Failed to send file\r\n");
}
```

14.17.2.17 FTPSetDir()

```
int FTPSetDir (
    int ftp_Session,
    const char * new_dir,
    uint16_t timeout)
#include <ftp.h>
Set the current working directory.
```

Parameters

<i>ftp_Session</i>	FTP session number
<i>new_dir</i>	Directory path to set
<i>timeout</i>	Timeout in system Time Ticks

Returns

[ftpClientReturnCodes](#)

14.17.2.18 FTPUpDir()

```
int FTPUpDir (
    int ftp_Session,
    uint16_t timeout)
#include <ftp.h>
Move up one directory level.
```

Parameters

<i>ftp_Session</i>	FTP session number
<i>timeout</i>	Timeout in system Time Ticks

Returns

[ftpClientReturnCodes](#)

14.18 FTP Server**Topics**

- [FTP Server Callback](#)

Core Functions for running a FTP Server on a NetBurner device.

- [FTP Server with EFFS-FAT](#)
- [FTP Server with EFFS-STD](#)

14.18.1 Detailed Description

There are two APIs to implement an FTP Server:

- The FTP Server Callback API, which enables an application to create an FTP Server and process commands in any manor with the associated callback functions.
- The FTP Server File System (EFFS) API, which utilizes the FTP Callback API to implement a FTP Server using the EFFS-FAT or EFFS-STD file system. This is the most commonly used implementation.

14.18.2 FTP Server Callback

Core Functions for running a FTP Server on a NetBurner device.

Topics

- [FTP Server Core Return Codes](#)
- [FTP Server File Return Codes](#)

Initialization functions

- typedef void [FTPDCallBackReportFunc](#)(int handle, const char *name_to_report)
FTP Session callback typedef Typedef for all directory reporting callbacks This callback type definition is used by the directory reporting functions.
- int [FTPStartSSL](#) (uint16_t port, uint8_t server_priority, bool enableFTPS=false, bool encryptData=false)
Starts the FTP Server task (SSL) Listens for incoming connections. Only one instance of the FTPD is allowed.
- int [FTPStart](#) (uint16_t port, uint8_t server_priority)
Starts the FTP Server task (non-SSL/TLS) Listens for incoming connections. Only one instance of the FTPD is allowed. If FTPD SSL support exists, [FTPStartSSL\(\)](#) may be used instead.
- int [FTPStopReq](#) ()
Stops the FTP Server task Sends a stop request to the currently running FTPD.

FTP Session callback functions that must be implemented by the programmer

- void * [FTPSessionStart](#) (const char *user, const char *passwd, const [IPADDR4](#) hi_ip)
Start an FTP user session (User) This function is called following the creation of a new FTP session. The function needs to determine the validity of the user/password pair. The returned void pointer will be passed to all access functions, which will be asked to determine the validity of the operation based on the permissions associated with the return value.
- void [FTPSessionEnd](#) (void *pSession)
Indicate an FTP user session will be terminated. This callback function gives the user program the opportunity to clean up any storage associated with the void pointer returned from the [FTPSessionStart\(\)](#) call.

FTP Directory callback functions that must be implemented by the programmer

- int [FTPD_DirectoryExists](#) (const char *full_directory, void *pSession)
Called by the FTP Server to test for the existence of a directory Called by the FTP Server as the result of an attempt to change to a new directory. This function can also be used to validate the permissions of the session. This function must be implemented by the programmer.
- int [FTPD_CreateSubDirectory](#) (const char *current_directory, const char *new_dir, void *pSession)
Called by the FTP Server to create a directory Called by the FTP Server as the result of an attempt to create a new directory. This function can also be used to validate the permissions of the session. This function must be implemented by the programmer.
- int [FTPD_DeleteSubDirectory](#) (const char *current_directory, const char *sub_dir, void *pSession)

Called by the FTP Server to delete a directory Called by the FTP Server as the result of an attempt to delete a subdirectory. This function call can be used to validate the permissions of this session. This function must be implemented by the programmer.

- int [FTPD_ListSubDirectories](#) (const char *current_directory, void *pSession, [FTPDCallBackReportFunc](#) *pFunc, int handle)

Called by the FTP Server to list all subdirectories under the current directory Called by the FTP Server as the result of a client's attempt to list the contents of a directory. This function must be implemented by the programmer.

FTP file callback functions that must be implemented by the programmer

- int [FTPD_FileExists](#) (const char *full_directory, const char *file_name, void *pSession)
Report on the whether or not a file exists Check for the existence of a file, usually just before an attempt is made to download the file. This function must be implemented by the programmer.
- int [FTPD_GetFileSize](#) (const char *full_directory, const char *file_name)
Returns the size of a specific file.
- int [FTPD_SendFileToClient](#) (const char *full_directory, const char *file_name, void *pSession, int fd)
Send the contents of a file to a file descriptor Send a file to a FTP client. This function must be implemented by the programmer.
- int [FTPD_AbleToCreateFile](#) (const char *full_directory, const char *file_name, void *pSession)
Report on the ability to create/receive a file Determine if a file can be created. This function must be implemented by the programmer.
- int [FTPD_GetFileFromClient](#) (const char *full_directory, const char *file_name, void *pSession, int fd)
Create/get a file Receive a file from the FTP client. This function must be implemented by the programmer.
- int [FTPD_DeleteFile](#) (const char *current_directory, const char *file_name, void *pSession)
Delete a file Delete a file. This function must be implemented by the programmer.
- int [FTPD_Rename](#) (const char *current_directory, const char *cur_file_name, const char *new_file_name, void *pSession)
Rename a file Rename a file. This function must be implemented by the programmer.
- int [FTPD_ListFile](#) (const char *current_directory, void *pSession, [FTPDCallBackReportFunc](#) *pFunc, int handle)
Lists all files in the current directory List all files in the current directory. This function must be implemented by the programmer.

14.18.2.1 Detailed Description

Core Functions for running a FTP Server on a NetBurner device.

```
#include< ftpd.h>
```

The FTP Server Core Functions described in this section consist of function to start and stop the FTP server, and callback functions to handle FTP processing such as: session handling, directory and file creation, deletion and listings, etc. These core callback functions enable applications to create a FTP server in many different ways, with or without a file system.

Without a file system. Real time processing of files sent or received. For example, application data can be sent in response to a FTP client request as a text file by specifying the file type as .txt and sending the data. There are examples of this method in the "\nburn\examples\FTP" folder.

With a file system. There are 3 choices on your NetBurner platform: external flash cards (EFFS-FAT), on-chip flash memory (EFFS-STD), and ram drives. Examples of these methods are located in the "\nburn\examples\EFFS" folder. When using a file system these core functions do not need to be used directly. Since the choice has been made to use a file system rather than the callback functions, applications should use the functions described in the examples since the work of linking the callbacks to the file system has already been done for you.

Deprecated For NetBurner customers porting from the 2.x tools to 3.x: FTP_FAIL and FTP_OK were deprecated on Jan 16, 2008. These defines were removed and all server functions (denoted as FTPD) were modified to use FTPD_OK and FTPD_FAIL. This was necessary so that the FTP client and server header files could be used at the same time. If you have any errors in your application code, you can safely change all server functions from FTP_OK/FAIL to FTPD_OK/FAIL.

14.18.2.2 Typedef Documentation

14.18.2.2.1 FTPDCallBackReportFuncnt

```
typedef void FTPDCallBackReportFuncnt(int handle, const char *name_to_report)
#include <ftpd.h>
```

FTP Session callback typedef Typedef for all directory reporting callbacks This callback type definition is used by the directory reporting functions.

Parameters

<i>handle</i>	The handle passed into the listing function
<i>name_to_report</i>	The file name to report for use in a directory listing

See also

[FTPD_ListSubDirectories](#), [FTPD_ListFile](#)

14.18.2.3 Function Documentation

14.18.2.3.1 FTPD_AbleToCreateFile()

```
int FTPD_AbleToCreateFile (
    const char * full_directory,
    const char * file_name,
    void * pSession)
#include <ftpd.h>
```

Report on the ability to create/receive a file Determine if a file can be created. This function must be implemented by the programmer.

Parameters

<i>full_directory</i>	The current value of the session directory
<i>file_name</i>	The name of the file to create
<i>pSession</i>	The void * object returned from the FTPBSessionStart() function call

Return values

<i>FTPD_OK</i>	The requested file can be written/created
<i>FTPD_FAIL</i>	The requested file could not be created

See also

[FTPD_FileExists](#), [FTPD_SendFileToClient](#), [FTPD_GetFileFromClient](#), [FTPD_DeleteFile](#), [FTPD_ListFile](#)

14.18.2.3.2 FTPD_CreateSubDirectory()

```
int FTPD_CreateSubDirectory (
    const char * current_directory,
    const char * new_dir,
    void * pSession)
#include <ftpd.h>
```

Called by the FTP Server to create a directory Called by the FTP Server as the result of an attempt to create a new directory. This function can also be used to validate the permissions of the session. This function must be implemented by the programmer.

Parameters

<i>current_directory</i>	The current value of the session directory
<i>new_dir</i>	The directory to create under the <i>current_directory</i>
<i>pSession</i>	The void * object returned from the FTPBSessionStart() function call

Return values

<i>FTPD_OK</i>	The requested directory was created
<i>FTPD_FAIL</i>	The requested directory could not be created

See also

[FTPD_DirectoryExists](#), [FTPD_DeleteSubDirectory](#)

14.18.2.3.3 FTPD_DeleteFile()

```
int FTPD_DeleteFile (
    const char * current_directory,
    const char * file_name,
    void * pSession)
```

```
#include <ftpd.h>
```

Delete a file Delete a file. This function must be implemented by the programmer.

Parameters

<i>current_directory</i>	The current value of the session directory
<i>file_name</i>	The name of the file to delete
<i>pSession</i>	The void * object returned from the FTPBSessionStart() function call

Return values

<i>FTPD_OK</i>	The requested file was deleted
<i>FTPD_FAIL</i>	The requested file could not be deleted

See also

[FTPD_FileExists](#), [FTPD_SendFileToClient](#), [FTPD_AbleToCreateFile](#), [FTPD_GetFileFromClient](#), [FTPD_ListFile](#)

14.18.2.3.4 FTPD_DeleteSubDirectory()

```
int FTPD_DeleteSubDirectory (
    const char * current_directory,
    const char * sub_dir,
    void * pSession)
```

```
#include <ftpd.h>
```

Called by the FTP Server to delete a directory Called by the FTP Server as the result of an attempt to delete a subdirectory. This function call can be used to validate the permissions of this session. This function must be implemented by the programmer.

Parameters

<i>current_directory</i>	The current value of the session current directory

<i>sub_dir</i>	The directory to delete under the <i>current_directory</i>
<i>pSession</i>	The void * object returned from the <code>FTPBSessionStart()</code> function call at the beginning of the session

Return values

<i>FTPD_OK</i>	The requested directory was deleted
<i>FTPD_FAIL</i>	The requested directory could not be deleted

See also

[FTPD_DirectoryExists](#), [FTPD_CreateSubDirectory](#)

14.18.2.3.5 FTPD_DirectoryExists()

```
int FTPD_DirectoryExists (
    const char * full_directory,
    void * pSession)
```

```
#include <ftpd.h>
```

Called by the FTP Server to test for the existence of a directory Called by the FTP Server as the result of an attempt to change to a new directory. This function can also be used to validate the permissions of the session. This function must be implemented by the programmer.

Parameters

<i>full_directory</i>	Name of the new directory to test
<i>pSession</i>	The void * object returned from the FTPBSessionStart() function call

Return values

<i>FTPD_OK</i>	The requested directory exists
<i>FTPD_FAIL</i>	The requested directory does not exist, or access is not permitted for the user

See also

[FTPD_CreateSubDirectory](#), [FTPD_DeleteSubDirectory](#)

14.18.2.3.6 FTPD_FileExists()

```
int FTPD_FileExists (
    const char * full_directory,
    const char * file_name,
    void * pSession)
```

```
#include <ftpd.h>
```

Report on the whether or not a file exists Check for the existence of a file, usually just before an attempt is made to download the file. This function must be implemented by the programmer.

Parameters

<i>full_directory</i>	The current value of the session directory
<i>file_name</i>	The name of the file to check
<i>pSession</i>	The void * object returned from the FTPBSessionStart() function call

Return values

<i>FTPD_OK</i>	The requested file exists
<i>FTPD_FAIL</i>	The requested file does not exist

See also

[FTPD_FileExists](#), [FTPD_SendFileToClient](#), [FTPD_AbleToCreateFile](#), [FTPD_GetFileFromClient](#), [FTPD_DeleteFile](#), [FTPD_ListFile](#)

14.18.2.3.7 FTPD_GetFileFromClient()

```
int FTPD_GetFileFromClient (
    const char * full_directory,
    const char * file_name,
    void * pSession,
    int fd)
```

#include <ftpd.h>

Create/get a file Receive a file from the FTP client. This function must be implemented by the programmer.

Parameters

<i>full_directory</i>	The current value of the session directory
<i>file_name</i>	The name of the file to create
<i>pSession</i>	The void * object returned from the FTPBSessionStart() function call
<i>fd</i>	The file descriptor that will be used to receive the file

Return values

<i>FTPD_OK</i>	The requested file was written/created
<i>FTPD_FAIL</i>	The requested file was not created

See also

[FTPD_FileExists](#), [FTPD_SendFileToClient](#), [FTPD_AbleToCreateFile](#), [FTPD_DeleteFile](#), [FTPD_ListFile](#)

14.18.2.3.8 FTPD_GetFileSize()

```
int FTPD_GetFileSize (
    const char * full_directory,
    const char * file_name)
```

#include <ftpd.h>

Returns the size of a specific file.

Returns

file size

Return values

<i>FTPD_FILE_SIZE_UNKNOWN</i>	
<i>FTPD_FILE_SIZE_NOSUCH_FILE</i>	

14.18.2.3.9 FTPD_ListFile()

```
int FTPD_ListFile (
    const char * current_directory,
    void * pSession,
    FTPDCallBackReportFunc * pFunc,
    int handle)
```

#include <ftpd.h>

Lists all files in the current directory List all files in the current directory. This function must be implemented by the programmer.

Example: Everything inside the callback function stub must be supplied by the programmer.

The FTP server will automatically call this function and provide values for the function variables. It is the programmer's responsibility to execute `pFunc( )` with the provided handle and a pointer to the string representing the file name. `pFunc( )` must be executed once for each file name. In this example, the variables `number_of_directories` and `FileNames[]` must be declared and initialized elsewhere in the application program.

```
int FTPD_ListFile(const char * current_directory, void * pSession, FTPDCallBackReportFunct * pFunc, int handle)
{
    for (int n = 0; n < numberof_files; n++)
        pFunc(handle, FileNames[n]);
    return FTPD_OK;
}
```

Parameters

<i>current_directory</i>	The current value of the session directory
<i>pSession</i>	The void * object returned from the <code>FTPBSessionStart()</code> function call
<i>pFunc</i>	The pointer to the callback function to be called for each file name. This is a callback function provided and used by the NetBurner internal FTP code.
<i>handle</i>	The handle value to be passed back into the <code>pFunc</code> . This is a handle provided and used by the NetBurner internal FTP code.

Return values

<i>FTPD_OK</i>	The requested files were listed
<i>FTPD_FAIL</i>	The requested file was not listed

See also

[FTPD_FileExists](#), [FTPD_SendFileToClient](#), [FTPD_AbleToCreateFile](#), [FTPD_GetFileFromClient](#), [FTPD_DeleteFile](#)

14.18.2.3.10 FTPD_ListSubDirectories()

```
int FTPD_ListSubDirectories (
    const char * current_directory,
    void * pSession,
    FTPDCallBackReportFunct * pFunc,
    int handle)
```

```
#include <ftpd.h>
```

Called by the FTP Server to list all subdirectories under the current directory Called by the FTP Server as the result of a client's attempt to list the contents of a directory. This function must be implemented by the programmer.

Example:

Everything inside the callback function stub must be supplied by the programmer. The FTP server will automatically call this function and provide values for the function variables. It is the programmer's responsibility to execute `pFunc( )` with the provided handle and a pointer to the string representing the subdirectory name. `pFunc( )` must be executed once for each subdirectory name. In this example, the variables `number_of_directories` and `DirectoryName[]` must be declared and initialized elsewhere in the application program.

```
int FTPD_ListSubDirectories(const char * current_directory, void * pSession, FTPDCallBackReportFunct * pFunc, int handle);
{
    for (int n = 0; n < number_of_dir; n++)
        pFunc(handle, DirectoryName[n]);
    return FTPD_OK;
}
```

Parameters

<i>current_directory</i>	The current value of the session current directory

Parameters

<i>pSession</i>	The void * object returned from the FTPBSessionStart() function call
<i>pFunc</i>	The pointer to the callback function to be called for each subdirectory
<i>handle</i>	The handle value to be passed back into the pFunc

Return values

<i>FTPD_OK</i>	The requested listing was successfully delivered
<i>FTPD_FAIL</i>	The requested directory could not be listed

See also

[FTPD_DirectoryExists](#), [FTPD_DeleteSubDirectory](#), [FTPD_CreateSubDirectory](#)

14.18.2.3.11 FTPD_Rename()

```
int FTPD_Rename (
    const char * current_directory,
    const char * cur_file_name,
    const char * new_file_name,
    void * pSession)
```

```
#include <ftpd.h>
```

Rename a file Rename a file. This function must be implemented by the programmer.

Parameters

<i>current_directory</i>	The current value of the session directory
<i>cur_file_name</i>	The name of the file to rename
<i>new_file_name</i>	
<i>pSession</i>	The void * object returned from the FTPBSessionStart() function call

Return values

<i>FTPD_OK</i>	The requested file was deleted
<i>FTPD_FAIL</i>	The requested file could not be renamed

See also

[FTPD_FileExists](#), [FTPD_SendFileToClient](#), [FTPD_AbleToCreateFile](#), [FTPD_GetFileFromClient](#), [FTPD_ListFile](#)

14.18.2.3.12 FTPD_SendFileToClient()

```
int FTPD_SendFileToClient (
    const char * full_directory,
    const char * file_name,
    void * pSession,
    int fd)
```

```
#include <ftpd.h>
```

Send the contents of a file to a file descriptor Send a file to a FTP client. This function must be implemented by the programmer.

Parameters

<i>full_directory</i>	The current value of the session directory
<i>file_name</i>	The name of the file to send
<i>pSession</i>	The void * object returned from the FTPBSessionStart() function call
<i>fd</i>	The file descriptor to send to

Return values

<i>FTPD_OK</i>	The requested file was sent.
<i>FTPD_FAIL</i>	The requested file was not sent.

See also

[FTPD_FileExists](#), [FTPD_AbleToCreateFile](#), [FTPD_GetFileFromClient](#), [FTPD_DeleteFile](#), [FTPD_ListFile](#)

14.18.2.3.13 FTPDSessionEnd()

```
void FTPDSessionEnd (
    void * pSession)
#include <ftpd.h>
```

Indicate an FTP user session will be terminated. This callback function gives the user program the opportunity to clean up any storage associated with the void pointer returned from the FTPBSessionStart() call.

Parameters

<i>pSession</i>	The void * object returned from the FTPBSessionStart() function call

See also

[FTPBSessionStart](#)

14.18.2.3.14 FTPDSessionStart()

```
void * FTPDSessionStart (
    const char * user,
    const char * passwd,
    const IPADDR4 hi_ip)
#include <ftpd.h>
```

Start an FTP user session (User) This function is called following the creation of a new FTP session. The function needs to determine the validity of the user/password pair. The returned void pointer will be passed to all access functions, which will be asked to determine the validity of the operation based on the permissions associated with the return value.

Parameters

<i>user</i>	The name of the user attempting to establish an FTP session
<i>passwd</i>	The password of the user attempting to establish an FTP session
<i>hi_ip</i>	The IP address of the server trying to establish this connection

Return values

<i>NULL</i>	The user name/password set is invalid
<i>(obj)</i>	A non-null void pointer to an object that will be associated with this login session

See also

[FTPDSessionEnd](#)

14.18.2.3.15 FTPDStart()

```
int FTPDStart (
    uint16_t port,
    uint8_t server_priority)
#include <ftpd.h>
```

Starts the FTP Server task (non-SSL/TLS) Listens for incoming connections. Only one instance of the FTPD is allowed. If FTPD SSL support exists, [FTPDStartSSL\(\)](#) may be used instead.

Parameters

<i>port</i>	The TCP port to listen to for incoming FTP requests
<i>server_priority</i>	The RTOS task priority for the FTP Server

Return values

<i>FTPD_RUNNING</i>	- The FTPD is already running
<i>FTPD_LISTEN_ERR</i>	- The listen socket was not able to be opened
<i>FTPD_OK</i>	- The FTPD was succesfully started
<i>FTPD_FAIL</i>	- The FTPD task could not be created

See also

[FTPDStartSSL](#), [FTPDStopReq](#)

14.18.2.3.16 FTPDStartSSL()

```
int FTPDStartSSL (
    uint16_t port,
    uint8_t server_priority,
    bool enableFTPS = false,
    bool encryptData = false)
#include <ftpd.h>
```

Starts the FTP Server task (SSL) Listens for incoming connections. Only one instance of the FTPD is allowed.

Parameters

<i>port</i>	The TCP port to listen to for incoming FTP requests
<i>server_priority</i>	The RTOS task priority for the FTP Server
<i>enableFTPS</i>	
<i>encryptData</i>	

Return values

<i>FTPD_RUNNING</i>	- The FTPD is already running
<i>FTPD_LISTEN_ERR</i>	- The listen socket was not able to be opened
<i>FTPD_OK</i>	- The FTPD was succesfully started
<i>FTPD_FAIL</i>	- The FTPD task could not be created

See also

[FTPDStart](#), [FTPDStopReq](#)

14.18.2.3.17 FTPDStopReq()

```
int FTPDStopReq ()  
#include <ftpd.h>
```

Stops the FTP Server task Sends a stop request to the currently running FTPD.

Return values

<i>FTPD_RUNNING</i>	- The FTPD is still running
<i>FTPD_NOT_RUNNING</i>	- The FTPD is no longer running

See also

[FTPDStart](#), [FTPDStartSSL](#)

14.18.2.4 FTP Server Core Return Codes

Macros

- **#define FTPD_FAIL (0)**
Operation failed.
- **#define FTPD_OK (1)**
Operation succeeded.
- **#define FTPD_RUNNING (2)**
FTP Server running.
- **#define FTPD_NOT_RUNNING (3)**
FTP Server not running.
- **#define FTPD_LISTEN_ERR (4)**
FTP listen socket could not be opened, or is not open.

14.18.2.4.1 Detailed Description

14.18.2.5 FTP Server File Return Codes

Macros

- **#define FTPD_FILE_SIZE_NOSUCH_FILE (-1)**
No such file.
- **#define FTPD_FILE_SIZE_UNKNOWN (0)**
File size unknown.

14.18.2.5.1 Detailed Description

14.18.3 FTP Server with EFFS-FAT

```
#include< ftpd_effsfat.h>
```

FTP Server with EFFS-FAT file system implementation. For use with flash cards such as SD and microSD flash cards.

Note

Please refer to the FTP and HTTP examples in "\nburn\examples\EFFS\Fat".

14.18.4 FTP Server with EDFS-STD

```
#include< ftpd_edsstd.h>
```

FTP Server with EDFS-STD file system implementation. Implements a flash file system in the flash chips of your NetBurner device.

Note

Please refer to the HTTP example in "\nburn\examples\EDFS\Std".

14.19 File Descriptor Printf - fdprintf

Printf-style formatted output to file descriptors.

Functions

- int [fdprintf](#) (int fd, const char *format,...)
Print formatted output to a file descriptor.
- int [fdiprintf](#) (int fd, const char *format,...)
Print formatted output to a file descriptor (integer-only, no float)
- int [vfdprintf](#) (int fd, const char *format, va_list args)
Print formatted output to a file descriptor using va_list.
- int [vfdiprintf](#) (int fd, const char *format, va_list args)
Print formatted output to a file descriptor using va_list (integer-only)

14.19.1 Detailed Description

Printf-style formatted output to file descriptors.

```
#include< fdprintf.h>
```

The fdprintf family of functions extends the familiar printf() functionality to work with any file descriptor in the NetBurner system. This enables formatted output to:

- Serial ports (UART)
- TCP/IP network connections
- UDP sockets
- Custom file descriptors
- Any device that implements the file descriptor interface

14.19.2 Overview

Unlike printf() which outputs to stdout (typically the debug serial port), [fdprintf\(\)](#) allows you to direct formatted output to any open file descriptor. This is essential for network protocols, multi-port communications, and custom output devices.

14.19.3 Function Variants

Four variants are provided to suit different needs:

Function	Float Support	Argument Type	Use Case
fdprintf()	Yes	Variable args	General purpose with float
fdiprintf()	No	Variable args	Integer-only (smaller code)
vfdprintf()	Yes	va_list	Wrapper functions with float
vfdiprintf()	No	va_list	Wrapper functions, integer-only

14.19.4 Example Usage

14.19.4.1 Serial Port Output

```
#include <fdprintf.h>
#include <serial.h>

void sendSerialData() {
    int serialFd = SimpleOpenSerial(0, 115200);

    fdprintf(serialFd, "Temperature: %.2f C\r\n", 23.45);
    fdprintf(serialFd, "Status: %s\r\n", "OK");

    close(serialFd);
}
```

14.19.4.2 Network Socket Output

```
#include <fdprintf.h>
#include <tcp.h>

void sendToClient(int clientFd) {
    fdprintf(clientFd, "HTTP/1.1 200 OK\r\n");
    fdprintf(clientFd, "Content-Type: text/html\r\n\r\n");
    fdprintf(clientFd, "<html><body>\r\n");
    fdprintf(clientFd, "<h1>Device Status</h1>\r\n");
    fdprintf(clientFd, "<p>Uptime: %lu seconds</p>\r\n", Secs);
    fdprintf(clientFd, "</body></html>\r\n");
}
```

14.19.4.3 Creating Wrapper Functions

```
#include <fdprintf.h>
#include <stdarg.h>

// Custom logging function that adds timestamps
void logToSerial(const char *format, ...) {
    static int logFd = -1;

    if (logFd < 0) {
        logFd = SimpleOpenSerial(1, 115200);
    }

    // Print timestamp
    fdprintf(logFd, "[%lu] ", Secs);

    // Print user message using va_list
    va_list args;
    va_start(args, format);
    vfdprintf(logFd, format, args);
    va_end(args);
}

// Usage
logToSerial("Sensor reading: %d\n", sensorValue);
```

14.19.4.4 Integer-Only for Code Size

```
#include <fdprintf.h>

// Use fdiprintf when you don't need floating point
// This saves significant code space
void sendIntegerStatus(int fd) {
    fdiprintf(fd, "Counter: %d\r\n", counter);
    fdiprintf(fd, "Error Code: 0x%08X\r\n", errorCode);
    fdiprintf(fd, "Status: %s\r\n", statusString);
}
```

14.19.5 Format Specifiers

[fdprintf\(\)](#) supports standard C printf format specifiers:

- d, i - Signed decimal integer
- u - Unsigned decimal integer
- x, X - Hexadecimal
- o - Octal

- c - Character
- s - String
- p - Pointer
- f, e, g - Floating point (fdprintf/vdprintf only)
- %% - Literal %

Width, precision, and flags are supported:

- %5d - Minimum width
- %05d - Zero-padded width
- %.2f - Precision
- %-10s - Left-aligned
- %+d - Force sign

14.19.6 Performance Considerations

- **Code Size:** `fdiprintf()` is ~10-15KB smaller than `fdprintf()` due to no float support
- **Speed:** Integer formatting is faster than floating point
- **Buffering:** Consider buffering for high-frequency output
- **Network:** `fdprintf()` to network sockets may block if buffer is full

14.19.7 Best Practices

1. **Error Checking:** Always check file descriptor validity before use
2. **Return Value:** Check return value for write errors
3. **Buffer Size:** Be aware of output buffer limits on network sockets
4. **Float Support:** Use `fdiprintf()` when floating point is not needed
5. **Line Endings:** Use `\r\n` for network protocols and serial terminals

14.19.8 Important Notes

Note

The `fdprintf()` family does not append a newline automatically. You must include `\n` or `\r\n` in your format string if desired.

Output to network sockets may block if the socket send buffer is full. Consider using non-blocking I/O or checking buffer space for critical applications.

File descriptors must be properly opened before use and closed when done.

Warning

Floating point formatting (fdprintf/vdprintf) increases code size significantly. Use `fdiprintf` or `vdiprintf` when float support is not required.

Large format strings or many arguments can overflow stack space. Keep format operations reasonably sized.

14.19.9 Example Code

Complete examples are available in the NetBurner SDK:

- `\nburn\examples\Filedescriptor` - File descriptor usage examples
- Examples include custom file descriptor creation and usage

See also

`printf()` Standard printf to stdout

`iprintf()` Integer-only printf to stdout

`snprintf()` Format to string buffer

[writeall\(\)](#) Write raw data to file descriptor

14.19.10 Function Documentation

14.19.10.1 fdprintf()

```
int fdprintf (
    int fd,
    const char * format,
    ...)
```

```
#include <fdprintf.h>
```

Print formatted output to a file descriptor (integer-only, no float)

Similar to [fdprintf\(\)](#) but without floating point support. This significantly reduces code size (~10-15KB smaller) and is faster for integer-only formatting. Use this when you only need to format integers, strings, and other non-float types.

Parameters

<i>fd</i>	File descriptor to write to. Must be a valid, open file descriptor such as: • Serial port (from SimpleOpenSerial()) • TCP socket (from accept() or connect()) • UDP socket (from <code>socket()</code>) • Custom file descriptor
<i>format</i>	Format string using standard printf syntax. Supports all specifiers EXCEPT floating point (f, e, g are not supported): • Integer formats: d, i, u, x, X, o • String format: s • Character format: c • Pointer format: p • Width and precision modifiers
...	Variable arguments corresponding to format specifiers in the format string

Returns

Number of bytes written to the file descriptor, or negative value on error

Return values

>0	Number of bytes successfully written
<0	Error occurred during write operation

Precondition

- fd must be a valid, open file descriptor
- format must not be NULL
- format must not contain floating point specifiers (f, e, g)

Postcondition

- Data is written to the file descriptor's output buffer

Note

This function does NOT support floating point formatting. Attempting to use f, e, or g will produce undefined results.

No newline is automatically appended. Include `\n` or `\r\n` in the format string if line ending is desired.

Warning

Using floating point specifiers (f, e, g) will cause undefined behavior. Use [fdprintf\(\)](#) if float formatting is needed.

Example: Basic Integer Output

```
int fd = SimpleOpenSerial(0, 115200);
if (fd >= 0) {
    fdiprintf(fd, "Counter: %d\r\n", counter);
    fdiprintf(fd, "Status: 0x%08X\r\n", status);
    close(fd);
}
```

Example: Network Status Messages

```
void sendStatus(int clientFd) {
    fdiprintf(clientFd, "Device ID: %d\r\n", deviceId);
    fdiprintf(clientFd, "Uptime: %lu seconds\r\n", Secs);
    fdiprintf(clientFd, "Memory Free: %d bytes\r\n", OSFreeMemory());
}
```

Example: Code Size Optimization

```
// Instead of this (adds 15KB):
fprintf(fd, "Value: %d\n", value);

// Use this (saves 15KB if no other float formatting):
fdiprintf(fd, "Value: %d\n", value);
```

See also

- [fdprintf\(\)](#) Full version with floating point support
- [vfdiprintf\(\)](#) va_list version for wrapper functions
- [iprintf\(\)](#) Integer-only printf to stdout

14.19.10.2 fdprintf()

```
int fdprintf (
    int fd,
    const char * format,
    ...)
```

```
#include <fdprintf.h>
```

Print formatted output to a file descriptor.

Outputs formatted text to the specified file descriptor using the same format syntax as standard printf(). Supports all standard format specifiers including floating point.

Parameters

<i>fd</i>	File descriptor to write to. Must be a valid, open file descriptor such as: • Serial port (from SimpleOpenSerial()) • TCP socket (from accept() or connect()) • UDP socket (from socket()) • Custom file descriptor
<i>format</i>	Format string using standard printf syntax. Supports: • All standard format specifiers (d, s, f, x, etc.) • Width and precision modifiers (%-10s, %.2f, %05d) • All standard flags (+, -, 0, space, #)
...	Variable arguments corresponding to format specifiers in the format string

Returns

Number of bytes written to the file descriptor, or negative value on error

Return values

>0	Number of bytes successfully written
<0	Error occurred during write operation

Precondition

fd must be a valid, open file descriptor

format must not be NULL

Postcondition

Data is written to the file descriptor's output buffer

On network sockets, data may be queued for transmission

Note

This function includes floating point support which adds ~10-15KB to code size. Use [fdprintf\(\)](#) if floating point is not needed.

No newline is automatically appended. Include `\n` or `\r\n` in the format string if line ending is desired.

Warning

May block if writing to a network socket with a full send buffer. Consider non-blocking I/O for time-critical applications.

Very large format operations may overflow stack space. Keep output operations reasonably sized.

Example: Basic Usage

```
int fd = SimpleOpenSerial(0, 115200);
if (fd >= 0) {
    fdprintf(fd, "Temperature: %.2f C\r\n", 23.45);
    close(fd);
}
```


Example: Network Protocol

```
void sendHTTPResponse(int clientFd, int statusCode) {
    fprintf(clientFd, "HTTP/1.1 %d OK\r\n", statusCode);
    fprintf(clientFd, "Content-Length: %d\r\n", contentLen);
    fprintf(clientFd, "\r\n");
}
```

Example: Error Checking

```
int bytesWritten = fprintf(fd, "Status: %d\n", status);
if (bytesWritten < 0) {
    fprintf("Error writing to file descriptor\n");
}
```

See also

[fprintf\(\)](#) Integer-only version (smaller code size)

[vfprintf\(\)](#) va_list version for wrapper functions

[printf\(\)](#) Standard output version

14.19.10.3 vfprintf()

```
int vfprintf (
    int fd,
    const char * format,
    va_list args)
```

```
#include <fprintf.h>
```

Print formatted output to a file descriptor using va_list (integer-only)

Variant of [fprintf\(\)](#) that accepts a va_list argument instead of variable arguments. This is used for creating wrapper functions that need to forward variable arguments. Does NOT support floating point, resulting in smaller code size.

Parameters

<i>fd</i>	File descriptor to write to. Must be a valid, open file descriptor
<i>format</i>	Format string using standard printf syntax. Supports all specifiers EXCEPT floating point (f, e, g are not supported): - Integer formats: d, i, u, x, X, o - String format: s - Character format: c - Pointer format: p
<i>args</i>	Variable argument list (va_list) containing the arguments specified by the format string. Must be initialized with va_start() and cleaned up with va_end()

Returns

Number of bytes written to the file descriptor, or negative value on error

Return values

>0	Number of bytes successfully written
<0	Error occurred during write operation

Precondition

fd must be a valid, open file descriptor
 format must not be NULL
 format must not contain floating point specifiers (f, e, g)
 args must be properly initialized with va_start()

Postcondition

Data is written to the file descriptor's output buffer
 args parameter is consumed and should not be reused

Note

This function does NOT support floating point formatting. Use [vfdprintf\(\)](#) if float support is needed.
 The caller is responsible for calling va_start() before calling this function and va_end() after the call.
 This function is ~10-15KB smaller than [vfdprintf\(\)](#) due to lack of floating point support.

Warning

Using floating point specifiers (f, e, g) will cause undefined behavior. Use [vfdprintf\(\)](#) if float formatting is needed.

Example: Integer-Only Logging Wrapper

```
#include <fdprintf.h>
#include <stdarg.h>

// Memory-efficient logging (no float support)
void quickLog(int fd, const char *format, ...) {
    va_list args;
    va_start(args, format);
    vfdiprintf(fd, format, args);
    va_end(args);
}

// Usage - only integers and strings
quickLog(logFd, "Counter: %d, Status: %s\n", count, status);
```

Example: Diagnostic Output Wrapper

```
void diagPrint(const char *format, ...) {
    static int diagFd = -1;

    if (diagFd < 0) {
        diagFd = SimpleOpenSerial(2, 115200);
    }

    if (diagFd >= 0) {
        va_list args;
        va_start(args, format);
        vfdiprintf(diagFd, format, args);
        va_end(args);
    }
}

// Usage
diagPrint("Error Code: 0x%08X\n", errorCode);
diagPrint("Free Memory: %d bytes\n", OSFreeMemory());
```

Example: Protocol Wrapper

```
// Modbus protocol wrapper (integer-only)
void modbusResponse(int fd, uint8_t funcCode, ...) {
    fdiprintf(fd, ":%02X", funcCode);

    va_list args;
    va_start(args, funcCode);

    // Additional data fields
    int dataCount = va_arg(args, int);
```

```

    for (int i = 0; i < dataCount; i++) {
        int value = va_arg(args, int);
        fdprintf(fd, "%04X", value);
    }
    va_end(args);

    fdprintf(fd, "\r\n");
}

```

See also

[fdprintf\(\)](#) Direct variable argument version

[vfdprintf\(\)](#) Full version with floating point support

[va_start\(\)](#) Initialize va_list

[va_end\(\)](#) Clean up va_list

14.19.10.4 vfdprintf()

```

int vfdprintf (
    int fd,
    const char * format,
    va_list args)

```

#include <fdprintf.h>

Print formatted output to a file descriptor using va_list.

Variant of [fdprintf\(\)](#) that accepts a va_list argument instead of variable arguments. This is used for creating wrapper functions that need to forward variable arguments. Includes full floating point support.

Parameters

<i>fd</i>	File descriptor to write to. Must be a valid, open file descriptor
<i>format</i>	Format string using standard printf syntax. Supports all standard format specifiers including floating point
<i>args</i>	Variable argument list (va_list) containing the arguments specified by the format string. Must be initialized with va_start() and cleaned up with va_end()

Returns

Number of bytes written to the file descriptor, or negative value on error

Return values

>0	Number of bytes successfully written
<0	Error occurred during write operation

Precondition

fd must be a valid, open file descriptor

format must not be NULL

args must be properly initialized with va_start()

Postcondition

Data is written to the file descriptor's output buffer

args parameter is consumed and should not be reused

Note

This function is primarily used for creating wrapper functions. Direct application code should typically use [fdprintf\(\)](#) instead.

The caller is responsible for calling `va_start()` before calling this function and `va_end()` after the call.

This function includes floating point support. Use [vfdprintf\(\)](#) for integer-only formatting to save code space.

Example: Creating a Logging Wrapper

```
#include <fdprintf.h>
#include <stdarg.h>

// Custom logging function with automatic timestamp
void logMessage(int fd, const char *format, ...) {
    // Add timestamp prefix
    fdprintf(fd, "[%lu] ", Secs);

    // Forward variable arguments to vfdprintf
    va_list args;
    va_start(args, format);
    vfdprintf(fd, format, args);
    va_end(args);
}

// Usage
int logFd = SimpleOpenSerial(1, 115200);
logMessage(logFd, "Temperature: %.2f C\n", temp);
logMessage(logFd, "Status: %s\n", statusStr);
```

Example: Multi-Destination Output

```
void sendToMultiple(const char *format, ...) {
    va_list args;

    // Send to serial port
    va_start(args, format);
    vfdprintf(serialFd, format, args);
    va_end(args);

    // Send to network client
    va_start(args, format);
    vfdprintf(networkFd, format, args);
    va_end(args);
}
```

Example: Buffered Output Wrapper

```
static char outputBuffer[512];

void bufferedOutput(int fd, const char *format, ...) {
    // Format to buffer first
    va_list args;
    va_start(args, format);
    int len = vsnprintf(outputBuffer, sizeof(outputBuffer), format, args);
    va_end(args);

    // Write buffer to fd
    if (len > 0) {
        writeall(fd, outputBuffer, len);
    }
}
```

See also

[fdprintf\(\)](#) Direct variable argument version

[vfdprintf\(\)](#) Integer-only `va_list` version

`va_start()` Initialize `va_list`

`va_end()` Clean up `va_list`

14.20 GDB Stub (Remote Debugging)

GDB Stub functions.

Stub Initialization functions

- void [InitGDBStub](#) (int port, int baudrate)

Start the GDB stub and then immediately stop at a break point This function will start the GDB stub, and will stop immediately at a breakpoint. This function is best used during development to stop the system from executing questionable code at startup. It is also useful to control the system execution to debug startup and initialization code.

- void [InitGDBStubNoBreak](#) (int port, int baudrate)

Start the GDB stub and continue without stopping This function will start the GDB stub, and continue with normal program execution. This function is best used to debug a program that is operating correctly and has intermitant bugs or anomallies. The program will run as normal until you connect the debugger. Thus you can operate the program until one needs to debug it.

14.20.1 Detailed Description

GDB Stub functions.

```
#include <gdbstub.h>
```

This module provides code for operating a serial GDB stub. This is used to debug programs running on a NetBurner system. More information can be found in ../nburn/docs/gdb/GDB-Gettingstarted

14.20.2 Function Documentation

14.20.2.1 InitGDBStub()

```
void InitGDBStub (
    int port,
    int baudrate)
#include <gdbstub.h>
```

Start the GDB stub and then immediately stop at a break point This function will start the GDB stub, and will stop immediately at a breakpoint. This function is best used during development to stop the system from executing questionable code at startup. It is also useful to control the system execution to debug startup and initialization code.

Parameters

<i>port</i>	The serial port to use this can be 0 or 1 corresponding to uart 0 or uart 1
<i>baudrate</i>	The baud rate to use on the port.

See also

[InitGDBStubNoBreak](#)

14.20.2.2 InitGDBStubNoBreak()

```
void InitGDBStubNoBreak (
    int port,
    int baudrate)
#include <gdbstub.h>
```

Start the GDB stub and continue without stopping This function will start the GDB stub, and continue with normal program execution. This function is best used to debug a program that is operating correctly and has intermitant bugs or anomallies. The program will run as normal until you connect the debugger. Thus you can operate the program until one needs to debug it.

Parameters

<i>port</i>	The serial port to use this can be 0 or 1 corresponding to uart 0 or uart 1

<i>baudrate</i>	The baud rate to use on the port.

See also

[InitGDBStub](#)

14.21 GPIO

Topics

- [PinIO](#)
The Netburner GPIO [PinIO](#) class driver (common across most platforms)
- [SOMRT1061 Pin Function Constants](#)

14.21.1 Detailed Description

GPIO API for NetBurner Platforms

`#include< cpu_pins.h >` (in the appropriate arch include subfolder)

14.21.2 PinIO

The Netburner GPIO [PinIO](#) class driver (common across most platforms)

Classes

- class [PinIO](#)
GPIO Pin Class.
- class [_PinVector](#)
GPIO Pin Vector Base Class.
- class [PinVector< n >](#)
GPIO Pin Vector Class [PinVector](#) is a template instantiation of the [_PinVector](#) class, allowing for minimal storage requirements for potentially large vectors, without heavy code duplication due to template copies.

Variables

- volatile Pio & [PinIO::pio](#)
- uint32_t [PinIO::mask](#)

14.21.2.1 Detailed Description

The Netburner GPIO [PinIO](#) class driver (common across most platforms)

`#include< cpu_pins.h>`

14.21.2.2 Variable Documentation

14.21.2.2.1 mask

`uint32_t PinIO::mask`

Bit mask setting which pins in the given Pio hardware module the [PinIO](#) object will manage.

14.21.2.2.2 pio

`volatile Pio& PinIO::pio`

Reference to the Pio hardware module that this [PinIO](#) will manage. Combines with the mask to manage a single pin or multiple within the module.

14.21.3 SOMRT1061 Pin Function Constants

14.21.3.1 Detailed Description

Helper constants for setting functions for each pin in a readable way

14.22 HAL - Hardware Abstraction Layer

Functions

- void **HardwareSetup** ()
Initializes core system hardware components during the boot sequence.
- void **PostConfigHardwareInit** ()
Initializes hardware components that require configuration data to be loaded.
- void **ForceReboot** (bool fromIRQ=false)
Initiates an immediate hardware-level system reset of the NetBurner device.
- int **HalStorage_Save** (uint8_t area, void *pData, int len, int offset=0)
Save a blob to a specific persistent storage area as defined by the platform. This routine will perform all storage maintenance routines necessary to ensure a valid write.
- int **HalStorage_SavePartial** (uint8_t area, void *pData, int len, int offset)
Save a blob to a specific persistent storage area as defined by the platform. This routine requires that any maintenance required for a valid write be explicitly performed prior to being called.
- int **HalStorage_Prepare** (uint8_t area, int len, int offset=0)
Prepare a storage area for writing new data. For platforms with direct flash mapping this function is an alias for HalStorage_Erase.
- int **HalStorage_Finalize** (uint8_t area)
Finalize a storage area after writing new data. This will perform any final completion or cleanup routines required to persist the previously saved data from HalStorage_SavePartial calls.
- int **HalStorage_Erase** (uint8_t area, int len=-1, int offset=0)
Erase all or part of a storage area. Note: due to physical storage granularities, the total area erased may extend beyond the requested area.
- int **HalStorage_GetAllocated** (uint8_t area)
Obtain the total size allocated to the given persistent storage area.
- int **HalStorage_GetMaxAllocation** (uint8_t area)
Obtain the maximum size that may be allocated to the given persistent storage area. For direct flash mapped platforms, this is aliased to HalStorage_GetAllocated.
- int **HalStorage_RemainingSpace** (uint8_t area)
Obtain the number of remaining bytes available to be written in the given persistent storage area.
- int **HalStorage_WriteOffset** (uint8_t area)
Get the offset of the next byte that can be written in the given persistent storage area.
- int **HalStorage_AddressOffset** (uint8_t area, void *pWhere)
Get the offset within a persistent storage area of an address. A platform may map any address to any offset of its choosing for a given storage area, and may modify its behavior on an area by area basis.
- void **FlashErase** (void *pWhere, int len)
Erases the flash memory.
- void **FlashProgram** (void *pWhere, void *pWhat, int len)
Program flash memory.
- void **FlashProgramApplImage** (void *pWhere, void *pWhat, int len)
Write an application image to flash memory.
- void **DisableCache** ()
Disable the instruction and data cache.
- void **EnableCache** ()
Enable the instruction and data cache.
- uint32_t **spaceleft** ()

- Report how much free unallocated space is left in dynamic memory.*

 - uint16_t [HalGetTickFraction](#) (void)

Returns the fraction of the current system time tick.
- void [StdioCheckIntc](#) (void)

Check STDIO interrupt sources.
- void [SysLogCheckIntc](#) (void)

This is just like the [StdioCheckIntc\(\)](#) function, except that the results are displayed via UDP.
- bool [HalDeviceCertValid](#) ()

Determine if the stored certificate is valid.
- uint8_t * [HalGetDeviceCert](#) ()

Get a pointer to the stored certificate.
- uint16_t [HalGetDeviceCertLen](#) ()

Get the length of the stored certificate.
- uint8_t * [HalGetDeviceKey](#) ()

Get a pointer to the stored certificate.
- uint16_t [HalGetDeviceKeyLen](#) ()

Get the length of the stored key.
- bool [HalSaveNewDeviceCert](#) (const uint8_t *cert, uint16_t certlen, uint8_t format=SSL_FILETYPE_PEM)

Save a device certificate in persistent storage.
- bool [HalSaveNewDeviceKey](#) (const uint8_t *key, uint16_t keylen, uint8_t format=SSL_FILETYPE_PEM)

Save a device key in persistent storage.
- void [HalEraseDeviceCertAndKey](#) ()

Clear the device certificate and key from persistent storage.

Variables

- void(* [watchdog_service_function](#))(void)

Watchdog callback service function.
- uint32_t [HalTickMaxCount](#)

Rollover value for the system hardware tick timer.

14.22.1 Detailed Description

```
#include< hal.h>
```

The HAL provides a common function name interface to the various NetBurner hardware platforms. Other than the [ForceReboot\(\)](#) and [spaceleft\(\)](#) functions, these are advanced functions that should only be used by developers experienced with NetBurner hardware, firmware and memory maps.

14.22.2 Function Documentation

14.22.2.1 FlashErase()

```
void FlashErase (
    void * pWhere,
    int len)
#include <hal.h>
Erases the flash memory.
```

Parameters

<i>pWhere</i>	The starting location in flash memory to begin the erasure
<i>len</i>	Specified the number of bytes to erase

14.22.2.2 FlashProgram()

```
void FlashProgram (
    void * pWhere,
    void * pWhat,
    int len)
#include <hal.h>
Program flash memory.
```

Parameters

<i>pWhere</i>	Pointer to the starting location in flash to begin programming
<i>pWhat</i>	Pointer to the content that will be programmed into flash
<i>len</i>	Number of bytes to program

14.22.2.3 FlashProgramAppImage()

```
void FlashProgramAppImage (
    void * pWhere,
    void * pWhat,
    int len)
#include <hal.h>
```

Write an application image to flash memory.

On some platforms writing the application image may require calling ProgramImage even though it may only call FlashErase and FlashProgram. As a result, ProgramImage should always be used when updating the application image.

Parameters

<i>pWhere</i>	Pointer to the starting location in flash to begin the programming
<i>pWhat</i>	Pointer to the content that will be programmed into flash
<i>len</i>	Number of bytes to write

14.22.2.4 ForceReboot()

```
void ForceReboot (
    bool fromIRQ = false)
#include <hal.h>
```

Initiates an immediate hardware-level system reset of the NetBurner device.

This function triggers a complete system restart by directly commanding the hardware reset mechanisms. The reset is immediate and unconditional - all running processes are terminated, memory contents are cleared, and the system restarts from the boot sequence. This is equivalent to a power cycle but is performed through software control. The function never returns to the calling code as the system undergoes a complete restart.

Parameters

<i>fromIRQ</i>	Optional parameter indicating the reset context true: reset is being triggered from within an interrupt service routine or exception handler, which may require special handling procedures false: reset is being triggered from normal application code (default)

Note

This function never returns - execution does not continue after this call. All unsaved data, network connections, and application state will be lost. File systems and persistent storage should be properly synchronized before calling. The fromIRQ parameter ensures proper reset handling when called from interrupt contexts. After reset, the system will restart from the beginning of the boot sequence. Use this function only when a complete system restart is absolutely necessary.

Warning

This function immediately terminates all system operations without cleanup. Ensure critical data is saved and network connections are properly closed before use.

See also

attribute((noreturn)) indicates this function never returns to the caller

14.22.2.5 HalDeviceCertValid()

```
bool HalDeviceCertValid ()
#include <hal.h>
```

Determine if the stored certificate is valid.

This is the only function recommended to be used to determine the validity of the stored certificate. Is is not recommended to use [HalGetDeviceCert\(\)](#) nor [HalGetDeviceCertLen\(\)](#) to determine validity. For safest use of the cert/key getter functions, use [HalDeviceCertValid\(\)](#) prior.

Return values

<i>true</i>	if the certificate is valid false otherwise

14.22.2.6 HalEraseDeviceCertAndKey()

```
void HalEraseDeviceCertAndKey ()
#include <hal.h>
```

Clear the device certificate and key from persistent storage.

[HalEraseDeviceCertAndKey\(\)](#) is recommended instead of [HalStorage_Erase\(\)](#) to clear the stored certificate and key.

14.22.2.7 HalGetDeviceCert()

```
uint8_t * HalGetDeviceCert ()
#include <hal.h>
```

Get a pointer to the stored certificate.

It is recommended to use [HalDeviceCertValid\(\)](#) prior to calling this function.

Return values

<i>pointer</i>	to the certificate storage location in memory.

14.22.2.8 HalGetDeviceCertLen()

```
uint16_t HalGetDeviceCertLen ()
#include <hal.h>
```

Get the length of the stored certificate.

It is recommended to use [HalDeviceCertValid\(\)](#) prior to calling this function.

Return values

<i>length</i>	of the stored certificate. 0 or 65535 if no certificates is stored.

14.22.2.9 HalGetDeviceKey()

```
uint8_t * HalGetDeviceKey ()
#include <hal.h>
```

Get a pointer to the stored certificate.

It is recommended to use [HalDeviceCertValid\(\)](#) prior to calling this function.

Return values

<i>pointer</i>	to the certificate storage location in memory.

14.22.2.10 HalGetDeviceKeyLen()

```
uint16_t HalGetDeviceKeyLen ()
#include <hal.h>
```

Get the length of the stored key.

It is recommended to use [HalDeviceCertValid\(\)](#) prior to calling this function.

Return values

<i>length</i>	of the stored key. 0 or 65535 if no key is stored.

14.22.2.11 HalGetTickFraction()

```
uint16_t HalGetTickFraction (
    void ) [extern]
#include <hal.h>
```

Returns the fraction of the current system time tick.

return - The current counter value used to generate tick count

14.22.2.12 HalSaveNewDeviceCert()

```
bool HalSaveNewDeviceCert (
    const uint8_t * cert,
    uint16_t certlen,
    uint8_t format = SSL_FILETYPE_PEM)
#include <hal.h>
```

Save a device certificate in persistent storage.

Parameters

<i>cert</i>	Pointer to the buffer containing the certificate
<i>certlen</i>	length of the certificate
<i>format</i>	filetype (SSL_FILETYPE_PEM, or SSL_FILETYPE_ASN1)

Return values

<i>true</i>	if the certificate was successfully stored
<i>false</i>	otherwise

14.22.2.13 HalSaveNewDeviceKey()

```
bool HalSaveNewDeviceKey (
    const uint8_t * key,
    uint16_t keylen,
    uint8_t format = SSL_FILETYPE_PEM)
```

```
#include <hal.h>
```

Save a device key in persistent storage.

Parameters

<i>key</i>	Pointer to the buffer containing the key
<i>keylen</i>	length of the key
<i>format</i>	filetype (SSL_FILETYPE_PEM, or SSL_FILETYPE_ASN1)

Return values

<i>true</i>	if the key was successfully stored
<i>false</i>	otherwise

14.22.2.14 HalStorage_AddressOffset()

```
int HalStorage_AddressOffset (
    uint8_t area,
    void * pWhere)
```

```
#include <hal.h>
```

Get the offset within a persistent storage area of an address. A platform may map any address to any offset of its choosing for a given storage area, and may modify its behavior on an area by area basis.

Parameters

<i>area</i>	The storage area being queried
<i>pWhere</i>	An address value to be translated

Returns

Offset of the requested address within the storage area. Returns negative on error.

14.22.2.15 HalStorage_Erase()

```
int HalStorage_Erase (
    uint8_t area,
    int len = -1,
    int offset = 0)
```

```
#include <hal.h>
```

Erase all or part of a storage area. Note: due to physical storage granularities, the total area erased may extend beyond the requested area.

Parameters

<i>area</i>	The storage area to be erased
<i>len</i>	Number of bytes to be erased (a negative length will erase the entire area)
<i>offset</i>	The location within the area that the erasure should begin

Returns

Number of bytes erased. Returns negative on error.

See also

[HalEraseDeviceCertAndKey\(\)](#)

14.22.2.16 HalStorage_Finalize()

```
int HalStorage_Finalize (  
    uint8_t area)
```

```
#include <hal.h>
```

Finalize a storage area after writing new data. This will perform any final completion or cleanup routines required to persist the previously saved data from HalStorage_SavePartial calls.

Parameters

<i>area</i>	The storage area to finalize

Returns

Returns negative on error.

14.22.2.17 HalStorage_GetAllocated()

```
int HalStorage_GetAllocated (  
    uint8_t area)
```

```
#include <hal.h>
```

Obtain the total size allocated to the given persistent storage area.

Parameters

<i>area</i>	The storage area being queried

Returns

Number of bytes allocated. Returns negative on error.

14.22.2.18 HalStorage_GetMaxAllocation()

```
int HalStorage_GetMaxAllocation (  
    uint8_t area)
```

```
#include <hal.h>
```

Obtain the maximum size that may be allocated to the given persistent storage area. For direct flash mapped platforms, this is aliased to HalStorage_GetAllocated.

Parameters

<i>area</i>	The storage area being queried

Returns

Number of bytes allocated. Returns negative on error.

14.22.2.19 HalStorage_Prepare()

```
int HalStorage_Prepare (
    uint8_t area,
    int len,
    int offset = 0)
```

```
#include <hal.h>
```

Prepare a storage area for writing new data. For platforms with direct flash mapping this function is an alias for HalStorage_Erase.

Parameters

<i>area</i>	The storage area to be readied
<i>len</i>	Number of bytes to be written by later HalStorage_SavePartial calls
<i>offset</i>	The location within the area that the readying should begin

Returns

Number of bytes readied. Returns negative on error.

14.22.2.20 HalStorage_RemainingSpace()

```
int HalStorage_RemainingSpace (
    uint8_t area)
```

```
#include <hal.h>
```

Obtain the number of remaining bytes available to be written in the given persistent storage area.

Parameters

<i>area</i>	The storage area being queried

Returns

Number of bytes available. Returns negative on error.

14.22.2.21 HalStorage_Save()

```
int HalStorage_Save (
    uint8_t area,
    void * pData,
    int len,
    int offset = 0)
```

```
#include <hal.h>
```

Save a blob to a specific persistent storage area as defined by the platform. This routine will perform all storage maintenance routines necessary to ensure a valid write.

Parameters

<i>area</i>	The storage area to be used. A platform is only <i>required</i> to support the areas as defined by HalStorage_t, however it may define additional areas as needed
<i>pData</i>	Pointer to the data to be written
<i>len</i>	Number of bytes to be written to the storage area
<i>offset</i>	Where in the storage area the first byte should be written. Default is 0.

Returns

Number of bytes written. Returns negative on error.

14.22.2.22 HalStorage_SavePartial()

```
int HalStorage_SavePartial (
    uint8_t area,
    void * pData,
    int len,
    int offset)
```

```
#include <hal.h>
```

Save a blob to a specific persistent storage area as defined by the platform. This routine requires that any maintenance required for a valid write be explicitly performed prior to being called.

Parameters

<i>area</i>	The storage area to be used. A platform is only <i>required</i> to support the areas as defined by HalStorage_t, however it may define additional areas as needed.
<i>pData</i>	Pointer to the data to be written
<i>len</i>	Number of bytes to be written to the storage area
<i>offset</i>	Where in the storage area the first byte should be written.

Returns

Number of bytes written. Returns negative on error.

14.22.2.23 HalStorage_WriteOffset()

```
int HalStorage_WriteOffset (
    uint8_t area)
```

```
#include <hal.h>
```

Get the offset of the next byte that can be written in the given persistent storage area.

Parameters

<i>area</i>	The storage area being queried

Returns

Offset of the next byte that can be written. Returns negative on error.

14.22.2.24 HardwareSetup()

```
void HardwareSetup ()
#include <hal.h>
```

Initializes core system hardware components during the boot sequence.

This function performs essential low-level hardware initialization that must occur early in the system startup process. It configures fundamental hardware subsystems including system timers, processor cache settings, and CPU clock frequencies to establish the basic operational environment for the NetBurner device. This initialization is performed using fixed hardware parameters and does not depend on user configuration settings or stored preferences.

Note

This function is typically called automatically during system boot and should not be called directly by user applications. Must be called before any code that depends on accurate timing or cache operations. The initialization sequence is critical and must complete successfully for proper system operation. Clock speed configuration affects all subsequent timing-dependent operations.

See also

[PostConfigHardwareInit\(\)](#) for configuration-dependent hardware initialization

[init\(\)](#) for the main system initialization sequence

14.22.2.25 PostConfigHardwareInit()

```
void PostConfigHardwareInit ()  
#include <hal.h>
```

Initializes hardware components that require configuration data to be loaded.

This function performs hardware initialization for subsystems that depend on stored configuration variables or user settings. It is called after configuration data has been loaded and processed, allowing hardware components like the watchdog timer to be configured according to user preferences or stored settings. This separation ensures that hardware initialization can be customized based on application requirements or stored configuration parameters.

Note

This function is typically called automatically during system initialization after configuration data has been loaded and should not be called directly by user applications. Must be called after configuration variables are available and validated. Watchdog configuration affects system stability and recovery behavior. Some hardware features may remain inactive until this initialization completes.

See also

[HardwareSetup\(\)](#) for core hardware initialization that occurs before configuration loading
[init\(\)](#) for the complete system initialization sequence

14.22.2.26 spaceleft()

```
uint32_t spaceleft ()  
#include <hal.h>
```

Report how much free unallocated space is left in dynamic memory.

Returns

Number of bytes unallocated in dynamic memory

14.22.2.27 StdioCheckIntc()

```
void StdioCheckIntc (  
    void )  
#include <hal.h>
```

Check STDIO interrupt sources.

Checks the interrupt sources used in the system along with their associated interrupt level and priority, and also checks for any conflicts in using the same level and priority between different sources. These results are sent to stdout.

See also

[SysLogCheckIntc\(\)](#)

14.22.2.28 SysLogCheckIntc()

```
void SysLogCheckIntc (  
    void )  
#include <hal.h>
```

This is just like the [StdioCheckIntc\(\)](#) function, except that the results are displayed via UDP.

See also

[StdioCheckIntc\(\)](#)

14.22.3 Variable Documentation

14.22.3.1 HalTickMaxCount

```
uint32_t HalTickMaxCount [extern]
```

```
#include <hal.h>
```

Rollover value for the system hardware tick timer.

Returns

The max/rollover counter value used to generate the tick count

14.22.3.2 watchdog_service_function

```
void(* watchdog_service_function) (void) (
```

```
void ) [extern]
```

```
#include <hal.h>
```

Watchdog callback service function.

If the following function pointer is set to point at a function of the form:

```
void YourFunc( void );
```

Then the system will call this function throughout the AutoUpdate/TCPUpdate process to make sure that the watchdog timer is serviced.

14.23 HTTP and HTML Functions

Classes

- struct [HTTP_Request](#)
HTTP Request Structure.
- class [HtmlPageHandler](#)
Base class for HTTP request handlers that process requests for specific URLs and HTTP methods.
- class [CallBackFunctionPageHandler](#)
Implements the [HtmlPageHandler](#) class as a function pointer callback for GET requests.
- class [CallBackFunctionPostHandler](#)
Implements the [HtmlPostHandler](#) class as a function pointer callback for POST requests.
- class [HtmlPostVariableListCallback](#)
Implements the [HtmlPostVariableListHandler](#) class as a function pointer callback for HTTP POST submissions.

Typedefs

- typedef int [http_gethandlerfunc](#)(int sock, [HTTP_Request](#) &pd)
Implements the [HtmlPageHandler](#) class as a function pointer callback for GET requests.
- typedef int [http_posthandler](#)(int sock, [HTTP_Request](#) &httpReqInfo)
Type definition of the [HtmlPostHandler](#) callback for POST requests.

Enumerations

- enum [HTTP_RequestTypes](#) {
 [tUnknown](#) , [tGet](#) , [tPost](#) , [tDelete](#) ,
 [tPut](#) , [tHead](#) }
HTTP request types for HTTP page handler callback functions.
- enum [HTTP_ACCESS](#) {
 [HTTP_OK_TO_SERVE](#) , [HTTP_NEED_PASSWORD](#) , [HTTP_NOTFOUND](#) , [HTTP_FORBIDEN](#) ,
 [HTTP_RESPONSE_HANDLED](#) }
HTTP page access return values.
- enum [HTTP_REQ_RD_RESULT](#) { [HTTP_RRR_BUF_FULL](#) = 0 , [HTTP_RRR_FINISHED](#) = 1 ,
 [HTTP_RRR_END_FOUND](#) = 2 , [HTTP_RRR SOCK_ERR](#) = -1 }
HTTP Request Read Result.

Functions

- int [SendEmailResponse](#) (int sock, const char *name, const char *attachment)
Send an email with HTML formatting.
- int [SendFullResponse](#) (char const *name, int fd)
Send a file with the proper HTTP header, file specified by file name.
- int [SendFullResponse](#) (HTML_FILE_RECORD *fr, int fd, const char *pUrl)
Send a file with the proper HTTP header, file specified by HTML_FILE_RECORD.
- int [SendHeaderResponse](#) (char const *name, int fd)
Send a HTTP header response for the specified file type.
- int [SendHeaderResponse](#) (HTML_FILE_RECORD *fr, int fd)
Send a HTTP header response for the specified HTML_FILE_RECORD type.
- int32_t [SendFileFragment](#) (char const *name, int32_t fd, PCSTR url=NULL)
Send a file fragment without a header.
- HTML_FILE_RECORD * [GetRecordFromName](#) (char const *name)
Returns a pointer to a HTML_FILE_RECORD for the specified file name.
- CONFIG_RENDER_OBJ [ConfigRenderFunc](#) (int mode, const char *pobj, int len=20, const char *extra=0)
Render a configuration object as HTML for the specified configuration object.
- CONFIG_RENDER_OBJ [ConfigRenderFunc](#) (int mode, config_leaf &cl, int len=20, const char *extra=0)
Render a configuration object as HTML for the specified configuration leaf object.
- void [WriteHtmlVariable](#) (int fd, CONFIG_RENDER_OBJ co)
Send a CONFIG_RENDER_OBJ to the client.
- HTTP_Request * [GetActiveHttpRequest](#) ()
Get the current active running http request. Only valid from within http get or post handling. Also valid during page fill in operations.
- HTTP_ACCESS [CheckHttpAccess](#) (int sock, int access_level, HTTP_Request &Req)
All HTTP requests go through this function.
- void [StartHttp](#) (uint16_t port, bool RunConfigMirror)
Start the HTTP web server. Further documentation in the Initialization section [Initialization - System Initialization Functions](#).
- void [StopHttp](#) ()
Stop the HTTP web server.
- void [SendHTMLHeader](#) (int sock)
Send a HTML response header.
- void [SendHTMLHeaderWCookie](#) (int sock, char *cookie)
Send a HTML response header and cookie.
- void [SendTextHeader](#) (int sock)
Send a HTML plain text header.
- void [SendGifHeader](#) (int sock)
Send a HTML GIF header.
- void [EmptyResponse](#) (int sock)
Send an empty response back.
- void [NoContentResponse](#) (int sock)
Send a no content response back.
- void [RedirectResponse](#) (int sock, PCSTR new_page)
Redirect a HTTP request to a different page.
- void [NotFoundResponse](#) (int sock, PCSTR new_page)
Send a page not found response.
- void [ForbiddenResponse](#) (int sock, PCSTR new_page)
Send a page is forbidden response.
- void [NotAvailableResponse](#) (int sock, PCSTR new_page)
Send a response indicating that the requested resource is not available at this time.

- void [BadRequestResponse](#) (int sock, PCSTR url, PCSTR data)
Send a response indicating that the client request itself is faulty in some manner.
- int [writeallsafestring](#) (int fd, PCSTR str)
Send a string and escape all special characters.
- int [writesafestring](#) (int fd, PCSTR str, size_t strLength)
Send a string with a specified length and escape all special characters.
- int [httpstricmp](#) (PCSTR str1, PCSTR str1sUpper2)
Special string compare. Returns 1 if the strings match until one string ends with a null (0).
- void [StartHttps](#) (uint16_t ssl_port, uint16_t http_port)
Start the HTTPS secure web server.
- void [EnableRemoteConsole](#) ()
Enables the remote webconsole feature.

14.23.1 Detailed Description

```
#include< htmlfiles.h>
```

API to process and delivery files from the web server to clients. In particular, processing of files in a project's html folder.

For the majority of applications these functions will not be necessary; the web server system will automatically handle both static and dynamic content. This API is available if more advances processing is required, such as sending custom response headers or very large amounts of data.

```
#include< http.h>
```

In addition to the functions to start a web server, [StartHttp\(\)](#), this module provides functions used to create a custom web interface. Please refer to the Programmers Guide for additional details on web interface programming and dynamic content.

See also

[StartHttps\(\)](#)

```
#include< httppost.h>
```

```
#include< https.h>
```

14.23.2 Typedef Documentation

14.23.2.1 http_gethandlerfunc

```
typedef int http_gethandlerfunc(int sock, HTTP_Request &pd)
```

```
#include <http.h>
```

Implements the [HtmlPageHandler](#) class as a function pointer callback for GET requests.

Return values

0	Not handled, no changes are made
1	Handled
2	TCP socket will be kept. Socket will get closed by some other method. For example, a large file could be sent from a different task. Any return value other than 2 means the page was processed normally.

14.23.2.2 http_posthandler

```
typedef int http_posthandler(int sock, HTTP\_Request &httpReqInfo)
#include <httppost.h>
```

Type definition of the HtmlPostHandler callback for POST requests.

This should only be used if the specific variable callback implementations will not work in your application. This is mostly an internal use function.

Parameters

<i>sock</i>	The socket the request came in on.
<i>httpReqInfo</i>	Information about the request HTTP_Request .

Return values

0	Not handled, no changes are made
1	Handled
2	TCP socket will be kept. Socket will get closed by some other method. For example, a large file could be sent from a different task. Any return value other than 2 means the page was processed normally.

See also

[HtmlPostVariableListCallback](#)

HtmlPostVariableListHandler

14.23.3 Enumeration Type Documentation

14.23.3.1 HTTP_ACCESS

```
enum HTTP\_ACCESS
```

```
#include <http.h>
```

HTTP page access return values.

Return codes for web page function return codes such as page not found, and if the username/password access group level is granted.

Enumerator

HTTP_OK_TO_SERVE	Ok to serve the page.
HTTP_NEED_PASSWORD	Either the password in the HTTP request is wrong or missing.
HTTP_NOTFOUND	Report the page is not found.
HTTP_FORBIDEN	Report the page forbidden.
HTTP_RESPONSE_HANDLED	This means this function handled the full response, including closing the socket.

14.23.3.2 HTTP_REQ_RD_RESULT

```
enum HTTP\_REQ\_RD\_RESULT
```

```
#include <http.h>
```

HTTP Request Read Result.

Enumerator

HTTP_RRR_BUF_FULL	Buffer full, more data remains to be read.
HTTP_RRR_FINISHED	Read read all content as indicated by headers.
HTTP_RRR_END_FOUND	Read found the trailing \r\n and terminated.
HTTP_RRR_SOCK_ERR	Socket returned error while having no data, probably closed.

14.23.3.3 HTTP_RequestTypes

```
enum HTTP_RequestTypes
```

```
#include <http.h>
```

HTTP request types for HTTP page handler callback functions.

Enumerator

tUnknown	The type of request is not yet known.
tGet	GET request.
tPost	POST request.
tDelete	Delete Request.
tPut	Put request.
tHead	Header request, does not include content.

14.23.4 Function Documentation

14.23.4.1 BadRequestResponse()

```
void BadRequestResponse (
    int sock,
    PCSTR url,
    PCSTR data)
```

```
#include <http.h>
```

Send a response indicating that the client request itself is faulty in some manner.

Parameters

<i>sock</i>	Socket to send response
<i>url</i>	The URL of the page to display
<i>data</i>	An explanation or segment of data to display indicating the fault

See also

[RedirectResponse\(\)](#), [NotFoundResponse\(\)](#), [ForbiddenResponse\(\)](#)

[NotAvailableResponse\(\)](#)

14.23.4.2 CheckHttpAccess()

```
HTTP_ACCESS CheckHttpAccess (
    int sock,
    int access_level,
    HTTP_Request & Req)
```

```
#include <http.h>
```

All HTTP requests go through this function.

By default the library provides a weak reference implementation that approves all transactions. To provide your own just add an implementation of this function to your code.

Parameters

<i>sock</i>	The network socket the request came in on.
<i>access_level</i>	The user defined access level for this request from 0 to 255. A value of 0 means no password is required.
<i>Req</i>	The HTTP request where one can extract the URL , access type, password, etc

Returns

[HTTP_ACCESS](#)

14.23.4.3 ConfigRenderFunc() [1/2]

```
CONFIG_RENDER_OBJ ConfigRenderFunc (
    int mode,
    config_leaf & cl,
    int len = 20,
    const char * extra = 0)
```

```
#include <htmlfiles.h>
```

Render a configuration object as HTML for the specified configuration leaf object.

Parameters

<i>mode</i>	CONFIG_RENDER_OBJ eMode value
<i>cl</i>	Reference to the config_leaf object to render
<i>len</i>	Optional length parameter for text input elements
<i>extra</i>	Pointer to any additional information to be added to the HTML input element. Will be added to the end of the <input> tag.

Returns

A CONFIG_RENDER_OBJ that can be sent to the client using WriteHtmlVariable()

14.23.4.4 ConfigRenderFunc() [2/2]

```
CONFIG_RENDER_OBJ ConfigRenderFunc (
    int mode,
    const char * pobj,
    int len = 20,
    const char * extra = 0)
```

```
#include <htmlfiles.h>
```

Render a configuration object as HTML for the specified configuration object.

Parameters

<i>mode</i>	CONFIG_RENDER_OBJ eMode value
<i>pobj</i>	Pointer to a configuration object. Can be specified as the JSON configuration path. For example, "Config/Sys/NetIf/Ethernet0/StaticAddr".
<i>len</i>	Optional length parameter for text input elements
<i>extra</i>	Pointer to any additional information to be added to the HTML input element. Will be added to the end of the <input> tag.

Returns

A CONFIG_RENDER_OBJ that can be sent to the client using WriteHtmlVariable()

14.23.4.5 EmptyResponse()

```
void EmptyResponse (
    int sock)
#include <http.h>
```

Send an empty response back.

Parameters

<i>sock</i>	Socket to send response

See also

[ForbiddenResponse\(\)](#), [ForbiddenResponse\(\)](#), [SendHTMLHeaderWCookie\(\)](#)
[NotAvailableResponse\(\)](#), [BadRequestResponse\(\)](#), [RedirectResponse\(\)](#)

14.23.4.6 EnableRemoteConsole()

```
void EnableRemoteConsole ()
#include <remoteconsole.h>
```

Enables the remote webconsole feature.

Creates or uses the existing web server and a console.html file as well as a WebSocket listener that sends and receives stdout/stdin such that you can access the "serial" console remotely via a web browser.

See the [Remote Web Console for Serial/stdout](#) for more

Note that there is no security on this by default, so it is best used for development purposes.

14.23.4.7 ForbiddenResponse()

```
void ForbiddenResponse (
    int sock,
    PCSTR new_page)
#include <http.h>
```

Send a page is forbidden response.

Parameters

<i>sock</i>	Socket to send response
<i>new_page</i>	The URL of the page to display

See also

[RedirectResponse\(\)](#), [NotFoundResponse\(\)](#)
[NotAvailableResponse\(\)](#), [BadRequestResponse\(\)](#)

14.23.4.8 GetRecordFromName()

```
HTML_FILE_RECORD * GetRecordFromName (
    char const * name)
#include <htmlfiles.h>
```

Returns a pointer to a HTML_FILE_RECORD for the specified file name.

Parameters

<i>name</i>	Name of file in htmldata.cpp to search for

Return values

<i>A</i>	pointer to a HTML_FILE_RECORD if found
<i>NULL</i>	if file name not found

14.23.4.9 httpstricmp()

```
int httpstricmp (
    PCSTR str1,
    PCSTR strIsUpper2)
```

#include <http.h>

Special string compare. Returns 1 if the strings match until one string ends with a null (0).

strIsUpper2 must be specified in upper case

Used to match URL with stored file prefixes.

Parameters

<i>str1</i>	The string to search
<i>strIsUpper2</i>	The string to search for, must be specified in upper case

Return values

0	if the prefixes do not match
1	if the prefixes match

Example: httpstricmp(str1, "LED.HTML");

str1	Return Value
LED	1
led.HTML	1
led.html	1
LED.HTM?	0

14.23.4.10 NoContentResponse()

```
void NoContentResponse (
    int sock)
```

#include <http.h>

Send a no content response back.

Parameters

<i>sock</i>	Socket to send response

See also

[ForbiddenResponse\(\)](#), [ForbiddenResponse\(\)](#), [SendHTMLHeaderWCookie\(\)](#)
[NotAvailableResponse\(\)](#), [BadRequestResponse\(\)](#), [RedirectResponse\(\)](#)

14.23.4.11 NotAvailableResponse()

```
void NotAvailableResponse (
    int sock,
    PCSTR new_page)
```

#include <http.h>

Send a response indicating that the requested resource is not available at this time.

Parameters

<i>sock</i>	Socket to send response
<i>new_page</i>	The URL of the page to display

See also

[RedirectResponse\(\)](#), [NotFoundResponse\(\)](#), [ForbiddenResponse\(\)](#)
[BadRequestResponse\(\)](#)

14.23.4.12 NotFoundResponse()

```
void NotFoundResponse (
    int sock,
    PCSTR new_page)
#include <http.h>
Send a page not found response.
```

Parameters

<i>sock</i>	Socket to send response
<i>new_page</i>	The URL of the page to display

See also

[ForbiddenResponse\(\)](#), [RedirectResponse\(\)](#)
[NotAvailableResponse\(\)](#), [BadRequestResponse\(\)](#)

14.23.4.13 RedirectResponse()

```
void RedirectResponse (
    int sock,
    PCSTR new_page)
#include <http.h>
Redirect a HTTP request to a different page.
```

Parameters

<i>sock</i>	Socket to send response
<i>new_page</i>	The URL of the new page

See also

[ForbiddenResponse\(\)](#), [ForbiddenResponse\(\)](#), [SendHTMLHeaderWCookie\(\)](#)
[NotAvailableResponse\(\)](#), [BadRequestResponse\(\)](#), [EmptyResponse\(\)](#)

14.23.4.14 SendEmailResponse()

```
int SendEmailResponse (
    int sock,
    const char * name,
    const char * attachment)
#include <htmlfiles.h>
Send an email with HTML formatting.
```

Parameters

<i>sock</i>	Network socket in which to send the data
<i>name</i>	Name of file from project's html folder, this is the email body
<i>attachment</i>	

Return values

1	Success, the file specified the name parameter was found
0	File does not exist

14.23.4.15 SendFileFragment()

```
int32_t SendFileFragment (
    char const * name,
    int32_t fd,
    PCSTR url = NULL)
```

```
#include <htmlfiles.h>
```

Send a file fragment without a header.

Search through the files stored in the system by comphtml in the project's html folder for the specified file name. If found, the file content is sent as a file fragment. A SendHeaderResponse function must be sent prior to sending file fragments.

This function is useful when building HTML responses with large amounts of data. If the file has embedded dynamic HTML function tags and/or variable tags, such a CPPCALL and VARIABLE, they will be processed as the content is delivered.

Parameters

<i>name</i>	Name of the file
<i>fd</i>	File descriptor used to send the file fragment
<i>url</i>	Optional URL string in fragment header

Return values

1	if the file was found
0	otherwise

14.23.4.16 SendFullResponse() [1/2]

```
int SendFullResponse (
    char const * name,
    int fd)
```

```
#include <htmlfiles.h>
```

Send a file with the proper HTTP header, file specified by file name.

This function searches through the files stored in the system by comphtml in the project's html folder for the specified file name. If found, it sends the proper HTTP header and file to the client. If the file has embedded dynamic HTML function tags and/or variable tags, such a CPPCALL and VARIABLE, they will be processed as the content is delivered.

Parameters

<i>name</i>	Name of the file
<i>fd</i>	File descriptor used to send the file

Return values

1	Succes, file was found
0	File does not exist

See also

[SendFullResponse\(HTML_FILE_RECORD *fr, int fd, const char *pUrl\)](#)

14.23.4.17 SendFullResponse() [2/2]

```
int SendFullResponse (
    HTML_FILE_RECORD * fr,
    int fd,
    const char * pUrl)
```

```
#include <htmlfiles.h>
```

Send a file with the proper HTTP header, file specified by HTML_FILE_RECORD.

While the [SendFullResponse\(char const *name, int fd\)](#) function will resolve the file name, this lower level function requires knowledge of the file pointer to the HTML_FILE_RECORD in htmldata.cpp.

Parameters

<i>fr</i>	Pointer to the HTML_FILE_RECORD in htmldata.cpp
<i>fd</i>	File descriptor used to send the file
<i>pUrl</i>	Pointer to a URL string used by response functions such a CPPCALL and VARIABLE callbacks. In default web server processing the URL is automatically set to what the client requested. When processing web requests manually you must specify the URL string. If you are not using dynamic content that relies on the URL, this can be set to NULL.

Return values

<i>1</i>	if the file was found
<i>0</i>	otherwise

See also

[SendFullResponse\(char const *name, int fd\)](#)

14.23.4.18 SendGifHeader()

```
void SendGifHeader (
    int sock)
```

```
#include <http.h>
```

Send a HTML GIF header.

Sends a GIF response header. Can be useful when dynamically creating your own GIF images.

Parameters

<i>sock</i>	Socket to send response

See also

[SendHTMLHeader\(\)](#), [SendTextHeader\(\)](#), [SendHTMLHeaderWCookie\(\)](#)

14.23.4.19 SendHeaderResponse() [1/2]

```
int SendHeaderResponse (
    char const * name,
    int fd)
```

```
#include <htmlfiles.h>
```

Send a HTTP header response for the specified file type.

This function searches through the files stored in the system by comphtml in the project's html folder for the specified file name. If found, it sends the proper HTTP header only to the client. Unlike a full response, only the header is sent.

Parameters

<i>name</i>	Name of the file
<i>fd</i>	File descriptor used to send the file

Return values

1	Succes, file was found
0	File does not exist

See also

[SendHeaderResponse\(HTML_FILE_RECORD *fr, int fd\)](#)

14.23.4.20 SendHeaderResponse() [2/2]

```
int SendHeaderResponse (
    HTML_FILE_RECORD * fr,
    int fd)
```

```
#include <htmlfiles.h>
```

Send a HTTP header response for the specified HTML_FILE_RECORD type.

While the [SendFullResponse\(HTML_FILE_RECORD *fr, int fd, const char *pUrl\)](#); function will resolve the file name, this lower level function requires knowledge of the file pointer to the HTML_FILE_RECORD array in `htmldata.cpp`.

Parameters

<i>fr</i>	Pointer to the HTML_FILE_RECORD in <code>htmldata.cpp</code>
<i>fd</i>	File descriptor used to send the file

Return values

1	Succes, file was found
0	File does not exist

See also

[SendHeaderResponse\(char const *name, int fd\)](#)

14.23.4.21 SendHTMLHeader()

```
void SendHTMLHeader (
    int sock)
```

```
#include <http.h>
```

Send a HTML response header.

Used to build your own HTML response

Parameters

<i>sock</i>	Socket to send response

See also

[SendHTMLHeaderWCookie\(\)](#), [SendTextHeader\(\)](#), [SendGifHeader\(\)](#)

14.23.4.22 SendHTMLHeaderWCookie()

```
void SendHTMLHeaderWCookie (  
    int sock,  
    char * cookie)
```

```
#include <http.h>
```

Send a HTML response header and cookie.

Sends a header as well as a cookie to be stored by the client browser.

Parameters

<i>sock</i>	Socket to send response
<i>cookie</i>	Pointer to the cookie

See also

[SendHTMLHeader\(\)](#), [SendGifHeader\(\)](#), [SendTextHeader\(\)](#)

14.23.4.23 SendTextHeader()

```
void SendTextHeader (  
    int sock)
```

```
#include <http.h>
```

Send a HTML plain text header.

Use as the first part of building your own plain text response.

Parameters

<i>sock</i>	Socket to send response

See also

[SendHTMLHeader\(\)](#), [SendGifHeader\(\)](#), [SendHTMLHeaderWCookie\(\)](#)

14.23.4.24 StartHttp()

```
void StartHttp (  
    uint16_t port,  
    bool RunConfigMirror)
```

```
#include <http.h>
```

Start the HTTP web server. Further documentation in the Initialization section [Initialization - System Initialization Functions](#).

If no parameters are specified it will listen on port 80, and enable the configuration mirror feature so an application can create its own custom configuration web page.

Parameters

<i>port</i>	Listen port, defaults to port 80
<i>RunConfigMirror</i>	Enable configuration mirror, default is true

See also

[StopHttp\(\)](#), [StartHttps\(\)](#)

14.23.4.25 StartHttps()

```
void StartHttps (
    uint16_t ssl_port,
    uint16_t http_port)
```

```
#include <https.h>
```

Start the HTTPS secure web server.

If no parameters are specified, the web server will listen on ports 443 and 80, and enable the configuration mirror feature so an application can create its own custom configuration web page. If no user certificate/key is installed, the default system key will be used.

Default parameter values defined in [init.h](#)

Parameters

<i>ssl_port</i>	Secure listen port, defaults to port 443
<i>http_port</i>	Non-secure listen port, defaults to port 80

See also

[StartHttp\(\)](#), [StopHttp\(\)](#)

14.23.4.26 StopHttp()

```
void StopHttp ()
```

```
#include <http.h>
```

Stop the HTTP web server.

See also

[StartHttp\(\)](#)

14.23.4.27 writeallsafestring()

```
int writeallsafestring (
    int fd,
    PCSTR str)
```

```
#include <http.h>
```

Send a string and escape all special characters.

When sending a HTML text response, certain characters (e.g. '<') are interpreted by the browser as formatting. This function properly escapes the text so it will appear as intended.

Parameters

<i>fd</i>	File descriptor used to send response
<i>str</i>	String to send

Returns

The number of characters sent

14.23.4.28 WriteHtmlVariable()

```
void WriteHtmlVariable (
    int fd,
    CONFIG_RENDER_OBJ co)
```

```
#include <htmlfiles.h>
```

Send a CONFIG_RENDER_OBJ to the client.

The CONFIG_RENDER_OBJ must have been previously created with a [ConfigRenderFunc\(\)](#) function.

Parameters

<i>fd</i>	File descriptor to send data to
<i>co</i>	CONFIG_RENDER_OBJ to send

14.23.4.29 writesafestring()

```
int writesafestring (
    int fd,
    PCSTR str,
    size_t strLength)
```

```
#include <http.h>
```

Send a string with a specified length and escape all special characters.

When sending a HTML text response, certain characters (e.g. '<') are interpreted by the browser as formatting. This function properly escapes the text so it will appear as intended.

Parameters

<i>fd</i>	File descriptor used to send response
<i>str</i>	String to send
<i>strLength</i>	number of characters from str to send

Returns

The number of characters sent

14.24 I2C

Topics

- [Bit Bang I2C \(GPIO\)](#)
- [MOD5441x, NANO54415, SB800EX](#)
- [MODM7AE70, SBE70LC](#)
- [Wire Interface](#)

Implements the same [TwoWire](#) interface as other popular microcontrollers for easy cross-compatibility with third party libraries.

14.24.1 Detailed Description

[I2C](#) API for NetBurner Platforms

The [I2C](#) peripheral interfaces can be controlled four ways:

- The [Wire Interface](#) (recommended), which offers an interface more familiar to users of other platforms.
- The [I2CDevice](#) class (recommended for older hardware)
- The [I2C](#) class, which offers additional low level control for older hardware
- The [WireIntf](#) class, which offers a Two Wire Interface API for older hardware

```
#include< Wire.h >
or
#include< i2c.h > (in the appropriate arch include subfolder)
or
#include< bb\_i2c.h > (if no i2c.h in the appropriate arch)
```

[BBI2C](#) is specifically for platforms without native hardware [I2C](#) support. See other sub-modules for device-specific [I2C](#) details.

Provides access to the all the [I2C](#) peripherals on the processor. The [I2C](#) peripherals available will vary by platform. For example, `Wire.read();` will execute a read operation on [I2C](#) module 0.

14.24.2 Bit Bang I2C (GPIO)

Classes

- class [BBI2C](#)
I2C Peripheral Class.

14.24.2.1 Detailed Description

```
#include< bb_i2c.h>
```

14.24.3 MOD5441x, NANO54415, SB800EX

Functions

- void [MultiChannel_I2CInit](#) (int moduleNum=DEFAULT_I2C_MODULE, uint8_t slave_Addr=0x08, uint8_t freqdiv=0x3C)
Initialize the I2C peripheral module.
- uint8_t [MultiChannel_I2CSendBuf](#) (int moduleNum, uint8_t addr, uint8_t buf, int num, bool stop=true)
Send a buffer of bytes to an I2C device.
- uint8_t [MultiChannel_I2CReadBuf](#) (int moduleNum, uint8_t addr, uint8_t buf, int num, bool stop=true)
Read a number of bytes from an I2C device and store in the specified buffer.
- void [I2CMultiChannelResetPeripheral](#) (int moduleNum)
Reset the specified I2C peripheral module.
- uint8_t [MultiChannel_I2CRestart](#) (int moduleNum, uint8_t addr, bool Read_Not_Write, uint32_t ticks_to_wait=I2C_RX_TX_TIMEOUT)
Restart communication with a I2C device.
- uint8_t [MultiChannel_I2CStart](#) (int moduleNum, uint8_t addr, bool Read_Not_Write, uint32_t ticks_to_wait=I2C_START_TIMEOUT)
Send an I2C start to an I2C device to begin communication.
- uint8_t [MultiChannel_I2CStop](#) (int moduleNum=DEFAULT_I2C_MODULE, uint32_t ticks_to_wait=I2C_RX_TX_TIMEOUT)
Issue an I2C stop terminate communication with an I2C device and release the bus.
- uint8_t [MultiChannel_I2CSend](#) (int moduleNum, uint8_t val, uint32_t ticks_to_wait=I2C_RX_TX_TIMEOUT)
Send a single byte on the I2C bus.
- uint8_t [MultiChannel_I2CRead](#) (int moduleNum, uint8_t val, uint32_t ticks_to_wait=I2C_RX_TX_TIMEOUT)
Read a single byte from the I2C bus.

Multichannel I2C Return Values

Function return values

- #define **I2C_OK** (0)
Last instruction terminated correctly.
- #define **I2C_NEXT_WRITE_OK** (1)
I2C bus is OK for a write.
- #define **I2C_NEXT_READ_OK** (2)
I2C bus is OK for a read.
- #define **I2C_MASTER_OK** (3)
I2C finished transmission but still owns bus (need to stop or restart)
- #define **I2C_TIMEOUT** (4)
A timeout occurred while trying communicate on I2C bus.
- #define **I2C_BUS_NOT_AVAIL** (5)

- *A timeout occurred while trying gain I2C bus control.*
- **#define I2C_NOT_READY (6)**
A read or write was attempted before I2C ready or during a slave transmission.
- **#define I2C_LOST_ARB (7)**
Lost arbitration during start.
- **#define I2C_LOST_ARB_ADD (8)**
Lost arbitration and then winner addressed our slave address.
- **#define I2C_NO_LINK_RX_ACK (9)**
We are in Master TX mode and received no ACK from slave device, possibly during start.

Multichannel I2C Macros

I2C Macros

- **#define I2C_SR_BUSY (((0x20 & I2C_SR) == 0x20))**
Bus is busy (bit 5 of I2SR)
- **#define I2C_CR_SLAVE (((0x20 & I2C_CR) == 0x00))**
Bus set as slave (bit 5 of I2CR)
- **#define I2C_SR_ARB_LOST (((0x10 & I2C_SR) == 0x10))**
Bus arbitration was lost (bit 4 of I2SR)
- **#define I2C_SR_ADRES_AS_SLAVE (((0x40 & I2C_SR) == 0x40))**
Addressed as a slave (bit 6 of I2SR)
- **#define I2C_SR_SLAVE_TX (((0x04 & I2C_SR) == 0x04))**
State of read/write bit of the received address command (bit 2 of I2SR)
- **#define I2C_CR_TX (((0x10 & I2C_CR) == 0x10))**
Configured for transmit (bit 4 of I2CR)
- **#define I2C_SR_RX_ACK (((0x01 & I2C_SR) == 0x00))**
Received a RX ACK after last transmit (bit 0 of I2SR)
- **#define I2C_CR_RX_ACK (((0x08 & I2C_CR) == 0x00))**
Configured to RX Ack.
- **#define I2C_SET_NO_ACK ((I2C_CR |= 0x08))**
Configure I2C module not to send a RX ACK (bit 3 of I2CR)
- **#define I2C_SET_ACK ((I2C_CR &= 0xF7))**
Configure I2C module to send a RX ACK (bit 3 of I2CR)
- **#define I2C_SET_TX ((I2C_CR |= 0x10))**
Configure I2C module to be in TX mode (bit 4 of I2CR)
- **#define I2C_SET_RX ((I2C_CR &= 0xEF))**
Configure I2C module to be in RX mode (bit 4 of I2CR)
- **#define I2C_SET_REPEAT_START ((I2C_CR |= 0x04))**
Configure I2C to send a repeated start signal.
- **#define I2C_CLR_ARB_LOST ((I2C_SR &= 0xEF))**
Clear Arbitration lost error condition.

14.24.3.1 Detailed Description

Supported platforms:

- MOD54415
- MOD54417
- NANO54415

- SB800EX

The platforms have multiple I2C interfaces called "modules". The available I2C modules available on a particular platform are determined by the connector configuration of that platform. Available I2C module numbers are defined below. Since the processor pins are multiplexed with many other functions, the modules chosen are not sequential. The default module number is 0.

I2C MODULE NUMBERS BY PLATFORM:

- MOD5441x: 0, 1, 2, 4, 5.
- NANO54415: 0, 1, 4, 5.

The most common I2C functions used are listed below. They automatically handle I2C start and stop conditions, as well as checking for I2C bus errors and recovery:

- [MultiChannel_I2CInit \(\)](#)
- [MultiChannel_I2CSendBuf \(\)](#)
- [MultiChannel_I2CReadBuf \(\)](#)
- [I2CMultiChannelResetPeripheral\(\)](#)

14.24.3.2 Function Documentation

14.24.3.2.1 I2CMultiChannelResetPeripheral()

```
void I2CMultiChannelResetPeripheral (
    int moduleNum)
#include <multichanneli2c.h>
```

Reset the specified I2C peripheral module.
Clears all interrupts, I2C conditions, and releases the bus.

Parameters

<i>moduleNum</i>	I2C peripheral module number.

14.24.3.2.2 MultiChannel_I2CInit()

```
void MultiChannel_I2CInit (
    int moduleNum = DEFAULT_I2C_MODULE,
    uint8_t slave_Addr = 0x08,
    uint8_t freqdiv = 0x3C)
#include <multichanneli2c.h>
```

Initialize the I2C peripheral module.

The frequency is based on the internal bus clock of the processor. The formula is: $\text{Freq} = (250\text{MHZ (sys clock)}/2) / \text{freqdiv}$ to give max baud rate of the master mode I2C bus clock. Values for freqdiv are found in the i2fdr section of the NXP MCF5441x User Manual.

Parameters

<i>moduleNum</i>	I2C peripheral module number.
<i>slave_Addr</i>	I2C address of the module. If not specified, default value is 0x08.
<i>freqdiv</i>	I2C module frequency divider. If not specified, default value is 0x3C, which is the maximum value of 100KHz.

14.24.3.2.3 MultiChannel_I2CRead()

```
uint8_t MultiChannel_I2CRead (
    int moduleNum,
    uint8_t val,
    uint32_t ticks_to_wait = I2C_RX_TX_TIMEOUT)
```

```
#include <multichanneli2c.h>
```

Read a single byte from the [I2C](#) bus.

Reads a single byte from the [I2C](#) bus and stores it in val. A start or restart must have been previously issued to become the bus master, and a stop must be issued to release the bus when you are finished reading the desired number of bytes.

Does not handle no Ack of last byte, which needs to have a `I2C_SET_NO_ACK` called before the last read.

Parameters

<i>moduleNum</i>	I2C peripheral module number.
<i>val</i>	Location to store the byte read.
<i>ticks_to_wait</i>	Amount of time to wait for a start transmission to complete, specified as system time ticks. The <code>TICKS_PER_SECOND</code> macro can be used with a multiplier or divisor. Includes the time to receive an ack from the I2C device. If not specified, the default time of <code>I2C_RX_TX_TIMEOUT</code> will be used.

Returns

The state of the [I2C](#) bus: [I2C Return Values](#)

14.24.3.2.4 MultiChannel_I2CReadBuf()

```
uint8_t MultiChannel_I2CReadBuf (
    int moduleNum,
    uint8_t addr,
    uint8_t buf,
    int num,
    bool stop = true)
```

```
#include <multichanneli2c.h>
```

Read a number of bytes from an [I2C](#) device and store in the specified buffer.

Sends an [I2C](#) start, reads the specified number of bytes into buf, and finishes with or without an [I2C](#) stop (default is stop). If a stop is not issued, for purposes such as a restart, a stop must be called manually.

Parameters

<i>moduleNum</i>	I2C peripheral module number.
<i>addr</i>	I2C device address.
<i>buf</i>	The buffer to store the data.
<i>num</i>	The number of bytes to read.
<i>stop</i>	Send a stop signal at the end of the transaction. If not specified, the default value is true.

Returns

The state of the [I2C](#) bus: [I2C Return Values](#)

14.24.3.2.5 MultiChannel_I2CRestart()

```
uint8_t MultiChannel_I2CRestart (
    int moduleNum,
    uint8_t addr,
```

```

        bool Read_Not_Write,
        uint32_t ticks_to_wait = I2C_RX_TX_TIMEOUT)
#include <multichanneli2c.h>

```

Restart communication with a [I2C](#) device.

Restarts communication with an [I2C](#) device after completion of a read or write command in which a stop was not issued. Enables communicate on bus again without giving up control as the [I2C](#) bus master.

Parameters

<i>moduleNum</i>	I2C peripheral module number.
<i>addr</i>	I2C device address.
<i>Read_Not_Write</i>	Specifies the next bus operation: true = read, false = write.
<i>ticks_to_wait</i>	Amount of time to wait for a start transmission to complete, specified as system time ticks. The TICKS_PER_SECOND macro can be used with a multiplier or divisor. Includes the time to receive an ack from the I2C device. If not specified, the default time of I2C_RX_TX_TIMEOUT will be used.

Returns

The state of the [I2C](#) bus: [I2C Return Values](#)

14.24.3.2.6 MultiChannel_I2CSend()

```

uint8_t MultiChannel_I2CSend (
    int moduleNum,
    uint8_t val,
    uint32_t ticks_to_wait = I2C_RX_TX_TIMEOUT)
#include <multichanneli2c.h>

```

Send a single byte on the [I2C](#) bus.

Sends a single byte on the [I2C](#) bus. A start or restart must have been previously issued to become the bus master, and a stop must be issued to release the bus when you are finished with writing the desired number of bytes.

Parameters

<i>moduleNum</i>	I2C peripheral module number.
<i>val</i>	Byte to send
<i>ticks_to_wait</i>	Amount of time to wait for a start transmission to complete, specified as system time ticks. The TICKS_PER_SECOND macro can be used with a multiplier or divisor. Includes the time to receive an ack from the I2C device. If not specified, the default time of I2C_RX_TX_TIMEOUT will be used.

Returns

The state of the [I2C](#) bus: [I2C Return Values](#)

14.24.3.2.7 MultiChannel_I2CSendBuf()

```

uint8_t MultiChannel_I2CSendBuf (
    int moduleNum,
    uint8_t addr,
    uint8_t buf,
    int num,
    bool stop = true)
#include <multichanneli2c.h>

```

Send a buffer of bytes to an [I2C](#) device.

Sends an [I2C](#) start, the bytes in buf, and and finishes with our without an [I2C](#) stop (default is stop). If a stop is not issued, for purposes such as a restart, a stop must be called manually.

Parameters

<i>moduleNum</i>	I2C peripheral module number.
<i>addr</i>	I2C device address.
<i>buf</i>	The buffer to send.
<i>num</i>	The number of bytes in the buffer.
<i>stop</i>	Send a stop signal at the end of transmission. If not specified, the default value is true.

Returns

The state of the I2C bus: [I2C Return Values](#)

14.24.3.2.8 MultiChannel_I2CStart()

```
uint8_t MultiChannel_I2CStart (
    int moduleNum,
    uint8_t addr,
    bool Read_Not_Write,
    uint32_t ticks_to_wait = I2C_START_TIMEOUT)
```

```
#include <multichanneli2c.h>
```

Send an I2C start to an I2C device to begin communication.

Restarts communication with an I2C device after completion of a read or write command in which a stop was not issued. Enables communicate on bus again without giving up control as the I2C bus master.

Parameters

<i>moduleNum</i>	I2C peripheral module number.
<i>addr</i>	I2C device address.
<i>Read_Not_Write</i>	Specifies the next bus operation: true = read, false = write.
<i>ticks_to_wait</i>	Amount of time to wait for a start transmission to complete, specified as system time ticks. The TICKS_PER_SECOND macro can be used with a multiplier or divisor. Includes the time to receive an ack from the I2C device. If not specified, the default time of I2C_START_TIMEOUT will be used.

Returns

The state of the I2C bus: [I2C Return Values](#)

14.24.3.2.9 MultiChannel_I2CStop()

```
uint8_t MultiChannel_I2CStop (
    int moduleNum = DEFAULT_I2C_MODULE,
    uint32_t ticks_to_wait = I2C_RX_TX_TIMEOUT)
```

```
#include <multichanneli2c.h>
```

Issue an I2C stop terminate communication with an I2C device and release the bus.

Terminate any communication and release the I2C bus.

Parameters

<i>moduleNum</i>	I2C peripheral module number.
<i>ticks_to_wait</i>	Amount of time to wait for a start transmission to complete, specified as system time ticks. The TICKS_PER_SECOND macro can be used with a multiplier or divisor. Includes the time to receive an ack from the I2C device. If not specified, the default time of I2C_RX_TX_TIMEOUT will be used.

Returns

The state of the [I2C](#) bus: [I2C Return Values](#)

14.24.4 MODM7AE70, SBE70LC**Classes**

- class [WireIntf](#)
Wire Interface Class for [I2C](#).
- class [I2C](#)
[I2C](#) Peripheral Class.
- class [I2CDevice](#)
[I2C](#) Device Class (recommended)

14.24.4.1 Detailed Description

Supported Platforms:

- MODM7AE70
- SBE70LC

The [I2C](#) peripheral interfaces can be controlled three ways:

- The [I2CDevice](#) class (recommended)
- The [I2C](#) Class, which offers additional low level control
- The [WireIntf](#) Class, which offers a Two Wire Interface API

14.24.5 Wire Interface

Implements the same [TwoWire](#) interface as other popular microcontrollers for easy cross-compatibility with third party libraries.

Classes

- class [TwoWire](#)
Wire interface class.

14.24.5.1 Detailed Description

Implements the same [TwoWire](#) interface as other popular microcontrollers for easy cross-compatibility with third party libraries.

```
#include< Wire.h>
```

To use, simply instantiate the [TwoWire](#) class in your main.cpp: [TwoWire](#) Wire; See examples in your NNDK under `examples/I2C/Wire`

14.25 IOSYS - I/O System**Typedefs**

- typedef void [FDCallBack](#)(int fd, [FDChangeType](#) change, void *pData)
Define the function signature for file descriptor notification callbacks.

Functions

- int [close](#) (int fd)
Close the specified file descriptor and free the associated resources.
- int [read](#) (int fd, char *buf, int nbytes)
Read data from a file descriptor (fd).
- int [peek](#) (int fd, char *c)
Peek at the data for the specified file descriptor (fd). Will block forever until at least one byte is available to be read. The byte returned is not removed from the fd buffer, it will be the first byte of data returned on any subsequent read operation.
- int [write](#) (int fd, const char *buf, int nbytes)
Write data to the stream associated with a file descriptor (fd). Can be used to write data to stdio, a TCP socket, or a Serial port.
- int [writestring](#) (int fd, const char *str)
Write a null terminated ascii string to the stream associated with a file descriptor (fd). Can be used to write data to stdio, a TCP socket, or a Serial port.
- int [writeall](#) (int fd, const char *buf, int nbytes=0)
Write the specified number of bytes to a file descriptor. Will block until all bytes are sent, or a file descriptor error occurs (such as a TCP socket error). Can be used to write data to stdio, a TCP socket, or a Serial port.
- int [ReadWithTimeout](#) (int fd, char *buf, int nbytes, unsigned long timeout)
Read data from a file descriptor(fd), or return if at least one byte is not read within the specified timeout. Can be used instead of the [read\(\)](#) function, which will block forever until at least one byte is read. File descriptors include such things as stdio, TCP sockets, TLS sockets, and Serial ports.
- int [ReadWithTickTimeout](#) (int fd, char *buf, int nbytes, TickTimeout &timeout)
Same as [ReadWithTimeout\(\)](#), except the timeout value parameter is of type TickTimeout instead of an unsigned long. The TickTimeout object is more robust in terms of sequential timeout calls and rollover protection.
- int [ReadAllWithTimeout](#) (int fd, char *buf, int nbytes, unsigned long timeout)
Attempt to read the specified number of bytes from a file descriptor, or return with the number of bytes read if the timeout value has expired.
- int [ReadAllWithTickTimeout](#) (int fd, char *buf, int nbytes, TickTimeout &timeout)
Same as [ReadWithTimeout\(\)](#), except the timeout value parameter is of type TickTimeout instead of an unsigned long. The TickTimeout object is more robust in terms of sequential timeout calls and rollover protection.
- int [readall](#) (int fd, char *buf, int nbytes)
Read the specified number of bytes from a file descriptor (fd). This function will block until either the requested number of bytes have been read, or a file descriptor error occurs. It can be used to read from stdio, TCP sockets, or Serial ports.
- int [SockPeekWithTimeout](#) (int fd, char *buf, uint32_t maxtime)
This function peeks at data from a file descriptor (fd), with a specified timeout value (as opposed to the peek function which will block forever until at least one byte is available to be read). This function can be used to peek from stdio, TCP sockets, or Serial ports.
- int [getchar](#) ()
Get a character from stdin. Will block if no character is available, from stdio library. The default stdin is the debug serial port.
- int [dataavail](#) (int fd)
Check the specified file descriptor to determine if data is available to be read.
- int [writeavail](#) (int fd)
Check the specified file descriptor to determine data can be written.
- int [haserror](#) (int fd)
Check if a file descriptor has an error.
- int [charavail](#) ()
Checks to see if data is available for read on stdin. By default, stdin is the boot/debug serial port.
- void [RegisterFDCallback](#) (int fd, [FDCallback](#) *fp, void *pData)
Register a callback function to receive notification when an FD state changes. Register a NULL fp to remove the notification.

- void **FD_ZERO** (fd_set *pfdsets)

Clear (set to 0) a fd_set (file descriptor set) so no file descriptors (fds) are selected.
- void **FD_CLR** (int fd, fd_set *pfdsets)

A fd_set (file descriptor set) holds a set of file descriptors (fds). This function clears or removes a specific file descriptor in an fd_set.
- void **FD_SET** (int fd, fd_set *pfdsets)

A fd_set (file descriptor set) holds a set of file descriptors (fds). This function sets or adds a specific file descriptor to an fd_set.
- int **FD_ISSET** (int fd, fd_set *pfdsets)

A fd_set (file descriptor set) holds a set of file descriptors (fds). This function checks whether or not a particular fd is set in the specified fd_set.
- int **FD_OVERLAP** (const fd_set *f1, const fd_set *f2)

A fd_set (file descriptor set) holds a set of file descriptors (fds). This function determines whether there exists a common file descriptor between the two sets.
- void **FD_COPY** (const fd_set *from, fd_set *to)

A fd_set (file descriptor set) holds a set of file descriptors (fds). This function copies one file descriptor set to another.
- void **FD_SETFROMSET** (const fd_set *from, fd_set *to)

A fd_set (file descriptor set) holds a set of file descriptors (fds). This function adds the file descriptors from one set to another.
- void **FD_CLRFROMSET** (const fd_set *from, fd_set *to)

A fd_set (file descriptor set) holds a set of file descriptors (fds). This function remove the file descriptors in one set from another.
- int **select** (int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds, unsigned long timeout)

Wait for events to occur on one or more I/O resources associated with a set of file descriptors (fds), such as data available to be read, a resource is available to write data, or an error has occurred.
- int **ZeroWaitSelect** (int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds)

Returns whether events have occurred on one or more I/O resources associated with a set of file descriptors (fds), and returns immediately.
- int **ioctl** (int fd, int cmd)

This function controls the selection of input/output control options for stdio: stdin = 0, stdout = 1 and stderr = 2.
- int **ReplaceStdio** (int stdio_fd, int new_fd)

Maps stdio to any file descriptor (fd).
- int **CurrentStdioFD** (int stdio_fd)

Returns the current file descriptor mapped to the stdio file descriptor.
- void **IrqStdio** ()

Open the system default serial port in interrupt mode using the system default baud rate, and assign this serial port to stdin, stdout and stderr.

FD Change Type

The notifications that a registered FD monitor can receive.

- enum **FDChangeType** { **eReadSet** , **eWriteSet** , **eErrorSet** , **eClosingNow** }
- The notifications that a registered FD monitor can receive.*

I/O Control Options

The legal options for use with **ioctl()** and in conjunction with **ioctlCmdFlags**

- #define **IOCTL_TX_CHANGE_CRLF** (1)

When set, transmitted char \n gets converted to \r\n
- #define **IOCTL_RX_CHANGE_CRLF** (2)

When set, received \r\n get turned into \n
- #define **IOCTL_RX_PROCESS_EDITS** (4)

When set, process backspace and do simple line editing.

- `#define IOCTL_RX_ECHO (8)`
When set, echo chars received to tx.
- `#define IOCTL_TX_NO_BLOCK (32)`
When set, stdout and stderr will drop output instead of blocking.
- `#define IOCTL_ALL_OPTIONS (15)`
When set, turns on all options.

I/O Control Command Flags

The command flags to use in conjunction with `ioctl()` and `ioctlOptions`

- `#define IOCTL_SET (0x4000)`
Set an option.
- `#define IOCTL_CLR (0x2000)`
Clear an option.

14.25.1 Detailed Description

```
#include< fdfuncs.h>
```

Allows you to set and get parameters for FD based connections and status of network interfaces, serial interfaces, and any custom file descriptors created by an application.

```
#include< iosys.h>
```

I/O system functions use file descriptors for communication and status of network interfaces, serial interfaces, and any custom file descriptors created by an application.

14.25.2 Typedef Documentation

14.25.2.1 FDCallBack

```
typedef void FDCallBack(int fd, FDChangeType change, void *pData)
```

```
#include <iosys.h>
```

Define the function signature for file descriptor notification callbacks.

Parameters

<i>fd</i>	The file descriptor number
<i>change</i>	The reason for the call back, <code>FDChangeType</code>
<i>*pData</i>	A void pointer passed into the register function. Can be used to reference objects or other data

See also

[FDChangeType](#), [RegisterFDCallBack](#)

14.25.3 Enumeration Type Documentation

14.25.3.1 FDChangeType

```
enum FDChangeType
```

```
#include <iosys.h>
```

The notifications that a registered FD monitor can receive.

Enumerator

<code>eReadSet</code>	When read data changes from not available to available.

Enumerator

eWriteSet	When an FD changes from not writable to writeable.
eErrorSet	When an FD goes from no error to having an error.
eClosingNow	when the underlying protocol closes the fd

14.25.4 Function Documentation

14.25.4.1 charavail()

```
int charavail ()
#include <iosys.h>
```

Checks to see if data is available for read on stdin. By default, stdin is the boot/debug serial port.

Return values

1	Data available
0	No data available

See also

[dataavail\(\)](#), [getchar\(\)](#), [writeavail\(\)](#), [read\(\)](#)

14.25.4.2 close()

```
int close (
    int fd)
#include <iosys.h>
```

Close the specified file descriptor and free the associated resources.

Parameters

<i>fd</i>	The file descriptor number

Returns

0 on success, or a resource specific error on failure.

See also

[peek\(\)](#)
[PeekWithTimeout\(\)](#)
[read\(\)](#)
[ReadWithTimeout\(\)](#)
[write\(\)](#)

14.25.4.3 CurrentStdioFD()

```
int CurrentStdioFD (
    int stdio_fd)
#include <iosys.h>
```

Returns the current file descriptor mapped to the stdio file descriptor.

Parameters

<i>stdio</i> ↔ <i>_fd</i>	The stdio file descriptor to check. Options are stdin (0), stdout (1), and stderr (2).

Returns

The current file descriptor mapped to `stdio_fd`

14.25.4.4 dataavail()

```
int dataavail (
    int fd)
#include <iosys.h>
```

Check the specified file descriptor to determine if data is available to be read.

Parameters

<i>fd</i>	The file descriptor number

Return values

1	Data available
0	No data available

See also

[charavail\(\)](#), [getchar\(\)](#), [peek\(\)](#), [read\(\)](#), [ReadWithTimeout\(\)](#)

14.25.4.5 FD_CLR()

```
void FD_CLR (
    int fd,
    fd_set * pfdset)
#include <iosys.h>
```

A `fd_set` (file descriptor set) holds a set of file descriptors (fds). This function clears or removes a specific file descriptor in an `fd_set`.

Parameters

<i>fd</i>	The file descriptor number
<i>*pfdset</i>	Pointer to the <code>fd_set</code> to modify

See also

[FD_ZERO\(\)](#), [FD_SET\(\)](#), [FD_ISSET\(\)](#), [select\(\)](#)

14.25.4.6 FD_CLRFROMSET()

```
void FD_CLRFROMSET (
    const fd_set * from,
    fd_set * to)
#include <iosys.h>
```

A `fd_set` (file descriptor set) holds a set of file descriptors (fds). This function remove the file descriptors in one set from another.

Parameters

<i>*from</i>	Pointer to the fd_set defining which file descriptors to remove.
<i>*to</i>	Pointer to the fd_set containing file descriptors to remove.

See also

[FD_ZERO\(\)](#), [FD_CLR\(\)](#), [FD_ISSET\(\)](#), [select\(\)](#)

14.25.4.7 FD_COPY()

```
void FD_COPY (
    const fd_set * from,
    fd_set * to)
#include <iosys.h>
```

A fd_set (file descriptor set) holds a set of file descriptors (fds). This function copies one file descriptor set to another.

Parameters

<i>*from</i>	Pointer to the file descriptor set being copied from.
<i>*to</i>	Pointer to the fd_set to copy to.

See also

[FD_ZERO\(\)](#), [FD_CLR\(\)](#), [FD_ISSET\(\)](#), [select\(\)](#)

14.25.4.8 FD_ISSET()

```
int FD_ISSET (
    int fd,
    fd_set * pfdset)
#include <iosys.h>
```

A fd_set (file descriptor set) holds a set of file descriptors (fds). This function checks whether or not a particular fd is set in the specified fd_set.

For example:

```
if ( FD_ISSET( fdListen, &readFds ) )
{
    // do processing for fdListen
}
```

Parameters

<i>fd</i>	The file descriptor number
<i>*pfdset</i>	A pointer to the fd_set to test

Return values

0	File descriptor is not set in the file descriptor set
>0	File descriptor is set in the file descriptor set

See also

[FD_ZERO\(\)](#), [FD_CLR\(\)](#), [FD_SET\(\)](#), [FD_OVERLAP\(\)](#), [select\(\)](#)

14.25.4.9 FD_OVERLAP()

```
int FD_OVERLAP (
    const fd_set * f1,
    const fd_set * f2)
```

```
#include <iosys.h>
```

A `fd_set` (file descriptor set) holds a set of file descriptors (fds). This function determines whether there exists a common file descriptor between the two sets.

Parameters

<i>*f1</i>	Pointer to the first file descriptor set.
<i>*f2</i>	Pointer to the second file descriptor set.

See also

[FD_ZERO\(\)](#), [FD_CLR\(\)](#), [FD_ISSET\(\)](#), [select\(\)](#)

14.25.4.10 FD_SET()

```
void FD_SET (
    int fd,
    fd_set * pfdset)
```

```
#include <iosys.h>
```

A `fd_set` (file descriptor set) holds a set of file descriptors (fds). This function sets or adds a specific file descriptor to an `fd_set`.

Parameters

<i>fd</i>	The file descriptor number
<i>*pfdset</i>	Pointer to the <code>fd_set</code> to modify

See also

[FD_ZERO\(\)](#), [FD_CLR\(\)](#), [FD_ISSET\(\)](#), [select\(\)](#)

14.25.4.11 FD_SETFROMSET()

```
void FD_SETFROMSET (
    const fd_set * from,
    fd_set * to)
```

```
#include <iosys.h>
```

A `fd_set` (file descriptor set) holds a set of file descriptors (fds). This function adds the file descriptors from one set to another.

Parameters

<i>*from</i>	Pointer to the file descriptor set being copied from.
<i>*to</i>	Pointer to the <code>fd_set</code> to add to.

See also

[FD_ZERO\(\)](#), [FD_CLR\(\)](#), [FD_ISSET\(\)](#), [select\(\)](#)

14.25.4.12 FD_ZERO()

```
void FD_ZERO (  
    fd_set * pfds)
```

```
#include <iosys.h>
```

Clear (set to 0) a `fd_set` (file descriptor set) so no file descriptors (fds) are selected.

Parameters

<i>*pfds</i>	Pointer to the file descriptor set

See also

[FD_CLR\(\)](#), [FD_SET\(\)](#), [FD_ISSET\(\)](#), [select\(\)](#)

14.25.4.13 getchar()

```
int getchar ()
#include <iosys.h>
```

Get a character from stdin. Will block if no character is available, from stdio library. The default stdin is the debug serial port.

Returns

Character value (int) from stdin.

See also

[charavail\(\)](#), [dataavail\(\)](#), [peek\(\)](#), [read\(\)](#), [ReadWithTimeout\(\)](#)

14.25.4.14 haserror()

```
int haserror (
    int fd)
#include <iosys.h>
```

Check if a file descriptor has an error.

Parameters

<i>fd</i>	File descriptor number

Return values

1	The file descriptor has an error
0	The file descriptor does not have an error

See also

[dataavail\(\)](#), [writeavail\(\)](#), [charavail\(\)](#)

14.25.4.15 ioctl()

```
int ioctl (
    int fd,
    int cmd)
#include <iosys.h>
```

This function controls the selection of input/output control options for stdio: stdin = 0, stdout = 1 and stderr = 2. The legal options are listed in [ioctlOptions](#).

Parameters

<i>fd</i>	The file descriptor number. The three options are stdin (0), stdout (1), and stderr (2).
<i>cmd</i>	The ioctl command options are ioctlCmdFlags and the bit of the associated options as defined by ioctlOptions .

Returns

The old option value.

```
ioctl(0,IOCTL_SET|IOCTL_ALL_OPTIONS); // Sets all options
ioctl(1,IOCTL_CLR|IOCTL_ALL_OPTIONS); // Clears all options
ioctl(1,IOCTL_SET|IOCTL_RX_CHANGE_CRLF|IOCTL_RX_PROCESS_EDITS); // Sets some options
ioctl(2,IOCTL_CLR|IOCTL_RX_CHANGE_CRLF|IOCTL_RX_PROCESS_EDITS); // Clears some options
```

See also

[ReplaceStdio\(\)](#)

14.25.4.16 peek()

```
int peek (
    int fd,
    char * c)
#include <iosys.h>
```

Peek at the data for the specified file descriptor (fd). Will block forever until at least one byte is available to be read. The byte returned is not removed from the fd buffer, it will be the first byte of data returned on any subsequent read operation.

Parameters

<i>fd</i>	File descriptor
<i>*c</i>	Pointer to the read destination character string

Returns

- 1 on success
- Negative value on error. Error value is determined by the type of fd, such as TCP [TCP Socket Status](#), or Serial [Serial Port Error Codes](#).

See also

[TCP Socket Status](#), [close\(\)](#), [read\(\)](#), [PeekWithTimeout\(\)](#), [ReadWithTimeout\(\)](#), [write\(\)](#)

14.25.4.17 read()

```
int read (
    int fd,
    char * buf,
    int nbytes)
#include <iosys.h>
```

Read data from a file descriptor (fd).

This function will block forever until at least one byte is available to be read from the input buffer of the specified file descriptor. To wait for a period of time if no data is immediately available in the buffer, please see [ReadWithTimeout\(\)](#) File descriptors include: TCP, UDP, Serial and Extra File Descriptors (EFD). EFDs can be used for custom purposes, such as reading/writing to buffers, or any peripheral custom to your application.

The function will return the number of bytes read, or a negative number if there is an error condition. The error value depends on the type of file descriptor, such as TCP or Serial:

- TCP Status Codes: [TCP Socket Status](#)
- Serial Status Codes: [Serial Port Error Codes](#)

Parameters

<i>fd</i>	The file descriptor number.
<i>*buf</i>	Pointer to the read data destination.
<i>nbytes</i>	Maximum number of bytes to read.

Returns

- Number of bytes read
- Negative value on error. Error value is determined by the type of fd, such as TCP [TCP Socket Status](#), or Serial [Serial Port Error Codes](#).

See also

[TCP Socket Status](#), [close\(\)](#), [peek\(\)](#), [PeekWithTimeout\(\)](#), [ReadWithTimeout\(\)](#), [ReadAllWithTimeout\(\)](#), [readall\(\)](#), [write\(\)](#)

14.25.4.18 readall()

```
int readall (  
    int fd,  
    char * buf,  
    int nbytes)  
#include <iosys.h>
```

Read the specified number of bytes from a file descriptor (fd). This function will block until either the requested number of bytes have been read, or a file descriptor error occurs. It can be used to read from stdio, TCP sockets, or Serial ports.

Parameters

<i>fd</i>	File descriptor
<i>*buf</i>	Pointer to the read destination buffer
<i>nbytes</i>	Number of bytes to read

Returns

- Number of bytes read.
- 0 on timeout.
- Negative value on error. Error value is determined by the type of fd, such as TCP [TCP Socket Status](#), or Serial [Serial Port Error Codes](#).

See also

[close\(\)](#), [peek\(\)](#), [read\(\)](#), [PeekWithTimeout\(\)](#), [ReadWithTimeout\(\)](#), [ReadAllWithTimeout\(\)](#), [write\(\)](#)

14.25.4.19 ReadAllWithTickTimeout()

```
int ReadAllWithTickTimeout (  
    int fd,  
    char * buf,  
    int nbytes,  
    TickTimeout & timeout)  
#include <iosys.h>
```

Same as [ReadWithTimeout\(\)](#), except the timeout value parameter is of type TickTimeout instead of an unsigned long. The TickTimeout object is more robust in terms of sequential timeout calls and rollover protection. This function can be used to read from stdio, TCP sockets, or Serial ports. This function will block until on of the following conditions occurs:

- The requested number of bytes have been read.
- The timeout expires. The number of bytes read are returned, and buf contains the data.
- An error occurs on the file descriptor, such as a TCP socket error.

Note

The return value must be checked to verify the number of bytes read and that a timeout or error did not occur.

Parameters

<i>fd</i>	The file descriptor number
<i>*buf</i>	A pointer to the read destination
<i>nbytes</i>	Number of bytes to read
<i>timeout</i>	A reference to the timeout value of type TickTimeout.

Returns

- Number of bytes read.
- 0 on timeout.
- Negative value on error. Error value is determined by the type of fd, such as TCP [TCP Socket Status](#), or Serial [Serial Port Error Codes](#).

See also

[close\(\)](#), [peek\(\)](#), [read\(\)](#), [PeekWithTimeout\(\)](#), [ReadWithTimeout\(\)](#), [ReadAllWithTimeout\(\)](#), [readall\(\)](#), [write\(\)](#)

14.25.4.20 ReadAllWithTimeout()

```
int ReadAllWithTimeout (
    int fd,
    char * buf,
    int nbytes,
    unsigned long timeout)
```

```
#include <iosys.h>
```

Attempt to read the specified number of bytes from a file descriptor, or return with the number of bytes read if the timeout value has expired.

This function can be used to read from stdio, TCP sockets, or Serial ports. This function will block until on of the following conditions occurs:

- The requested number of bytes have been read.
- The timeout expires. The number of bytes read are returned, and buf contains the data.
- An error occurs on the file descriptor, such as a TCP socket error.

Note

The return value must be checked to verify the number of bytes read and that a timeout or error did not occur.

Parameters

<i>fd</i>	The file descriptor number
<i>*buf</i>	A pointer to the read destination
<i>nbytes</i>	Number of bytes to read
<i>timeout</i>	The number of timer ticks to wait for data

Returns

- Number of bytes read.
- 0 on timeout.
- Negative value on error. Error value is determined by the type of fd, such as TCP [TCP Socket Status](#), or Serial [Serial Port Error Codes](#).

See also

[close\(\)](#), [peek\(\)](#), [read\(\)](#), [PeekWithTimeout\(\)](#), [ReadWithTimeout\(\)](#), [ReadAllWithTickTimeout\(\)](#), [ReadWithTickTimeout\(\)](#), [readall\(\)](#), [write\(\)](#)

14.25.4.21 ReadWithTickTimeout()

```
int ReadWithTickTimeout (
    int fd,
    char * buf,
    int nbytes,
    TickTimeout & timeout)
#include <iosys.h>
```

Same as [ReadWithTimeout\(\)](#), except the timeout value parameter is of type TickTimeout instead of an unsigned long. The TickTimeout object is more robust in terms of sequential timeout calls and rollover protection.

Note

This function operates like the read function in that it reads all available bytes up to the maximum and returns. The addition of a timeout does not cause the function to block until the specified maximum number of bytes are available. As with read, the application must use the return value of the ReadWithTimeout function to determine how many bytes were read, and call the function again if necessary.

When using with a TCP socket, you may want to use the [SockReadWithTimeout\(\)](#) function, which returns additional values related to network connections.

Parameters

<i>fd</i>	The file descriptor number.
<i>*buf</i>	A pointer to the read destination.
<i>nbytes</i>	Maximum number of bytes to read.
<i>timeout</i>	A reference to the timeout value of type TickTimeout.

Returns

- Number of bytes read.
- 0 on timeout.
- Negative value on error. Error value is determined by the type of fd, such as TCP [TCP Socket Status](#), or Serial [Serial Port Error Codes](#).

See also

[close\(\)](#), [peek\(\)](#), [read\(\)](#), [PeekWithTimeout\(\)](#), [ReadWithTimeout\(\)](#), [ReadAllWithTimeout\(\)](#), [readall\(\)](#), [SockReadWithTimeout\(\)](#), [write\(\)](#)

14.25.4.22 ReadWithTimeout()

```
int ReadWithTimeout (
    int fd,
    char * buf,
    int nbytes,
    unsigned long timeout)
#include <iosys.h>
```

Read data from a file descriptor(fd), or return if at least one byte is not read within the specified timeout. Can be used instead of the [read\(\)](#) function, which will block forever until at least one byte is read. File descriptors include such things as stdio, TCP sockets, TLS sockets, and Serial ports.

Note

This function operates like the read function in that it reads all available bytes up to the maximum and returns. The addition of a timeout does not cause the function to block until the specified maximum number of bytes are available. As with read, the application must use the return value of the ReadWithTimeout function to determine how many bytes were read, and call the function again if necessary.

When using with a TCP socket, you may want to use the [SockReadWithTimeout\(\)](#) function, which returns additional values related to network connections.

Parameters

<i>fd</i>	The file descriptor number.
<i>*buf</i>	A pointer to the read destination.
<i>nbytes</i>	Maximum number of bytes to read.
<i>timeout</i>	The number of timer ticks to wait for data.

Returns

- Number of bytes read.
- 0 on timeout.
- Negative value on error. Error value is determined by the type of fd, such as TCP [TCP Socket Status](#), or Serial [Serial Port Error Codes](#).

See also

[close\(\)](#), [peek\(\)](#), [read\(\)](#), [PeekWithTimeout\(\)](#), [ReadWithTickTimeout\(\)](#), [ReadAllWithTimeout\(\)](#), [readall\(\)](#), [SockReadWithTimeout\(\)](#), [write\(\)](#)

14.25.4.23 RegisterFDCallback()

```
void RegisterFDCallback (
    int fd,
    FDCallback * fp,
    void * pData)
```

```
#include <iosys.h>
```

Register a callback function to receive notification when an FD state changes. Register a NULL fp to remove the notification.

Parameters

<i>fd</i>	The file descriptor number
<i>*fp</i>	The function pointer to call back
<i>*pData</i>	A void * passed to the callback function, can be used for any purpose, such as object references

See also

[FDChangeType](#), [FDCCallback](#)

14.25.4.24 ReplaceStdio()

```
int ReplaceStdio (
    int stdio_fd,
    int new_fd)
```

```
#include <iosys.h>
```

Maps stdio to any file descriptor (fd).

If the file descriptor generates an error (such as a closed TCP connection), then stdio will be remapped to a negative fd (this will cause stdio to generate errors). When this function is used to remap a stdio channel that has errored, then the error will be cleared.

Parameters

<i>stdio↔ _fd</i>	The stdio file descriptor to map to. Options are stdin (0), stdout (1), and stderr (2).
<i>new_fd</i>	The file descriptor to replace stdio with. A value of 0 returns stdio to the default debug monitor based traps.

Return values

0	If stdio had not been previously mapped
>0	The value of the fd for the previous stdio override

See also

[ioctl\(\)](#)

14.25.4.25 select()

```
int select (
    int nfds,
    fd_set * readfds,
    fd_set * writefds,
    fd_set * errorfds,
    unsigned long timeout)
#include <iosys.h>
```

Wait for events to occur on one or more I/O resources associated with a set of file descriptors (fds), such as data available to be read, a resource is available to write data, or an error has occurred.

The most common use for this function is to monitor multiple serial and/or TCP file descriptors at one time to read incoming data. For example, an application could use a single RTOS task and a [select\(\)](#) to process incoming data on multiple TCP sockets at one time, instead of using multiple tasks or looping through each fd sequentially.

The file descriptors of interest are specified with file descriptor sets (fds) for read, write and error conditions. [select\(\)](#) will block until one of the fd conditions are true, or the specified timeout occurs (a value of 0 blocks forever). For example, if the readfds is used for multiple TCP sockets, [select\(\)](#) will return if data is available to read on one or more of them. The application would then use macros such as [FD_ISSET\(\)](#) to handle reading the data.

Note

A timeout value can be used to cause [select\(\)](#) to return periodically for any type of processing an application would like to do in the event no activity is occurring on the fd sets.

Parameters

<i>nfds</i>	The number of file descriptors to examine (This parameter is currently ignored)
<i>*readfds</i>	Pointer to the fd_set to select for read events, or null if no read events are to be monitored. It is modified on exit to reflect the read availability of the selected fds in the set.
<i>*writefds</i>	Pointer to the fd_set to select for write availability events, or null if no events are to be monitored. If non-null, it is modified on exit to reflect the write availability of the selected fds in the set.
<i>*errorfds</i>	Pointer to the fd_set to select for error events, or null if no error events are to be monitored. It is modified on exit to reflect the error state of the selected fds in the set.
<i>timeout</i>	The number of system time ticks to wait before timing out if no events occurred in the selected fd sets. Note: The TICKS_PER_SECOND multiplier should be used to specify the timeout in seconds. For example, TICKS_PER_SECOND * 5 to specify a timeout of 5 seconds.

Return values

0	Timeout occurred
>0	The number of fds in all of the non-null fd_sets

See also

[FD_ZERO\(\)](#), [FD_CLR\(\)](#), [FD_SET\(\)](#), [FD_ISSET\(\)](#)

14.25.4.26 SockPeekWithTimeout()

```
int SockPeekWithTimeout (
    int fd,
    char * buf,
    uint32_t maxtime)
```

```
#include <iosys.h>
```

This function peeks at data from a file descriptor (*fd*), with a specified timeout value (as opposed to the `peek` function which will block forever until at least one byte is available to be read). This function can be used to peek from `stdio`, TCP sockets, or Serial ports.

The byte returned is *not* removed from the *fd*. The byte returned will be the first byte of data returned on the first subsequent `read()`.

Parameters

<i>fd</i>	The file descriptor number
<i>*buf</i>	A pointer to the read destination
<i>maxtime</i>	The number of time ticks to wait for at least 1 byte of data

Returns

- 1 on success.
- 0 on timeout.
- Negative value on error. Error value is determined by the type of *fd*, such as TCP [TCP Socket Status](#), or Serial [Serial Port Error Codes](#).

See also

[TCP Socket Status](#), [close\(\)](#), [peek\(\)](#), [read\(\)](#), [ReadWithTimeout\(\)](#), [write\(\)](#)

14.25.4.27 write()

```
int write (
    int fd,
    const char * buf,
    int nbytes)
```

```
#include <iosys.h>
```

Write data to the stream associated with a file descriptor (*fd*). Can be used to write data to `stdio`, a TCP socket, or a Serial port.

Note

The function will block until at least 1 byte is written, but will not write all requested bytes if there is not enough space in the write buffer. The return value must always be checked to verify the number of bytes written.

Parameters

<i>fd</i>	File descriptor
<i>*buf</i>	Pointer to the buffer to write
<i>nbytes</i>	Maximum number of bytes to write

Returns

- The number of bytes written to write buffer.
- Negative value on error. Error value is determined by the type of *fd*, such as TCP [TCP Socket Status](#), or Serial [Serial Port Error Codes](#).

See also

[close\(\)](#), [peek\(\)](#), [read\(\)](#), [PeekWithTimeout\(\)](#), [ReadWithTimeout\(\)](#), [writestring\(\)](#), [writeall\(\)](#)

14.25.4.28 writeall()

```
int writeall (  
    int fd,  
    const char * buf,  
    int nbytes = 0)
```

```
#include <iosys.h>
```

Write the specified number of bytes to a file descriptor. Will block until all bytes are sent, or a file descriptor error occurs (such as a TCP socket error). Can be used to write data to stdio, a TCP socket, or a Serial port.

Note

The function will block until at least 1 byte is written, but will not write all requested bytes if there is not enough space in the write buffer. The return value must always be checked to verify the number of bytes written.

Parameters

<i>fd</i>	File descriptor
<i>*buf</i>	Pointer to the buffer to write
<i>nbytes</i>	Maximum number of bytes to write

Returns

- The number of bytes written to write buffer.
- Negative value on error. Error value is determined by the type of fd, such as TCP [TCP Socket Status](#), or Serial [Serial Port Error Codes](#).

See also

[close\(\)](#), [peek\(\)](#), [read\(\)](#), [PeekWithTimeout\(\)](#), [ReadWithTimeout\(\)](#), [write\(\)](#), [writestring\(\)](#)

14.25.4.29 writeavail()

```
int writeavail (  
    int fd)  
#include <iosys.h>
```

Check the specified file descriptor to determine data can be written.

Parameters

<i>fd</i>	File descriptor number

Return values

1	Data can be written
0	Data cannot be written, such as when an output buffer is full

See also

[dataavail\(\)](#), [charavail\(\)](#), [haserror\(\)](#)

14.25.4.30 writestring()

```
int writestring (
    int fd,
    const char * str)
```

```
#include <iosys.h>
```

Write a null terminated ascii string to the stream associated with a file descriptor (fd). Can be used to write data to stdio, a TCP socket, or a Serial port.

Note

The function will block until at least 1 byte is written, but will not write all requested bytes if there is not enough space in the write buffer. The return value must always be checked to verify the number of bytes written.

Parameters

<i>fd</i>	File descriptor
<i>*str</i>	Pointer to the null terminated string to write

Returns

- The number of bytes written to write buffer.
- Negative value on error. Error value is determined by the type of fd, such as TCP [TCP Socket Status](#), or Serial [Serial Port Error Codes](#).

See also

[close\(\)](#), [peek\(\)](#), [read\(\)](#), [PeekWithTimeout\(\)](#), [ReadWithTimeout\(\)](#), [write\(\)](#), [writeall\(\)](#)

14.25.4.31 ZeroWaitSelect()

```
int ZeroWaitSelect (
    int nfd,
    fd_set * readfds,
    fd_set * writefds,
    fd_set * errorfds)
```

```
#include <iosys.h>
```

Returns whether events have occurred on one or more I/O resources associated with a set of file descriptors (fds), and returns immediately.

Parameters

<i>nfd</i>	The number of file descriptors to examine. Note: This parameter is currently ignored.
<i>*readfds</i>	A pointer to the fd_set to select for read events. Note: This parameter can be NULL. It is modified on exit to reflect the read availability of the selected fds in the set.
<i>*writefds</i>	A pointer to the fd_set to select for write availability events. Note: This parameter can be NULL. It is modified on exit to reflect the write availability of the selected fds in the set.
<i>*errorfds</i>	A pointer to the fd_set to select for error events. Note: This parameter can be NULL. It is modified on exit to reflect the error state of the selected fds in the set.

Return values

<i>0</i>	If there is no valid fds
<i>>0</i>	The number of fds in all of the non null fd_sets

See also

[select\(\)](#), [FD_ZERO\(\)](#), [FD_CLR\(\)](#), [FD_SET\(\)](#), [FD_ISSET\(\)](#)

14.26 IP

Core IP layer definitions, structures, and functions for IPv4/IPv6 networking.

Topics

- [IPADDR4 Class](#)
- [IPADDR6 Class](#)

Classes

- struct [_IpHeaderIPv4](#)
IP Header Structure (IPv4)
- struct [IPPKT](#)
Internal IPv4 Header Representation.
- struct [UDPPKT](#)
Internal UDP Packet Header Structure.
- struct [PsudeoHeader](#)
UDP/TCP IPv4 Pseudo-header Structure (bits 0 through 95 of the pseudo-header)

Macros

- #define [ICMP_PING_DATA_SIZE](#) (32)
Default data payload size for ICMP echo request and reply packets (32 bytes)

Typedefs

- typedef struct [_IpHeaderIPv4](#) [IpHeaderIPv4](#)
IP Header Structure (IPv4)
- typedef void [FragmentProcessFunction](#)([PoolPtr](#) pp)
User-defined function pointer type for processing fragmented IP packets.
- typedef void [PromiscuousPacketFunc](#)([PoolPtr](#) pp)
User-defined function pointer type for processing promiscuous packets.

Functions

- void [IpProcessEthernetPacket](#) ([PoolPtr](#) poolPtr, uint16_t packetSizeInBytes)
Process an Ethernet packet for IP layer handling.
- uint16_t [GetSum](#) (uint16_t addr, uint16_t count)
Calculate Internet checksum for the specified data.
- uint16_t [GetSum20](#) (uint32_t addr)
Calculate Internet checksum for exactly 20 bytes of data.
- uint16_t [GetSumHdr](#) ([PsudeoHeader](#) &hdr, uint16_t addr, uint16_t count)
Calculate Internet checksum using a pseudo-header.
- BOOL [IsMyIp4](#) ([IPADDR4](#) ip, int ifc=-1)
Check if the specified IPv4 address belongs to this device.
- [IPADDR4](#) [GetSrcIp4](#) ([IPADDR4](#) dst)
Get the appropriate source IPv4 address for reaching a destination.
- [IPADDR4](#) [GetSrcIpwIf4](#) (int ifn, const [IPADDR4](#) dst)
Get the source IPv4 address for a specific interface and destination.
- BOOL [IsMyIp6](#) (const [IPADDR](#) &ip, int ifc=-1)
Check if the specified IPv6 address belongs to this device.
- BOOL [IsMyIp](#) (const [IPADDR](#) &ip, int ifc=-1)
Generic function to check if IP address belongs to this device (uses IPv6 when available)

- [IPADDR GetSrcIp6](#) (const [IPADDR](#) &dst)
Get the appropriate source IPv6 address for reaching a destination.
- [IPADDR GetSrcIp](#) (const [IPADDR](#) &dst)
Generic function to get source IP address for reaching a destination (IPv6 version).
- [IPADDR GetSrcIp6wIf](#) (int ifn, const [IPADDR](#) &dst)
Get the source IPv6 address for a specific interface and destination.
- [IPADDR GetSrcIpwIf](#) (int ifn, const [IPADDR](#) &dst)
Generic function to get source IP address for an interface (IPv6 version).
- void [SendICMPError](#) ([PoolPtr](#) pBadPacket, uint8_t type, uint8_t code)
Send an ICMP error message in response to a bad packet.
- void [FixHeaderAndSend](#) ([PoolPtr](#) p, [PIPPKT](#) plp)
Complete IP header fields and send packet on primary interface.
- void [FixHeaderAndSendViaInterface](#) ([PoolPtr](#) p, [PIPPKT](#) plp, int Interface)
Complete IP header fields and send packet via specified interface.
- void [InitializeStack](#) ()
Initialize the IP protocol stack.
- void [ShowIPBuffer](#) ([PoolPtr](#) rp)
Display the contents of a network buffer for diagnostic purposes.
- int [Ping4](#) ([IPADDR4](#) to, uint16_t id, uint16_t seq, uint16_t maxwaitticks)
Send a "ping" packet and wait the specified timeout for a response.
- int [Ping4ViaInterface](#) ([IPADDR4](#) to, uint16_t id, uint16_t seq, uint16_t wait, int interface)
Send a "ping" packet via a specific network interface and wait for a response.
- int [Ping6](#) (const [IPADDR](#) &to, uint16_t id, uint16_t seq, uint16_t maxwaitticks, int size=32)
Send an IPv6 "ping" packet and wait the specified timeout for a response.
- int [Ping6ViaInterface](#) (const [IPADDR](#) &to, uint16_t id, uint16_t seq, uint16_t wait, int interface, int size=32)
Send an IPv6 "ping" packet via a specific network interface.
- int [Ping](#) (const [IPADDR](#) &to, uint16_t id, uint16_t seq, uint16_t maxwaitticks, int size=32)
Generic ping function that uses IPv6 when IPv6 is enabled.
- int [PingViaInterface](#) (const [IPADDR](#) &to, uint16_t id, uint16_t seq, uint16_t wait, int interface, int size=32)
Generic ping via interface function that uses IPv6 when IPv6 is enabled.
- BOOL [IsLocal4](#) ([IPADDR4](#) ip, int ifc)
Check if an IPv4 address is on the local network segment.
- int [GetProperInterface4](#) ([IPADDR4](#) dst)
Get the appropriate interface number for reaching a destination.
- int [AddInterface](#) ([IPADDR4](#) addr, [IPADDR4](#) mask, [IPADDR4](#) gateway, int root_if=0)
Add a new network interface with the specified configuration.
- int [GetMultiHomeInterface](#) ([IPADDR4](#) ipa, int norgif)
Get the multi-home interface for a specific IP address.
- int [GetLocalIPInterface4](#) ([IPADDR4](#) ipa)
Get the local interface number for a specific IPv4 address.
- int [GetLocalIPInterface6](#) (const [IPADDR](#) &ipa)
Get the local interface number for a specific IPv6 address.
- int [GetLocalIPInterface](#) (const [IPADDR](#) &ipa)
Generic function to get local interface for an IP address (IPv6 version).
- uint16_t [GetEphemeralPort](#) ()
Generate a random ephemeral port number.

Variables

- [FragmentProcessFunction](#) * [pFragFunc](#)
Global function pointer for fragment processing callback.
- [PromiscuousPacketFunc](#) * [pPromiscuousPacketFunc](#)
Global function pointer for promiscuous packet processing callback.
- [TaskPacketFunc](#) * [pTaskPacketFunc](#)
Global function pointer for UDP task packet processing callback.

Global Configuration Variables

These global variables control various IP stack behaviors and can be modified by the application to customize network behavior.

- `uint8_t` **bTTL_Default**
Default Time To Live (hop count) for outgoing IP packets. Typical value is 64 or 128.
- `uint16_t` **wArpLifetime**
[ARP](#) cache entry lifetime in seconds. Entries older than this are refreshed. Typical value is 300-1200 seconds.
- `BOOL` **bQuietStart**
When `TRUE`, suppresses startup messages during network initialization. Set before calling [init\(\)](#).

Get Packet Functions

These inline functions provide convenient access to packet structures at various layers of the network stack. They handle the pointer arithmetic needed to locate specific headers within network buffer pool buffers.

WARNING:

If you use these functions on an uninitialized buffer you will get bogus values for the pointer as the header length field in the IP packet is not yet set up! Always ensure packets are properly initialized before using these accessor functions.

- [PIPPKT GetIpPkt](#) ([PoolPtr](#) p)
Get IP packet pointer from network buffer pool buffer.
- [PIPPKT GetIpPkt](#) ([PEFRAME](#) pFrame)
Get IP packet pointer from pointer to Ethernet frame.
- [PIPPKT GetIpPkt](#) ([PVLEFRAME](#) pFrame)
Get IP packet pointer from pointer to VLAN-tagged Ethernet frame.
- [PUDPPKT GetUdpPkt](#) ([PIPPKT](#) plp)
Get UDP packet pointer from IP packet pointer.

IP Version

Uncomment for debugging IP layer operations

- `#define IP_VERSION_MASK (0xF0)`
Mask for extracting IP version from version/header length byte.
- `#define IP_VERSION_IPv4 (0x40)`
IPv4 version identifier (value 4 in upper nibble)
- `#define IP_VERSION_IPv6 (0x60)`
IPv6 version identifier (value 6 in upper nibble)

Header Length

- `#define IP_HEADER_LENGTH_MASK (0x0F)`
Mask for extracting header length from version/header length byte.
- `#define IP_HEADER_LENGTH_NO_OPTIONS (0x05)`
Header length value when no IP options present (5 32-bit words = 20 bytes)

Flags and Fragment Offset

- **#define IP_FLAGS_MASK** (0xE000)
Mask for extracting IP flags (upper 3 bits of flags/fragment field)
- **#define IP_FRAGMENT_OFFSET_MASK** (0x1FFF)
Mask for extracting fragment offset (lower 13 bits of flags/fragment field)

IP Protocol Numbers

These are IANA-assigned protocol numbers that identify the next-level protocol encapsulated in the IP packet.

- **#define IP_PROTOCOL_ICMP** (0x01)
Internet Control Message Protocol.
- **#define IP_PROTOCOL_IGMP** (0x02)
Internet Group Management Protocol.
- **#define IP_PROTOCOL_INTERNET** (0x04)
IP-in-IP encapsulation (deprecated)
- **#define IP_PROTOCOL_TCP** (0x06)
Transmission Control Protocol.
- **#define IP_PROTOCOL_UDP** (0x11)
User Datagram Protocol (value 17 decimal)

ICMP Message Types and Codes

Internet Control Message Protocol (ICMP) types and codes used for network diagnostics and error reporting. See RFC 792 for details.

- **#define ICMP_TYPE_ECHO_REPLY** (0)
Echo Reply message (response to ping)
- **#define ICMP_CODE_ECHO_REPLY** (0)
Code for Echo Reply (always 0)
- **#define ICMP_TYPE_DEST_UNREACHABLE** (3)
Destination Unreachable error message.
- **#define ICMP_CODE_PORT_UNREACHABLE** (3)
Port Unreachable code within Destination Unreachable.
- **#define ICMP_TYPE_SOURCE_QUENCH** (4)
Source Quench message (deprecated, for flow control)
- **#define ICMP_CODE_SOURCE_QUENCH** (0)
Code for Source Quench (always 0)
- **#define ICMP_TYPE_REDIRECT_MESSAGE** (5)
Redirect message (routing optimization)
- **#define ICMP_TYPE_ECHO_REQUEST** (8)
Echo Request message (ping)
- **#define ICMP_CODE_ECHO_REQUEST** (0)
Code for Echo Request (always 0)

Well-Known Port Numbers

IANA-assigned well-known port numbers for common network services. These ports are reserved for specific protocols and services.

- **#define IANA_ECHO_PORT** (7)
Echo protocol port.
- **#define IANA_DISCARD_PORT** (9)
Discard protocol port (null service)

- `#define IANA_SSH_PORT (22)`
Secure Shell (SSH) protocol port.
- `#define IANA_TELNET_PORT (23)`
Telnet protocol port.
- `#define IANA_NETBIOS_NAME_SERVICE_PORT (137)`
NetBIOS Name Service port.

14.26.1 Detailed Description

Core IP layer definitions, structures, and functions for IPv4/IPv6 networking.

```
#include <ip.h>
```

This module provides comprehensive support for IP networking including:

- Packet structure definitions for IP, UDP, and ICMP
- Address resolution and routing functions
- Ping/ICMP echo request functionality
- Packet checksum calculation
- Multi-homing and multiple interface support
- Promiscuous mode and packet filtering callbacks
- Fragment processing support

14.26.2 Typedef Documentation

14.26.2.1 FragmentProcessFunction

```
typedef void FragmentProcessFunction(PoolPtr pp)
#include <ip.h>
```

User-defined function pointer type for processing fragmented IP packets.

This callback function is called only for IP packets that have the fragmentation flag set in the IP header. Applications can implement this function to handle packet fragmentation and reassembly according to their specific requirements. Typical implementations would:

- Store fragments until all pieces arrive
- Reassemble complete datagrams
- Handle fragment timeout and cleanup
- Free the buffer when done processing

Parameters

<i>pp</i>	Packet pointer from network buffer pool containing the fragmented packet

Note

The callback function is responsible for properly handling and freeing the buffer.

This function is only called for packets with fragmentation flags set.

Fragment reassembly is not provided by default - must be implemented by application.

See also

[pFragFunc](#)

14.26.2.2 IpHeaderIPv4

```
typedef struct _IpHeaderIPv4 IpHeaderIPv4
```

```
#include <ip.h>
```

IP Header Structure (IPv4)

Standard IPv4 packet header as defined in IETF RFC 791 (September 1981) and MIL-STD-1777. This structure represents the 20-byte minimum IP header. IP options, if present, follow this structure and are indicated by a header length value greater than 5.

All multi-byte fields are in network byte order (big-endian) and use the beuint types for automatic endianness handling.

Note

The structure is packed to ensure proper alignment with network packet data.

Standard header is 20 bytes when no options are present (header length = 5).

Maximum header size is 60 bytes (header length = 15) when options are included.

See also

[IPPKT](#) for the internal representation used within NetBurner stack

14.26.2.3 PromisciousPacketFunc

```
typedef void PromisciousPacketFunc(PoolPtr pp)
```

```
#include <ip.h>
```

User-defined function pointer type for processing promiscuous packets.

This callback function is called for packets that fail the "IP address == my address" test. This enables applications to receive and process packets not specifically addressed to this device, useful for:

- Network monitoring and packet capture
- Bridge or router implementations
- Protocol analyzers
- IDS/IPS systems
- Custom routing logic

Parameters

<i>pp</i>	Packet pointer from network buffer pool containing the promiscuous packet

Warning

This function **MUST** free the buffers it receives using [FreeBuffer\(\)](#) or equivalent. Failure to free buffers will result in buffer pool exhaustion and network failure.

Note

This callback is only invoked for packets not addressed to this device's IP address.

Ethernet interface must be set to promiscuous mode to receive these packets.

See also

[pPromisciousPacketFunc](#)

14.26.3 Function Documentation

14.26.3.1 AddInterface()

```
int AddInterface (
    IPADDR4 addr,
    IPADDR4 mask,
    IPADDR4 gateway,
    int root_if) [inline]
#include <ip.h>
```

Add a new network interface with the specified configuration.

Dynamically adds a new network interface to the system with the specified IP address, subnet mask, gateway, and root interface configuration. This is used in multi-homed systems to configure additional network interfaces at runtime.

Typical uses:

- Adding secondary IP addresses to an interface (IP aliasing)
- Configuring Auto-IP link-local addresses
- Setting up virtual interfaces
- Dynamic network reconfiguration

Parameters

<i>addr</i>	IP address for the new interface
<i>mask</i>	Subnet mask for the new interface
<i>gateway</i>	Gateway IP address for the new interface (0.0.0.0 for none)
<i>root_if</i>	Root interface number this interface is associated with (physical interface)

Returns

Interface number of the newly added interface (≥ 0), or negative value on error

Note

Only available when MULTIHOME or AUTOIP is defined.

Used for dynamic interface configuration in multi-homed systems.

The root_if parameter specifies which physical interface this logical interface uses.

```
// Add a secondary IP address to interface 0
IPADDR4 secondary = AsciiToIp4("192.168.1.100");
IPADDR4 mask = AsciiToIp4("255.255.255.0");
IPADDR4 gw = AsciiToIp4("192.168.1.1");
int new_if = AddInterface(secondary, mask, gw, 0);
if (new_if >= 0) {
    iprintf("Added interface %d\r\n", new_if);
}
```

Add a new network interface with the specified configuration.

Parameters

<i>addr</i>	IP address of new interface
<i>mask</i>	IP mask
<i>gateway</i>	IP gateway
<i>root_if</i>	Optional root physical interface number. If not specified the default is 0, for Ethernt 0.

Returns

Interface number of newly created interface

14.26.3.2 FixHeaderAndSend()

```
void FixHeaderAndSend (
    PoolPtr p,
    PIPPKT pIp)
```

```
#include <ip.h>
```

Complete IP header fields and send packet on primary interface.

Fills in remaining IP header fields (checksum, fragmentation info, etc.) and transmits the packet via the primary network interface. This function handles the final steps of IP packet transmission including:

- Calculating and setting the header checksum
- Setting the Time To Live (TTL) field if not already set
- Handling fragmentation if necessary
- Forwarding to the data link layer

Parameters

<i>p</i>	Network buffer pool pointer containing the packet
<i>p</i> ↔ <i>Ip</i>	Pointer to IP header within the packet

Note

The IP header should be mostly complete before calling this function.

Source and destination addresses must be set before calling.

This function calculates and fills in header checksum and other final fields.

See also

[FixHeaderAndSendViaInterface\(\)](#)

14.26.3.3 FixHeaderAndSendViaInterface()

```
void FixHeaderAndSendViaInterface (
    PoolPtr p,
    PIPPKT pIp,
    int Interface)
```

```
#include <ip.h>
```

Complete IP header fields and send packet via specified interface.

Similar to [FixHeaderAndSend\(\)](#), but allows specification of which network interface to use for transmission. This is useful in multi-homed systems where you need explicit control over the egress interface.

Parameters

<i>p</i>	Network buffer pool pointer containing the packet
<i>pIp</i>	Pointer to IP header within the packet
<i>Interface</i>	Interface number to use for transmission (0-based)

Note

Allows control over which interface transmits the packet.

Useful for routing and multi-homed configurations.

Interface number must be valid or packet will be dropped.

See also

[FixHeaderAndSend\(\)](#)

14.26.3.4 GetEphemeralPort()

```
uint16_t GetEphemeralPort ()
#include <ip.h>
```

Generate a random ephemeral port number.

Returns a random port number suitable for use as a local port in client connections. The port number is selected from the ephemeral (dynamic) port range and avoids well-known ports (0-1023) and registered ports (1024-49151). Ephemeral ports are temporary ports used by the client side of a connection. Each call returns a different random port number to avoid conflicts with other active connections.

Returns

Random ephemeral port number (typically in range 49152-65535 per RFC 6335)

Note

Used internally for automatic local port assignment in client connections.

Avoids conflicts with well-known and registered port numbers.

Each call returns a different random port number.

The random selection reduces the likelihood of port exhaustion and security issues.

```
// Get an ephemeral port for a client connection
uint16_t local_port = GetEphemeralPort();
int fd = connect(server_ip, server_port, local_port, timeout);
```

14.26.3.5 GetIpPkt() [1/3]

```
PIPPKT GetIpPkt (
    PEFRAME pFrame) [inline]
#include <ip.h>
```

Get IP packet pointer from pointer to Ethernet frame.

Returns a pointer to the IP header located immediately after the Ethernet header.

Parameters

<i>pFrame</i>	Pointer to Ethernet frame structure

Returns

Pointer to IP header within the frame

14.26.3.6 GetIpPkt() [2/3]

```
PIPPKT GetIpPkt (
    PoolPtr p) [inline]
#include <ip.h>
```

Get IP packet pointer from network buffer pool buffer.

Calculates the location of the IP header within a network buffer by adding the Ethernet header size (14 bytes) to the buffer's data pointer.

Parameters

<i>p</i>	Network buffer pool pointer

Returns

Pointer to IP header, or NULL if p is NULL

14.26.3.7 GetIpPkt() [3/3]

```
PIPKT GetIpPkt (
    PVLEFRAME pFrame) [inline]
#include <ip.h>
Get IP packet pointer from pointer to VLAN-tagged Ethernet frame.
Returns a pointer to the IP header located after the VLAN Ethernet header.
```

Parameters

<i>pFrame</i>	Pointer to VLAN Ethernet frame structure

Returns

Pointer to IP header within the frame

14.26.3.8 GetLocalIPInterface()

```
int GetLocalIPInterface (
    const IPADDR & ipa) [inline]
#include <ip.h>
Generic function to get local interface for an IP address (IPv6 version).
This inline function provides a generic interface that automatically uses the IPv6 version when IPv6 support is
compiled in.
```

Parameters

<i>ipa</i>	IP address to look up

Returns

Interface number associated with the address

See also

[GetLocalIPInterface6\(\)](#), [GetLocalIPInterface4\(\)](#)

14.26.3.9 GetLocalIPInterface4()

```
int GetLocalIPInterface4 (
    IPADDR4 ipa)
#include <ip.h>
Get the local interface number for a specific IPv4 address.
Determines which local interface is associated with the specified IPv4 address. Searches all interfaces (physical
and virtual) to find the one with the matching address.
```

Parameters

<i>ipa</i>	IPv4 address to look up

Returns

Interface number associated with the address, or -1 if not found

Note

Only available when MULTIHOME or AUTOIP is defined.
Returns -1 if the address is not assigned to any interface.

14.26.3.10 GetLocalIPInterface6()

```
int GetLocalIPInterface6 (  
    const IPADDR & ipa)
```

```
#include <ip.h>
```

Get the local interface number for a specific IPv6 address.

Determines which local interface is associated with the specified IPv6 address. Checks both link-local and global IPv6 addresses on all interfaces.

Parameters

<i>ipa</i>	IPv6 address to look up

Returns

Interface number associated with the address, or -1 if not found

Note

Only available when MULTIHOMER or AUTOIP is defined and IPv6 is enabled.

Important for determining interface for link-local addresses.

14.26.3.11 GetMultiHomeInterface()

```
int GetMultiHomeInterface (  
    IPADDR4 ipa,  
    int norgif)
```

```
#include <ip.h>
```

Get the multi-home interface for a specific IP address.

Finds the appropriate interface number for the specified IP address in a multi-homed system configuration. This searches through all configured interfaces (including virtual interfaces) to find a match.

Parameters

<i>ipa</i>	IP address to look up
<i>norgif</i>	Original interface number (for fallback if address not found)

Returns

Interface number for the specified IP address, or norgif if not found

Note

Only available when MULTIHOMER or AUTOIP is defined.

Returns norgif if the address is not assigned to any interface.

14.26.3.12 GetProperInterface4()

```
int GetProperInterface4 (  
    IPADDR4 dst)
```

```
#include <ip.h>
```

Get the appropriate interface number for reaching a destination.

Performs routing table lookup to determine which interface should be used to reach the specified IPv4 destination address. This considers:

- Direct network connections (local subnets)
- Default gateway routes
- Static routes if configured
- Interface priorities

Parameters

<i>dst</i>	Destination IPv4 address

Returns

Interface number that should be used to reach the destination

Note

Returns the interface number based on routing table configuration.

Used internally for automatic interface selection.

Returns primary interface if no specific route found.

```
IPADDR4 remote = AsciiToIp4("8.8.8.8");
int ifnum = GetProperInterface4(remote);
iprintf("Use interface %d to reach %s\r\n", ifnum, "8.8.8.8");
```

14.26.3.13 GetSrcIp()

```
IPADDR GetSrcIp (
    const IPADDR & dst) [inline]
#include <ip.h>
```

Generic function to get source IP address for reaching a destination (IPv6 version).

This inline function provides a generic interface that automatically uses the IPv6 version when IPv6 support is compiled in.

Parameters

<i>dst</i>	Destination IP address

Returns

Local IP address that should be used as source for packets to dst

See also

[GetSrcIp6\(\)](#), [GetSrcIp4\(\)](#)

14.26.3.14 GetSrcIp4()

```
IPADDR4 GetSrcIp4 (
    IPADDR4 dst)
#include <ip.h>
```

Get the appropriate source IPv4 address for reaching a destination.

Determines which local IPv4 address should be used as the source address when sending packets to the specified destination. This performs routing table lookup and interface selection based on the destination address.

Parameters

<i>dst</i>	Destination IPv4 address

Returns

Local IPv4 address that should be used as source for packets to dst

Note

Returns the IP address of the interface that would be used to reach *dst*.
Considers routing table and interface configuration.
Essential for multi-homed systems to select correct source address.

See also

[GetSrcIp\(\)](#), [GetSrcIpwlf4\(\)](#), [GetSrcIp6\(\)](#)

14.26.3.15 GetSrcIp6()

```
IPADDR GetSrcIp6 (  
    const IPADDR & dst)
```

```
#include <ip.h>
```

Get the appropriate source IPv6 address for reaching a destination.

Determines which local IPv6 address should be used as the source address when sending packets to the specified destination. This performs routing table lookup and interface selection.

Parameters

<i>dst</i>	Destination IPv6 address

Returns

Local IPv6 address that should be used as source for packets to *dst*

Note

Selects appropriate address scope (link-local vs global) based on destination.

See also

[GetSrcIp\(\)](#), [GetSrcIp4\(\)](#), [GetSrcIp6wlf\(\)](#)

14.26.3.16 GetSrcIp6wlf()

```
IPADDR GetSrcIp6wlf (  
    int ifn,  
    const IPADDR & dst)
```

```
#include <ip.h>
```

Get the source IPv6 address for a specific interface and destination.

Returns the IPv6 address that should be used as the source address when sending packets to the specified destination via the specified interface.

Parameters

<i>ifn</i>	Interface number to use
<i>dst</i>	Destination IPv6 address

Returns

Local IPv6 address of the specified interface

Note

Forces use of specified interface, bypassing routing table.

See also

[GetSrcIp6\(\)](#), [GetSrcIpwlf\(\)](#), [GetSrcIpwlf4\(\)](#)

14.26.3.17 GetSrcIpwIf()

```
IPADDR GetSrcIpwIf (  
    int ifn,  
    const IPADDR & dst) [inline]  
#include <ip.h>
```

Generic function to get source IP address for an interface (IPv6 version).

This inline function provides a generic interface that automatically uses the IPv6 version when IPv6 support is compiled in.

Parameters

<i>ifn</i>	Interface number to use
<i>dst</i>	Destination IP address

Returns

Local IP address of the specified interface

See also

[GetSrcIp6wIf\(\)](#), [GetSrcIpwIf4\(\)](#)

14.26.3.18 GetSrcIpwIf4()

```
IPADDR4 GetSrcIpwIf4 (  
    int ifn,  
    const IPADDR4 dst)  
#include <ip.h>
```

Get the source IPv4 address for a specific interface and destination.

Returns the IPv4 address that should be used as the source address when sending packets to the specified destination via the specified interface. This bypasses routing table lookup and uses the specified interface directly.

Parameters

<i>ifn</i>	Interface number to use
<i>dst</i>	Destination IPv4 address

Returns

Local IPv4 address of the specified interface

Note

Forces use of the specified interface regardless of routing table.

Returns the IP address assigned to the specified interface.

The *dst* parameter may be used for interface-specific address selection.

See also

[GetSrcIp4\(\)](#), [GetSrcIpwIf\(\)](#), [GetSrcIp6wIf\(\)](#)

14.26.3.19 GetSum()

```
uint16_t GetSum (  
    uint16_t addr,  
    uint16_t count)
```

```
#include <ip.h>
```

Calculate Internet checksum for the specified data.

Computes the standard Internet checksum (RFC 1071) over the specified data. This is the standard checksum algorithm used in IP, TCP, UDP, and ICMP headers.

The algorithm sums all 16-bit words using one's complement arithmetic, then returns the one's complement of the sum.

Parameters

<i>addr</i>	Pointer to data to checksum (must be 16-bit aligned)
<i>count</i>	Number of bytes to include in checksum (should be even)

Returns

16-bit Internet checksum value

Note

Data should be 16-bit aligned for optimal performance.

If count is odd, the last byte is padded with zero.

This is the general-purpose checksum function for variable-length data.

See also

[GetSum20\(\)](#), [GetSumHdr\(\)](#)

14.26.3.20 GetSum20()

```
uint16_t GetSum20 (  
    uint32_t addr)
```

```
#include <ip.h>
```

Calculate Internet checksum for exactly 20 bytes of data.

Optimized checksum calculation for exactly 20 bytes (5 32-bit words), which is the standard size of an IP header without options. This function is significantly faster than [GetSum\(\)](#) for this common case.

Parameters

<i>addr</i>	Pointer to 20 bytes of data to checksum (must be 32-bit aligned)

Returns

16-bit Internet checksum value

Note

This is an optimized version for the common case of IP header checksums.

Data must be 32-bit aligned for correct operation.

Use [GetSum\(\)](#) if you need to checksum a different amount of data.

See also

[GetSum\(\)](#), [GetSumHdr\(\)](#)

14.26.3.21 GetSumHdr()

```
uint16_t GetSumHdr (
    PsudeoHeader & hdr,
    puint16_t addr,
    uint16_t count)
```

```
#include <ip.h>
```

Calculate Internet checksum using a pseudo-header.

Computes the Internet checksum over a pseudo-header followed by data. This is used for TCP and UDP checksum calculation, which includes a pseudo-header containing source/destination IP addresses and protocol information. The pseudo-header protects against misrouted packets but is not actually transmitted on the wire.

Parameters

<i>hdr</i>	Reference to pseudo-header structure (UDP/TCP pseudo-header)
<i>addr</i>	Pointer to data to checksum following the pseudo-header
<i>count</i>	Number of bytes in the data portion

Returns

16-bit Internet checksum value including pseudo-header and data

Note

The pseudo-header is not transmitted, but is included in checksum calculation.

Used for TCP and UDP checksum computation per RFC specifications.

Provides protection against packets delivered to wrong IP address or protocol.

See also

[GetSum\(\)](#), [GetSum20\(\)](#), [PsudeoHeader](#)

14.26.3.22 GetUdpPkt()

```
PUDPKT GetUdpPkt (
    PIPPKT pIp) [inline]
```

```
#include <ip.h>
```

Get UDP packet pointer from IP packet pointer.

Calculates the location of the UDP header by skipping past the IP header. This function accounts for variable-length IP headers (with or without options) by examining the header length field.

Parameters

$p \leftrightarrow$ <i>Ip</i>	Pointer to IP header

Returns

Pointer to UDP header, or NULL if plp is NULL

Note

If the IP header length is exactly 5 (no options), uses fast path.

Otherwise calculates offset based on header length field.

14.26.3.23 InitializeStack()

```
void InitializeStack ()
```

```
#include <ip.h>
```

Initialize the IP protocol stack.

Initializes the IP layer and underlying network infrastructure. This function should be called once during system startup, before the application calls [init\(\)](#) or configures network interfaces.

This function:

- Initializes the IP layer data structures
- Sets up the routing table
- Prepares the network interface framework
- Allocates necessary system resources

Note

Should be called once and only once before UserMain resets its priority.

The default settings use the configuration record settings.

This provides the network infrastructure for additional network interfaces.

In most applications, this is called automatically by [init\(\)](#).

14.26.3.24 IpProcessEthernetPacket()

```
void IpProcessEthernetPacket (
```

```
    PoolPtr poolPtr,
```

```
    uint16_t packetSizeInBytes)
```

```
#include <ip.h>
```

Process an Ethernet packet for IP layer handling.

This internal function is called by the Ethernet layer to insert received packets into the IP processing pipeline. It handles the transition from Ethernet frame processing to IP packet processing.

Parameters

<i>poolPtr</i>	Pointer to network buffer pool buffer containing the packet
<i>packetSizeInBytes</i>	Total size of the packet in bytes

Note

This is an internal function typically called by the Ethernet driver.

Applications should not normally call this function directly.

14.26.3.25 IsLocal4()

```
BOOL IsLocal4 (
```

```
    IPADDR4 ip,
```

```
    int ifc)
```

```
#include <ip.h>
```

Check if an IPv4 address is on the local network segment.

Determines if the specified IPv4 address is on the same network segment as the specified interface by comparing network portions of the addresses using the interface's subnet mask. This is useful for:

- Routing decisions (local vs gateway)
- [ARP](#) vs routing determination
- Network topology validation

Parameters

<i>ip</i>	IPv4 address to check
<i>ifc</i>	Interface number to check against

Returns

TRUE if the address is on the local network segment, FALSE otherwise

Note

Uses the interface's subnet mask to determine network boundaries.

Returns FALSE if the interface number is invalid.

```
IPADDR4 target = AsciiToIp4("192.168.1.100");
if (IsLocal4(target, 0)) {
    iprintf("Address is on local network - use ARP\r\n");
} else {
    iprintf("Address is remote - use gateway\r\n");
}
```

14.26.3.26 IsMyIp4()

```
BOOL IsMyIp4 (
    IPADDR4 ip,
    int ifc = -1)
```

```
#include <ip.h>
```

Check if the specified IPv4 address belongs to this device.

Determines whether the given IPv4 address is assigned to any interface on this device. Optionally checks only a specific interface.

Parameters

<i>ip</i>	IPv4 address to check
<i>ifc</i>	Interface number to check (default: -1 for all interfaces)

Returns

TRUE if the address belongs to this device, FALSE otherwise

Note

When ifc is -1, all interfaces are checked.

When ifc is specified, only that interface is checked.

Useful for determining if a packet is addressed to this device.

See also

[IsMyIp\(\)](#), [IsMyIp6\(\)](#)

14.26.3.27 IsMyIp6()

```
BOOL IsMyIp6 (
    const IPADDR & ip,
    int ifc = -1)
```

```
#include <ip.h>
```

Check if the specified IPv6 address belongs to this device.

Determines whether the given IPv6 address is assigned to any interface on this device. Optionally checks only a specific interface.

Parameters

<i>ip</i>	IPv6 address to check
<i>ifc</i>	Interface number to check (default: -1 for all interfaces)

Returns

TRUE if the address belongs to this device, FALSE otherwise

Note

When *ifc* is -1, all interfaces are checked.

Checks both link-local and global IPv6 addresses.

See also

[IsMyIp\(\)](#), [IsMyIp4\(\)](#)

14.26.3.28 Ping()

```
int Ping (
    const IPADDR & to,
    uint16_t id,
    uint16_t seq,
    uint16_t maxwaitticks,
    int size = 32) [inline]
```

```
#include <ip.h>
```

Generic ping function that uses IPv6 when IPv6 is enabled.

This inline function provides a generic ping interface that automatically uses the appropriate IPv6 ping function when IPv6 support is compiled in.

Parameters

<i>to</i>	Destination IP address to ping
<i>id</i>	ICMP identifier field
<i>seq</i>	ICMP sequence number field
<i>maxwaitticks</i>	Maximum wait time in system time ticks
<i>size</i>	Size of ping data payload in bytes (default: 32)

Returns

Same as [Ping6\(\)](#)

See also

[Ping6\(\)](#), [Ping4\(\)](#)

14.26.3.29 Ping4()

```
int Ping4 (
    IPADDR4 to,
    uint16_t id,
    uint16_t seq,
    uint16_t maxwaitticks)
```

```
#include <ip.h>
```

Send a "ping" packet and wait the specified timeout for a response.

Sends an ICMP echo request packet to the specified destination address and waits for an echo reply. The ping uses the primary network interface (usually Ethernet). For interface selection, use [Ping4ViaInterface\(\)](#).

The ping packet contains 32 bytes of ASCII data (starting with 'a' (0x61) through 0xFF) and expects an echo reply with the same data. This follows the standard ICMP echo request/reply protocol defined in RFC 792.

Parameters

<i>to</i>	Destination IPv4 address to ping
<i>id</i>	ICMP identifier field (typically process ID or sequence)
<i>seq</i>	ICMP sequence number field (typically increments per ping)
<i>maxwaitticks</i>	Maximum wait time in system time ticks. Use TICKS_PER_SECOND macro for conversion (e.g., TICKS_PER_SECOND * 5 for 5 seconds)

Returns

>=0 Number of ticks elapsed waiting for the response (successful ping) - use this for RTT -1 Timeout occurred
 - no response received within maxwaitticks -2 Error other than timeout occurred (e.g., network unreachable, invalid parameters)

```
// Ping Google DNS with a 3 second timeout
IPADDR4 google_dns = AsciiToIp4("8.8.8.8");
int result = Ping4(google_dns, 0x1234, 1, TICKS_PER_SECOND * 3);
if (result >= 0) {
    iprintf("Ping successful, RTT: %d ticks (%d ms)\r\n",
        result, result * 1000 / TICKS_PER_SECOND);
} else if (result == -1) {
    iprintf("Ping timeout - host unreachable or not responding\r\n");
} else {
    iprintf("Ping error\r\n");
}
```

See also

[Ping4ViaInterface\(\)](#), [Ping\(\)](#), [PingViaInterface\(\)](#)

14.26.3.30 Ping4ViaInterface()

```
int Ping4ViaInterface (
    IPADDR4 to,
    uint16_t id,
    uint16_t seq,
    uint16_t wait,
    int interface)
```

```
#include <ip.h>
```

Send a "ping" packet via a specific network interface and wait for a response.

Similar to [Ping4\(\)](#), but allows selection of a specific network interface for sending the ICMP echo request. This is useful in multi-homed systems where you need to control which interface is used for the ping, such as:

- Testing specific network paths
- Verifying interface connectivity
- Routing diagnostics
- Multi-WAN configurations

Parameters

<i>to</i>	Destination IPv4 address to ping
<i>id</i>	ICMP identifier field (typically process ID)
<i>seq</i>	ICMP sequence number field
<i>wait</i>	Maximum wait time in system time ticks
<i>interface</i>	Network interface number to use for sending the ping (0-based)

Returns

>=0 Number of ticks elapsed waiting for the response (successful ping) -1 Timeout occurred - no response received within wait time -2 Error other than timeout occurred (e.g., invalid interface number)

```
// Ping via interface 1 with a 2 second timeout
IPADDR4 target = AsciiToIp4("192.168.1.1");
int result = Ping4ViaInterface(target, 0x5678, 1, TICKS_PER_SECOND * 2, 1);
```

See also

[Ping4\(\)](#), [Ping\(\)](#), [PingViaInterface\(\)](#)

14.26.3.31 Ping6()

```
int Ping6 (
    const IPADDR & to,
    uint16_t id,
    uint16_t seq,
    uint16_t maxwaitticks,
    int size = 32)
```

```
#include <ip.h>
```

Send an IPv6 "ping" packet and wait the specified timeout for a response.

Sends an ICMPv6 echo request packet to the specified IPv6 destination address and waits for an echo reply. The ping uses the primary network interface. This follows the ICMPv6 echo request/reply protocol defined in RFC 4443.

Parameters

<i>to</i>	Destination IPv6 address to ping
<i>id</i>	ICMP identifier field (typically process ID)
<i>seq</i>	ICMP sequence number field (typically increments per ping)
<i>maxwaitticks</i>	Maximum wait time in system time ticks
<i>size</i>	Size of ping data payload in bytes (default: 32)

Returns

>=0 Number of ticks elapsed waiting for the response (successful ping) -1 Timeout occurred - no response received within maxwaitticks -2 Error other than timeout occurred

Note

IPv6 ping packets may traverse different paths than IPv4 pings to the same host.

Link-local addresses require the appropriate zone ID/interface specification.

See also

[Ping6ViaInterface\(\)](#), [Ping4\(\)](#), [Ping\(\)](#)

14.26.3.32 Ping6ViaInterface()

```
int Ping6ViaInterface (
    const IPADDR & to,
    uint16_t id,
    uint16_t seq,
    uint16_t wait,
    int interface,
    int size = 32)
```

```
#include <ip.h>
```

Send an IPv6 "ping" packet via a specific network interface.

Similar to [Ping6\(\)](#), but allows selection of a specific network interface for sending the ICMPv6 echo request. This is particularly important for IPv6 where link-local addresses are interface-specific.

Parameters

<i>to</i>	Destination IPv6 address to ping
<i>id</i>	ICMP identifier field (typically process ID)
<i>seq</i>	ICMP sequence number field
<i>wait</i>	Maximum wait time in system time ticks
<i>interface</i>	Network interface number to use for sending the ping
<i>size</i>	Size of ping data payload in bytes (default: 32)

Returns

>=0 Number of ticks elapsed waiting for the response (successful ping) -1 Timeout occurred - no response received within wait time -2 Error other than timeout occurred

Note

Required for pinging link-local IPv6 addresses.
Interface selection affects which source address is used.

See also

[Ping6\(\)](#), [Ping4ViaInterface\(\)](#), [PingViaInterface\(\)](#)

14.26.3.33 PingViaInterface()

```
int PingViaInterface (
    const IPADDR & to,
    uint16_t id,
    uint16_t seq,
    uint16_t wait,
    int interface,
    int size = 32) [inline]
```

```
#include <ip.h>
```

Generic ping via interface function that uses IPv6 when IPv6 is enabled.

This inline function provides a generic ping via interface that automatically uses the appropriate IPv6 ping function when IPv6 support is compiled in.

Parameters

<i>to</i>	Destination IP address to ping
<i>id</i>	ICMP identifier field
<i>seq</i>	ICMP sequence number field
<i>wait</i>	Maximum wait time in system time ticks
<i>interface</i>	Network interface number to use
<i>size</i>	Size of ping data payload in bytes (default: 32)

Returns

Same as [Ping6ViaInterface\(\)](#)

See also

[Ping6ViaInterface\(\)](#), [Ping4ViaInterface\(\)](#)

14.26.3.34 SendICMPError()

```
void SendICMPError (
    PoolPtr pBadPacket,
    uint8_t type,
    uint8_t code)
```

```
#include <ip.h>
```

Send an ICMP error message in response to a bad packet.

Generates and sends an appropriate ICMP error message in response to a malformed or problematic IP packet. The original packet data is included in the ICMP error message as required by the ICMP specification (typically the first 8 bytes of the original packet's payload).

Parameters

<i>pBadPacket</i>	Pointer to the buffer containing the problematic packet
<i>type</i>	ICMP error type code (e.g., ICMP_TYPE_DEST_UNREACHABLE)
<i>code</i>	ICMP error code within the type (e.g., ICMP_CODE_PORT_UNREACHABLE)

Note

The function automatically includes the required portion of the original packet.

ICMP error messages are rate-limited to prevent ICMP flooding.

The buffer pBadPacket is typically freed by this function.

See also

ICMP type and code definitions in [ip.h](#)

14.26.3.35 ShowIPBuffer()

```
void ShowIPBuffer (
    PoolPtr rp)
```

```
#include <ip.h>
```

Display the contents of a network buffer for diagnostic purposes.

This function provides a formatted dump of the packet contents in the specified network buffer pool buffer. The output is sent using `iprintf()` and includes both hexadecimal and ASCII representations of the packet data, making it easy to analyze packet structure and contents.

Typical output format:

- Offset in hexadecimal
- 16 bytes per line in hex
- ASCII representation of printable characters
- Non-printable characters shown as '.'

Parameters

<i>rp</i>	Packet pointer from network buffer pool to display

Note

Uses `iprintf()` for output, so output destination depends on the current `iprintf` configuration (typically serial console).

Useful for debugging network packet issues and protocol analysis.

Does not modify or free the buffer.

```
// Display contents of received packet for debugging
ShowIPBuffer(received_packet);
```


14.26.4 Variable Documentation

14.26.4.1 pFragFunc

```
FragmentProcessFunction* pFragFunc [extern]
```

```
#include <ip.h>
```

Global function pointer for fragment processing callback.

Set this pointer to your FragmentProcessFunction implementation to handle fragmented IP packets. Set to NULL (default) to disable fragment processing callbacks, in which case fragmented packets will be silently dropped.

```
void MyFragmentHandler(PoolPtr pp) {
    // Process fragmented packet
    // Store in reassembly buffer
    // Must free the buffer when done or store for later reassembly
    FreeBuffer(pp);
}

// Register the callback during initialization
pFragFunc = MyFragmentHandler;
```

Note

Default value is NULL (fragmented packets dropped).

Only one global handler can be registered.

14.26.4.2 pPromiscuousPacketFunc

```
PromiscuousPacketFunc* pPromiscuousPacketFunc [extern]
```

```
#include <ip.h>
```

Global function pointer for promiscuous packet processing callback.

Set this pointer to your PromiscuousPacketFunc implementation to handle packets not addressed to this device. Set to NULL (default) to disable promiscuous mode, in which case non-local packets will be silently dropped.

```
void MyPromiscuousHandler(PoolPtr pp) {
    // Log packet for monitoring
    ShowIPBuffer(pp);

    // MUST free the buffer when done
    FreeBuffer(pp);
}

// Register the callback
pPromiscuousPacketFunc = MyPromiscuousHandler;
```

Note

Default value is NULL (non-local packets dropped).

Ethernet interface must also be configured for promiscuous mode.

Only one global handler can be registered.

14.26.4.3 pTaskPacketFunc

```
TaskPacketFunc* pTaskPacketFunc [extern]
```

```
#include <ip.h>
```

Global function pointer for UDP task packet processing callback.

Set this pointer to your TaskPacketFunc implementation to handle UDP packets in task context. Set to NULL (default) to disable task-level UDP packet processing.

Note

Default value is NULL (task-level processing disabled).

Only one global handler can be registered.

Regular UDP socket handlers are not affected by this callback.

14.26.5 IPADDR4 Class

Classes

- class [MACADR](#)

Used to store and manipulate MAC addresses.

- class [IPADDR4](#)

Used to store and manipulate IPv4 addresses in dual stack mode.

Typedefs

- typedef class MACADR **MACADR**

Used to store and manipulate MAC addresses.

- typedef [IPADDR6](#) **IPADDR**

IPADDR Object Type (either v4 or v6)

Functions

- bool **operator==** (const [MACADR](#) &i, const [MACADR](#) &j)
Check MAC equality.
- bool **operator!=** (const [MACADR](#) &i, const [MACADR](#) &j)
Check MAC inequality.
- bool **operator>** (const [MACADR](#) &i, const [MACADR](#) &j)
Check MAC greater than.

14.26.5.1 Detailed Description

```
#include< nettypes.h>
```

The IPADDR C++ objects were created to provide a portable and easy implementation to support IPv4 and IPv6 IP addresses. There are 3 types of IPADDR objects:

- [IPADDR4](#): Used exclusively for IPv4 addresses
- IPADDR: Used for either an IPv6 or IPv4 address
- **IPADDR**: The typedef [IPADDR](#) defines a type IPADDR. Its purpose is to provide clarity that an IPADDR object can be used for both IPv4 and IPv6 addresses. Any functions called with a parameter of type IPADDR will automatically execute the correct IPv4 or IPv6 underlying function.

14.26.6 IPADDR6 Class

Classes

- class [IPADDR6](#)

Used to hold and manipulate IPv4 and IPv6 addresses in dual stack mode.

14.26.6.1 Detailed Description

```
#include< ipv6_addr.h>
```

The IPADDR C++ objects were created to provide a portable and easy implementation to support IPv4 and IPv6 IP addresses. There are 3 types of IPADDR objects:

- [IPADDR4](#): Used exclusively for IPv4 addresses
- [IPADDR6](#): Used for either an IPv6 or IPv4 address
- **IPADDR**: The typedef [IPADDR](#) defines a type [IPADDR6](#). Its purpose is to provide clarity that an IPADDR object can be used for both IPv4 and IPv6 addresses. Any functions called with a parameter of type IPADDR will automatically execute the correct IPv4 or IPv6 underlying function.

14.27 IPv6 Interface Management

Topics

- [IPv6 DNS](#)
- [IPv6 Data Structures](#)
- [IPv6 Frame Processing](#)
- [IPv6 Multicast Constants](#)
- [IPv6 Neighbor Discovery](#)
- [IPv6 Routing](#)

Classes

- class [IPv6Interface](#)
Main IPv6 interface management class.

Functions

- void [ShowIP6Counters](#) ()
Display IPv6 counters and statistics.

14.27.1 Detailed Description

```
#include< ipv6_interface.h>
```

Core IPv6 interface operations including address management, routing, and neighbor discovery.

14.27.2 Function Documentation

14.27.2.1 ShowIP6Counters()

```
void ShowIP6Counters ()  
#include <ipv6_interface.h>  
Display IPv6 counters and statistics.  
Dumps IPv6 protocol counters to stdout for debugging and monitoring.
```

14.27.3 IPv6 DNS

Classes

- struct [IPV6_DNS](#)
IPv6 DNS server information element.

14.27.3.1 Detailed Description

```
#include< ipv6_interface.h>
```

IPv6 DNS server configuration and management.

14.27.4 IPv6 Data Structures

Classes

- struct [IPV6_ROOT_EL](#)
Base class for all IPv6 interface elements.
- struct [IPV6_ROOT_EL_CONTAINER](#)
Container for managing collections of IPv6 root elements.

- struct [IPV6_DHCPD](#)
DHCPv6 server information element.
- struct [IPV6_IA_ADDR](#)
DHCPv6 Identity Association address element.

Enumerations

- enum [IPV6_ROOT_EL_TYPE](#) {
[IPV6_ROOT_TYPE_EL](#) = 0 , [IPV6_ROOT_TYPE_EL_CONTAINER](#) = 1 , [IPV6_ROOT_TYPE_ROUTER](#) = 2
, [IPV6_ROOT_TYPE_DHCPD](#) = 3 ,
[IPV6_ROOT_TYPE_PREFIX](#) = 4 , [IPV6_ROOT_TYPE_DNS](#) = 5 , [IPV6_ROOT_TYPE_NEIGHBOR](#) = 6 ,
[IPV6_ROOT_TYPE_DEST](#) = 7 }
IPv6 root element types enumeration.

14.27.4.1 Detailed Description

```
#include< ipv6_interface.h>
```

Core data structures for IPv6 interface management.

14.27.4.2 Enumeration Type Documentation

14.27.4.2.1 IPV6_ROOT_EL_TYPE

```
enum IPV6\_ROOT\_EL\_TYPE
#include <ipv6_interface.h>
```

IPv6 root element types enumeration.
Defines the different types of elements that can be stored in IPv6 interface containers.

Enumerator

IPV6_ROOT_TYPE_EL	Base root element.
IPV6_ROOT_TYPE_EL_CONTAINER	Element container.
IPV6_ROOT_TYPE_ROUTER	Router element.
IPV6_ROOT_TYPE_DHCPD	DHCPv6 server element.
IPV6_ROOT_TYPE_PREFIX	Address prefix element.
IPV6_ROOT_TYPE_DNS	DNS server element.
IPV6_ROOT_TYPE_NEIGHBOR	Neighbor cache element.
IPV6_ROOT_TYPE_DEST	Destination cache element.

14.27.5 IPv6 Frame Processing

Classes

- struct [IPv6FrameProcessingStruct](#)
IPv6 frame processing structure for packet analysis and header walking.

14.27.5.1 Detailed Description

```
#include< ipv6_interface.h>
```

IPv6 packet processing and header walking functionality.

14.27.6 IPv6 Multicast Constants

Macros

- `#define ADDR_ALL_NODES_LINK_LOCAL IPADDR6::AsciiToIp6("FF02::1")`
All nodes on the local link.
- `#define ADDR_ALL_ROUTERS_LINK_LOCAL IPADDR6::AsciiToIp6("FF02::2")`
All routers on the local link.
- `#define ADDR_ALL_NB_LINK_LOCAL IPADDR6::AsciiToIp6("FF02::4e42")`
All NetBurner devices on the local link.
- `#define ADDR_UNICAST_LINK_LOCAL_PREFIX IPADDR6::AsciiToIp6("FE80::")`
Link-local unicast prefix.
- `#define ADDR_MLDV2_REPORTS IPADDR6::AsciiToIp6("FF02::16")`
MLDv2 reports multicast address.
- `#define IPV6_MULTICAST_NODE_LOCAL 0xFF01`
Node-local multicast scope.
- `#define IPV6_MULTICAST_LINK_LOCAL 0xFF02`
Link-local multicast scope.
- `#define IPV6_MULTICAST_REALM_LOCAL 0xFF03`
Realm-local multicast scope.
- `#define IPV6_MULTICAST_ADMIN_LOCAL 0xFF04`
Admin-local multicast scope.
- `#define IPV6_MULTICAST_SITE_LOCAL 0xFF05`
Site-local multicast scope.
- `#define IPV6_MULTICAST_ORG_LOCAL 0xFF08`
Organization-local multicast scope.
- `#define IPV6_MULTICAST_GLOBAL 0xFF0E`
Global multicast scope.

14.27.6.1 Detailed Description

```
#include< ipv6_interface.h>
```

Standard IPv6 multicast addresses and scope constants.

14.27.7 IPv6 Neighbor Discovery

Classes

- struct `IPV6_NEIGHBOR`
IPv6 neighbor cache element.

Enumerations

- enum `eMY_NEIGHBOR_STATE` {
 `eIncomplete` , `eReachable` , `eStale` , `eDelay` ,
 `eProbe` }
IPv6 neighbor states enumeration.

14.27.7.1 Detailed Description

```
#include< ipv6_interface.h>
```

IPv6 Neighbor Discovery protocol support and neighbor cache management.

14.27.7.2 Enumeration Type Documentation

14.27.7.2.1 eMY_NEIGHBOR_STATE

enum [eMY_NEIGHBOR_STATE](#)

```
#include <ipv6_interface.h>
```

IPv6 neighbor states enumeration.

Defines the various states a neighbor entry can be in according to RFC 4861.

Enumerator

eIncomplete	Address resolution in progress.
eReachable	Reachability confirmed within ReachableTime.
eStale	More than ReachableTime since reachability was confirmed.
eDelay	More than ReachableTime, packet sent, waiting for upper layer.
eProbe	Actively probing neighbor reachability.

14.27.8 IPv6 Routing

Classes

- struct [IPV6_ROUTER](#)
IPv6 router information element.
- struct [IPV6_PREFIX](#)
IPv6 prefix and address management element.
- struct [IPV6_DEST](#)
IPv6 destination cache element.

Enumerations

- enum [eMY_ADDR_STATE](#) { [eTenative](#) , [eValid_Preferred](#) , [eValid_Depricated](#) , [eInvalid](#) }
IPv6 address states enumeration.
- enum [ePrefixSource](#) {
 [eLinkLocal](#) , [eRouter](#) , [eDHCP](#) , [eStatic](#) ,
 [eUnknown](#) }
IPv6 prefix source enumeration.
- enum [eRouteOutResult](#) { [eSent](#) , [eDoingND](#) , [eNoRoute](#) }
Route output result enumeration.

14.27.8.1 Detailed Description

```
#include< ipv6_interface.h>
```

IPv6 routing and destination management functionality.

14.27.8.2 Enumeration Type Documentation

14.27.8.2.1 eMY_ADDR_STATE

enum [eMY_ADDR_STATE](#)

```
#include <ipv6_interface.h>
```

IPv6 address states enumeration.

Defines the various states an IPv6 address can be in during its lifecycle.

Enumerator

eTentative	Address is tentative (duplicate address detection in progress)
eValid_Preferred	Address is valid and preferred.
eValid_Deprecated	Address is valid but deprecated.
eInvalid	Address is invalid.

14.27.8.2.2 ePrefixSource

```
enum ePrefixSource
#include <ipv6_interface.h>
```

IPv6 prefix source enumeration.
Identifies how an IPv6 prefix was obtained.

Enumerator

eLinkLocal	Link-local address.
eRouter	From router advertisement.
eDHCP	From DHCPv6.
eStatic	Statically configured.
eUnknown	Unknown source.

14.27.8.2.3 eRouteOutResult

```
enum eRouteOutResult
#include <ipv6_interface.h>
```

Route output result enumeration.
Results from attempting to route a packet to a destination.

Enumerator

eSent	Packet was sent successfully.
eDoingND	Performing neighbor discovery.
eNoRoute	No route available.

14.28 Initialization - System Initialization Functions

Classes

- struct [listen_list_item](#)
Structure to use to create custom lists of listening ports and interfaces.

Functions

- void [init](#) ()
System initialization. Ideally called at the beginning of all applications, since the easiest Recovery procedures pause execution inside of `init`, which is impossible if there is a crash before `init`.
- void [StartHttp](#) (uint16_t port=80)
Start the HTTP web server.
- void [StartHttps](#) (uint16_t ssl_port=443, uint16_t http_port=80)
Start the HTTPS secure web server.
- void [SetHttpsPointers](#) ()

If you are both using the Start Custom below and using https you must call this.

- void [StartHttpCustom](#) ([listen_list_item](#) *pList_Of_Listens)
Start the HTTP web server with a custom list of listen ports.
- bool [WaitForActiveNetwork](#) (uint32_t ticks_to_wait=120 *[TICKS_PER_SECOND](#), int interface=-1)
Wait for an active network connection on at least one interface.
- void [EnableSystemDiagnostics](#) ()
Turn on the diagnostic reports from the config page.
- void [EnableSecureConfigServer](#) (bool bSec_Only)
Enable the minimal http config server to operate over TLS.

14.28.1 Detailed Description

```
#include< init.h>
```

14.28.2 Function Documentation

14.28.2.1 EnableSecureConfigServer()

```
void EnableSecureConfigServer (
    bool bSec_Only)
```

```
#include <init.h>
```

Enable the minimal http config server to operate over TLS.

Due to the nature of the server operating on a non-standard HTTP port, a client (such as a webbrowser) cannot infer whether to utilize TLS unless explicitly told to do so. Therefore, the secure configuration server has two modes it can operate in: a less secure, more compliant manner that allows both plaintext and TLS secured transactions and a HSTS (Http Strict Transport Security) mode that will only respond to plaintext requests with a redirect indicating the TLS requirement.

Parameters

<i>bSec_Only</i>	Whether the SecureConfigServer should disallow non-secure requests

14.28.2.2 EnableSystemDiagnostics()

```
void EnableSystemDiagnostics ()
#include <init.h>
```

Turn on the diagnostic reports from the config page.

This should be used under development. In production it has no performance impact, but could leak possibly sensitive information.

14.28.2.3 init()

```
void init ()
#include <init.h>
```

System initialization. Ideally called at the beginning of all applications, since the easiest Recovery procedures pause execution inside of `init`, which is impossible if there is a crash before `init`.

Initialize the system including the following:

- Enable interrupt driven stdio
- Read the configuration record and set parameters accordingly. If the configuration specifies [DHCP](#), start the [DHCP](#) Client. All interfaces will be checked.
- Initialize the network stack

- Start the configuration server for both web and serial
- Assign the UserMain() task priority to MAIN_PRIO
- Enable the Task Monitor diagnostic tool
- Initialize the GDB stub if in debug mode

14.28.2.4 StartHttp()

```
void StartHttp (
    uint16_t port = 80)
```

```
#include <init.h>
```

Start the HTTP web server.

If no parameters are specified it will listen on port 80, and enable the configuration mirror feature so an application can create its own custom configuration web page.

Parameters

<i>port</i>	Listen port, defaults to port 80

See also

[StartHttps\(\)](#), [StopHttp\(\)](#)

14.28.2.5 StartHttpCustom()

```
void StartHttpCustom (
    listen_list_item * pList_Of_Listens)
```

```
#include <init.h>
```

Start the HTTP web server with a custom list of listen ports.

If no parameters are specified it will listen on port 80, and enable the configuration mirror feature so an application can create its own custom configuration web page.

Parameters

<i>pList_Of_Listens</i>	A static array of listening sockets the last entry must be zero.

See also

[StartHttps\(\)](#), [StopHttp\(\)](#) [SetHttpsPointers\(\)](#);

14.28.2.6 StartHttps()

```
void StartHttps (
    uint16_t ssl_port = 443,
    uint16_t http_port = 80)
```

```
#include <init.h>
```

Start the HTTPS secure web server.

If no parameters are specified the web server will listen on ports 443 and 80, and enable the configuration mirror feature so an application can create its own custom configuration web page. If no user key is installed the default system key will be used.

Parameters

<i>ssl_port</i>	Secure listen port, defaults to port 443
<i>http_port</i>	Non-secure listen port, defaults to port 80

See also

[StartHttp\(\)](#), [StopHttp\(\)](#)

14.28.2.7 WaitForActiveNetwork()

```
bool WaitForActiveNetwork (
    uint32_t ticks_to_wait = 120 *TICKS_PER_SECOND,
    int interface = -1)
#include <init.h>
```

Wait for an active network connection on at least one interface.

If [DHCP](#) is enabled it will wait for a lease. If static IP parameters are configured it will wait for link.

Parameters

<i>ticks_to_wait</i>	Number of system time ticks to wait. Default is 120 seconds.
<i>interface</i>	Interface number to check. Default is -1, which checks all interfaces.

14.29 JSON Lexer

NetBurner's JSON library. See more details and examples below.

Classes

- struct [JsonAllocString](#)
A list of large strings that are created with malloc.
- class [JsonRef](#)
Represents a positional reference (pointer) of a location inside a [ParsedJsonDataSet](#) object.
- class [ParsedJsonDataSet](#)
A class to create, read, and modify a JSON object.
- class [JsonLexerFDAdapter](#)
Get the type name of the JSON primitive type passed in.

Typedefs

- typedef void [CharOutputFn](#)(const char *chars, int len, void *blob)
Helper function typedef for print functions.

Enumerations

- enum [json_primitive_type](#) {
[UNDEFINED](#) , [BEGIN_ARRAY](#) , [BEGIN_OBJECT](#) , [END_ARRAY](#) ,
[END_OBJECT](#) , [NAME](#) , [STRING](#) , [VALUE_SEPERATOR](#) ,
[NUMBER](#) , [FALSE_EL](#) , [TRUE_EL](#) , [NULL_EL](#) ,
[STRING_TOO_BIG](#) , [STRING_CHUNK](#) , [ALLOC_STRING](#) , [NOTFOUND](#) ,
[EOF_EL](#) }

The following types define the basic building blocks that make up a JSON data set. These are the values that will be returned from the functions used to parse the data set. Member functions include operators to return specific data type, as well as type validity checks.

14.29.1 Detailed Description

NetBurner's JSON library. See more details and examples below.

```
#include< json_lexer.h>
```

The library enables rapid, performant parsing, traversal, and querying of JSON data. It also works seamlessly as a [buffer_object](#) for the [Web Client](#)'s [DoGet\(\)](#) function.

[ParsedJsonDataSet](#) is the root object that holds the document, and [JsonRef](#) is the result of each query function, serving as a pointer to a location in the document but also as a chainable query and inspection interface.

See also the [JSON Lexer Example Applications](#).

Example

```
ParsedJsonDataSet pjd;
bool result = DoGet("https://www.example.com/users.json", pjd);
// pjd.PrintObject(true);
// pjd("users").PrintChildren(true);
if (!result ||
    !pjd("users")[0].IsObject() ||
    !pjd("users")[0]("id").IsNumber() ||
    !pjd("users")[0]("name").IsString()
) {
    return false;
}
printf("Got user %d name %s",
       (int)pjd("users")[0]("id"),
       (const char *)pjd("users")[0]("name")
);
```

14.29.2 Typedef Documentation

14.29.2.1 CharOutputFn

```
typedef void CharOutputFn(const char *chars, int len, void *blob)
```

```
#include <json_lexer.h>
```

Helper function typedef for print functions.

Parameters

<i>chars</i>	The characters to print.
<i>len</i>	The length of the characters to print.
<i>blob</i>	A pointer to where they should be printed.

14.29.3 Enumeration Type Documentation

14.29.3.1 json_primitive_type

```
enum json_primitive_type
#include <json_lexer.h>
```

The following types define the basic building blocks that make up a JSON data set. These are the values that will be returned from the functions used to parse the data set. Member functions include operators to return specific data type, as well as type validity checks.

Enumerator

UNDEFINED	Not a defined primitive type. This indicates an error somewhere.
BEGIN_ARRAY	[- Signals the start of an array.
BEGIN_OBJECT	{ - Signals the start of an object.
END_ARRAY	] - Signals the end of an array.
END_OBJECT	} - Signals the end of an object.
NAME	"xxx": - Signals the name of a name/value pair.
STRING	"xxx" - An element with a string value.

Enumerator

VALUE_SEPERATOR	, - Signals the separation between to arrays, objects, or values.
NUMBER	An element with a number value.
FALSE_EL	An element with the value false.
TRUE_EL	An element with the value true.
NULL_EL	An element with the null value.
STRING_TOO_BIG	Error, we got a sting but it was too big to fit in the storage scheme (1500 bytes).
STRING_CHUNK	Not quiet an error, but we got a sting but it was too big to fit in the storage scheme (1500 bytes).
ALLOC_STRING	We allocated a large string.
NOTFOUND	Return value when we don't find what we are looking for.
EOF_EL	Last token in the data set.

14.30 MQTT

NetBurner [MQTT](#) Library.

See the [MQTT](#) namespace for details.

```
#include <mqtt/mqtt.h>
```

Namespaces

- namespace [MQTT](#)
[MQTT](#) Namespace.

Macros

- #define **MQTT_WAIT** ([TICKS_PER_SECOND](#)*10)
- #define **MQTT_TLS_NEGOTIATE_TIMEOUT** ([TICKS_PER_SECOND](#)*50)
- #define **MQTT_OBJ_PRIORITY_COUNT** (16)
- #define **MQTT_OBJ_PRIORITY_MAX** 0
- #define **MQTT_OBJ_PRIORITY_MIN** 15
- #define **MQTT_MAX_PKT_SIZE** (0x2FFFFFFF)
- #define **MQTT_MAX_MAX_SUPPORTED_IN_FLIGHT** (256)
- #define **MQTT_MAX_SUPPORTED_IN_FLIGHT** (16)
- #define **MQTT_MAX_HANDLERS_PER_CONNECTION** (256)
- #define **MQTT_MAX_SUPPORTED_RX_TOPIC_ALIASES** (128)
- #define **MQTT_DEFAULT_SESSION_EXPIRY_INTERVAL** (3600)
- #define **MQTT_DEFAULT_KEEP_ALIVE** (30)
- #define **MQTT_MAX_SUB_IDS** (128)

14.30.1 Detailed Description

NetBurner [MQTT](#) Library.

See the [MQTT](#) namespace for details.

```
#include <mqtt/mqtt.h>
```

To easily begin using [MQTT](#), review the Simple [MQTT](#) example in the Examples folder. In general, all that's needed is:

- `#include <mqtt/mqtt.h>`
- `MQTT::ConfiguredClient mqttClient;`
- and optionally some kind of [MQTT](#) variable like `MQTT::mqtt_int myInt("myIntTopic");`

As soon as an [MQTT](#) client exists in your code, an [MQTT](#) section will be added to your device's Configuration page allowing you to set a broker hostname, Client ID, and other connection or security details.

14.31 Multicast

Functions

- void [RegisterMulticastFifo4](#) ([IPADDR4](#) group, uint16_t dest_port, OS_FIFO *pfifo, int interface=0)
Register to join a Multicast group.
- void [UnregisterMulticastFifo4](#) ([IPADDR4](#) group, uint16_t destination_port, int interface=0)
Unregister from a Multicast group.
- void [RegisterMulticastFifo6](#) ([IPADDR](#) group, uint16_t dest_port, OS_FIFO *pfifo, int interface=0)
Register to join a Multicast group.
- void [UnregisterMulticastFifo6](#) ([IPADDR](#) group, uint16_t destination_port, int interface=0)
Unregister from a Multicast group.

14.31.1 Detailed Description

```
#include< multicast.h>
```

The NetBurner Multicast API

14.31.2 Function Documentation

14.31.2.1 RegisterMulticastFifo4()

```
void RegisterMulticastFifo4 (
    IPADDR4 group,
    uint16_t dest_port,
    OS_FIFO * pfifo,
    int interface = 0)
```

```
#include <multicast.h>
```

Register to join a Multicast group.

Calling RegisterMulticastFifo() in dual stack mode will automatically select the correct IPv4/IPv6 function.

IGMP Multicast is a method for distributing UDP packets within a group of hosts and servers. The NetBurner Multicast functions extend the NetBurner UDP interface. Instead of [RegisterUDPFifo\(\)](#), the RegisterMulticastFifo() function is used to listen for Multicast UDP packets. To transmit Multicast packets, use the UDP Send function with a multicast IP address.

Parameters

<i>group</i>	The IP address of the group to join
<i>dest_port</i>	The port number of the group to join
<i>pfifo</i>	Pointer to the FIFO to store incoming packets

<i>interface</i>	Optional interface number. If not specified or 0, the first system interface will be used.

See also

[UnregisterMulticastFifo4\(\)](#)

14.31.2.2 RegisterMulticastFifo6()

```
void RegisterMulticastFifo6 (
    IPADDR group,
    uint16_t dest_port,
    OS_FIFO * pfifo,
    int interface = 0)
```

```
#include <multicast.h>
```

Register to join a Multicast group.

Calling RegisterMulticastFifo() in dual stack mode will automatically select the correct IPv4/IPv6 function.

IGMP Multicast is a method for distributing UDP packets within a group of hosts and servers. The NetBurner Multicast functions extend the NetBurner UDP interface. Instead of [RegisterUDPFifo\(\)](#), the RegisterMulticastFifo() function is used to listen for Multicast UDP packets. To transmit Multicast packets, use the UDP Send function with a multicast IP address.

Parameters

<i>group</i>	The IP address of the group to join
<i>dest_port</i>	The port number of the group to join
<i>pfifo</i>	Pointer to the FIFO to store incoming packets
<i>interface</i>	Optional interface number. If not specified or 0, the first system interface will be used.

See also

[UnregisterMulticastFifo4\(\)](#)

14.31.2.3 UnregisterMulticastFifo4()

```
void UnregisterMulticastFifo4 (
    IPADDR4 group,
    uint16_t destination_port,
    int interface = 0)
```

```
#include <multicast.h>
```

Unregister from a Multicast group.

Calling UnregisterMulticastFifo() in dual stack mode will automatically select the correct IPv4/IPv6 function.

Parameters

<i>group</i>	The IP address of the group to leave
<i>destination_port</i>	The port number of the group to leave
<i>interface</i>	Specifies the interface number, the default is 0

See also

[RegisterMulticastFifo4\(\)](#)

14.31.2.4 UnregisterMulticastFifo6()

```
void UnregisterMulticastFifo6 (
    IPADDR group,
    uint16_t destination_port,
    int interface = 0)
```

#include <multicast.h>

Unregister from a Multicast group.

Calling UnregisterMulticastFifo() in dual stack mode will automatically select the correct IPv4/IPv6 function.

Parameters

<i>group</i>	The IP address of the group to leave
<i>destination_port</i>	The port number of the group to leave
<i>interface</i>	Optional interface number. If not specified or 0, the first system interface will be used.

See also

[RegisterMulticastFifo4\(\)](#)

14.32 Multihome and VLAN

Functions

- int [AddVlanInterface](#) (IPADDR4 addr, IPADDR4 mask, IPADDR4 gateway, uint16_t vlan_tag, const char *ParentName)
Add a VLAN interface with a Parent Name.
- int [AddVlanInterface](#) (IPADDR4 addr, IPADDR4 mask, IPADDR4 gateway, uint16_t vlan_tag, [InterfaceBlock](#) &parent)
Add a VLAN interface with a Parent [InterfaceBlock](#) reference.
- int [AddVlanInterface](#) (IPADDR4 addr, IPADDR4 mask, IPADDR4 gateway, uint16_t vlan_tag, int root_if=0)
Add a VLAN interface with an interface number.
- int [AddInterface](#) (IPADDR4 addr, IPADDR4 mask, IPADDR4 gateway, const char *ParentName)
Add an interface with a Parent Name.
- int [AddInterface](#) (IPADDR4 addr, IPADDR4 mask, IPADDR4 gateway, [InterfaceBlock](#) &parent)
Add an interface with a Parent [InterfaceBlock](#) reference.
- int [AddInterface](#) (IPADDR4 addr, IPADDR4 mask, IPADDR4 gateway, int root_if)
Add an interface with an interface number.

14.32.1 Detailed Description

```
#include< multihome.h>
```

Create Multihome and VLAN Interfaces

14.32.2 Function Documentation

14.32.2.1 AddInterface() [1/3]

```
int AddInterface (
    IPADDR4 addr,
    IPADDR4 mask,
    IPADDR4 gateway,
    const char * ParentName) [inline]
```

#include <multihome.h>

Add an interface with a Parent Name.

Parameters

<i>addr</i>	IP address of new interface
<i>mask</i>	IP mask
<i>gateway</i>	IP gateway
<i>ParentName</i>	Name of parent physical interface, such as Ethernet0.

returns Interface number of newly created interface

14.32.2.2 AddInterface() [2/3]

```
int AddInterface (
    IPADDR4 addr,
    IPADDR4 mask,
    IPADDR4 gateway,
    int root_if) [inline]
#include <multihome.h>
Add an interface with an interface number.
Add a new network interface with the specified configuration.
```

Parameters

<i>addr</i>	IP address of new interface
<i>mask</i>	IP mask
<i>gateway</i>	IP gateway
<i>root_if</i>	Optional root physical interface number. If not specified the default is 0, for Ethernt 0.

Returns

Interface number of newly created interface

14.32.2.3 AddInterface() [3/3]

```
int AddInterface (
    IPADDR4 addr,
    IPADDR4 mask,
    IPADDR4 gateway,
    InterfaceBlock & parent) [inline]
#include <multihome.h>
Add an interface with a Parent InterfaceBlock reference.
```

Parameters

<i>addr</i>	IP address of new interface
<i>mask</i>	IP mask
<i>gateway</i>	IP gateway
<i>parent</i>	Reference to parent physical interface of type InterfaceBlock .

returns Interface number of newly created interface

14.32.2.4 AddVlanInterface() [1/3]

```
int AddVlanInterface (
    IPADDR4 addr,
    IPADDR4 mask,
    IPADDR4 gateway,
```



```

        uint16_t vlan_tag,
        const char * ParentName)
#include <multihome.h>
Add a VLAN interface with a Parent Name.

```

Parameters

<i>addr</i>	IP address of new interface
<i>mask</i>	IP mask
<i>gateway</i>	IP gateway
<i>vlan_tag</i>	VLAN tag/id for interface
<i>ParentName</i>	Name of parent physical interface, such as Ethernet0.

returns Interface number of newly created VLAN interface

14.32.2.5 AddVlanInterface() [2/3]

```

int AddVlanInterface (
    IPADDR4 addr,
    IPADDR4 mask,
    IPADDR4 gateway,
    uint16_t vlan_tag,
    int root_if = 0)
#include <multihome.h>
Add a VLAN interface with an interface number.

```

Parameters

<i>addr</i>	IP address of new interface
<i>mask</i>	IP mask
<i>gateway</i>	IP gateway
<i>vlan_tag</i>	VLAN tag/id for interface
<i>root_if</i>	Optional root physical interface number. If not specified the default is 0 for Ethernet 0.

returns Interface number of newly created VLAN interface

14.32.2.6 AddVlanInterface() [3/3]

```

int AddVlanInterface (
    IPADDR4 addr,
    IPADDR4 mask,
    IPADDR4 gateway,
    uint16_t vlan_tag,
    InterfaceBlock & parent)
#include <multihome.h>
Add a VLAN interface with a Partent InterfaceBlock reference.

```

Parameters

<i>addr</i>	IP address of new interface
<i>mask</i>	IP mask
<i>gateway</i>	IP gateway
<i>vlan_tag</i>	VLAN tag/id for interface
<i>parent</i>	Reference to parent physical interface of type InterfaceBlock

returns Interface number of newly created VLAN interface

14.33 NBRTOS Real Time Operating System

```
#include< nbrtos.h>
```

The NetBurner Real-Time OS

14.34 NBString - NetBurner String Class

Classes

- class [NBString](#)
Lightweight alternative to C++ CString class.
- class [NBInterpolatedString](#)
A wrapper for [NBString::Interpolate\(\)](#) ideal for easy access to ephemeral values.

14.34.1 Detailed Description

```
#include< nbstring.h>
```

The NetBurner lightweight String Class alternative to CString.

14.35 Network Interfaces

Network Interface Functions.

Topics

- [IPADDR4 Functions](#)
Network Interface functions with [IPADDR4](#) return types.
- [Low Level Processing \(NetDoRx\)](#)
Callback functions to add custom Ethernet handlers and pass Ethernet frames at the bottom of the TCP/IP stack.

Classes

- class [InterfaceBlock](#)
Network interface configuration block class for interface control and configuration.

Functions

- void **showIpAddresses** ()
Print system IP addresses to stdout.
- int [Removeinterface](#) (int interface)
Remove a network interface from the system.
- void [EnableMulticast](#) ([MACADR](#) macAddress, int interface=0)
Enable Multicast on an existing interface.
- void [DisableMulticast](#) ([MACADR](#) macAddress, int interface=0)
Disable Multicast on an existing interface.
- [InterfaceBlock](#) * [GetInterfaceBlock](#) (int interface=0)
Get an [InterfaceBlock](#) control and configuration object.
- int32_t [GetFirstInterface](#) (void)
Returns the Interface Number of the first registered network interface.
- int32_t [GetNextInterface](#) (int lastInterface)

- Returns the Interface Number of the next registered network interface.*
- `int32_t GetInterfaceNumber (InterfaceBlock *pifb)`
Returns the Interface Number of the specified network interface [InterfaceBlock](#).
- `int32_t GetInterfaceForMyAddress4 (IPADDR4 ip)`
Returns the Interface Number of the specified network interface IPv4 address.
- `bool GetInterfaceLink (int ifn)`
Returns the network interface link status.
- `MACADDR InterfaceMAC (int interface)`
Returns the MAC address of the specified network interface.
- `bool InterfaceLinkActive (int interface)`
Returns the link status of the specified network interface.
- `int InterfaceLinkSpeed (int interface)`
Returns the 10/100 link speed of the specified network interface.
- `bool InterfaceLinkDuplex (int interface)`
Returns the full or half link duplex of the specified network interface.
- `const char * InterfaceName (int interface)`
Returns the interface name of the specified network interface.

14.35.1 Detailed Description

Network Interface Functions.

Network interface management and low-level frame processing.

```
#include< ipshow.h>
```

```
#include< netinterface.h>
```

Network Interfaces are configured and controlled C++ [InterfaceBlock](#) objects. The [InterfaceBlock](#) contains variables and member functions for things such as:

- The Configuration Server interface
- IP settings for both IPv4 and IPv6 types
- Configuration to static or [DHCP](#), and [DHCP](#) status (both v4 and v6)
- Multihome
- Routing
- AutoIP

Note

While the recommended method for getting and setting network interface parameters is to obtain a pointer to an [InterfaceBlock](#), there are also non-member functions of the object that can be used for situations that are best served by a single function call, such as just getting a single IP address of an interface. When using those functions pay special attention to the return value type. While `IPADDR` should be used because it supports both [IPADDR4](#) and [IPADDR6](#) types automatically, some of these helper functions may specifically return an [IPADDR4](#) type.

```
#include< netrx.h>
```

14.35.2 Function Documentation

14.35.2.1 DisableMulticast()

```
void DisableMulticast (
    MACADDR macAddress,
    int interface = 0)
#include <netinterface.h>
```

Disable Multicast on an existing interface.
Calls registered interface multicast routine.

Parameters

<i>macAddress</i>	Multicast MAC address.
<i>interface</i>	Interface number. If not specified or 0, operates on the first registerd interface.

See also

[EnableMulticast\(MACADDR macAddress, int interface\)](#), [InterfaceBlock::EnableMulticast\(\)](#)

14.35.2.2 EnableMulticast()

```
void EnableMulticast (
    MACADDR macAddress,
    int interface = 0)
#include <netinterface.h>
```

Enable Multicast on an existing interface.
Calls registered interface multicast routine.

Parameters

<i>macAddress</i>	Multicast MAC address.
<i>interface</i>	Interface number. If not specified, default is is 0.

See also

[DisableMulticast\(\)](#), [InterfaceBlock::EnableMulticast\(\)](#)

14.35.2.3 GetFirstInterface()

```
int32_t GetFirstInterface (
    void )
#include <netinterface.h>
```

Returns the Interface Number of the first registered network interface.
If called on a device with wired Ethernet, this will be the first Ethernet interface typical interface number 1.

Returns

Interface number of the first network interface, or 0 if none exist.

14.35.2.4 GetInterfaceBlock()

```
InterfaceBlock * GetInterfaceBlock (
    int interface = 0)
#include <netinterface.h>
```

Get an [InterfaceBlock](#) control and configuration object.

Parameters

<i>interface</i>	Interface number. If not specified or 0, operates on the first registered interface.

Returns

Pointer to an [InterfaceBlock](#) on success, Null if requested interface is invalid.

14.35.2.5 GetInterfaceForMyAddress4()

```
int32_t GetInterfaceForMyAddress4 (
    IPADDR4 ip)
```

```
#include <netinterface.h>
```

Returns the Interface Number of the specified network interface IPv4 address.

Parameters

<i>ip</i>	IPv4 network interface address.

Returns

Interface number of the specified IPv4 address

14.35.2.6 GetInterfaceLink()

```
bool GetInterfaceLink (
    int ifn)
```

```
#include <netinterface.h>
```

Returns the network interface link status.

Parameters

<i>ifn</i>	Network interface number

Returns

True if link, otherwise false

See also

[InterfaceBlock::LinkActive\(\)](#), [InterfaceLinkActive\(\)](#)

14.35.2.7 GetInterfaceNumber()

```
int32_t GetInterfaceNumber (
    InterfaceBlock * pifb)
```

```
#include <netinterface.h>
```

Returns the Interface Number of the specified network interface [InterfaceBlock](#).

Parameters

<i>pifb</i>	Pointer to the InterfaceBlock

Returns

Interface number of the [InterfaceBlock](#).

See also

[InterfaceBlock::GetInterfaceNumber\(\)](#)

14.35.2.8 GetNextInterface()

```
int32_t GetNextInterface (
    int lastInterface)
#include <netinterface.h>
```

Returns the Interface Number of the next registered network interface.

Parameters

<i>lastInterface</i>	Specifies the interface number starting point to begin search.

Returns

Interface number of the next network interface, or 0 if none exist.

14.35.2.9 InterfaceLinkActive()

```
bool InterfaceLinkActive (
    int interface)
#include <netinterface.h>
```

Returns the link status of the specified network interface.

Parameters

<i>interface</i>	Interface number

Returns

True if link is active, otherwise false

See also

[InterfaceBlock::LinkActive\(\)](#), [GetInterfaceLink\(\)](#)

14.35.2.10 InterfaceLinkDuplex()

```
bool InterfaceLinkDuplex (
    int interface)
#include <netinterface.h>
```

Returns the full or half link duplex of the specified network interface.

Parameters

<i>interface</i>	Interface number

Returns

True if full duplex, false if half duplex

See also

[InterfaceBlock::LinkDuplex\(\)](#)

14.35.2.11 InterfaceLinkSpeed()

```
int InterfaceLinkSpeed (
    int interface)
#include <netinterface.h>
```

Returns the 10/100 link speed of the specified network interface.

Parameters

<i>interface</i>	Interface number

Returns

10 or 100 representing the Mbps speed

See also

[InterfaceBlock::LinkSpeed\(\)](#)

14.35.2.12 InterfaceMAC()

```
MACADDR InterfaceMAC (  
    int interface)  
#include <netinterface.h>  
Returns the MAC address of the specified network interface.
```

Parameters

<i>interface</i>	Interface number

Returns

MAC address of type [MACADDR](#)

14.35.2.13 InterfaceName()

```
const char * InterfaceName (  
    int interface)  
#include <netinterface.h>  
Returns the interface name of the specified network interface.
```

Parameters

<i>interface</i>	Interface number

Returns

Pointer to a character string containing the name of the network interface

See also

[InterfaceBlock::GetInterfaceName\(\)](#)

14.35.2.14 Removeinterface()

```
int Removeinterface (  
    int interface)  
#include <netinterface.h>  
Remove a network interface from the system.
```

Parameters

<i>interface</i>	Interface number

Returns

1 if successful, 0 if invalid interface number

See also

[InterfaceBlock::RegisterInterface\(\)](#)

14.35.3 IPADDR4 Functions

Network Interface functions with [IPADDR4](#) return types.

Functions

- [IPADDR4 InterfaceIP](#) (int interface)
Returns the IPv4 IP address of the specified network interface.
- [IPADDR4 InterfaceAutoIP](#) (int interface)
Returns the IPv4 IP AutoIP address of the specified network interface.
- [IPADDR4 InterfaceDNS](#) (int interface)
Returns the IPv4 DNS address of the specified network interface.
- [IPADDR4 InterfaceDNS2](#) (int interface)
Returns the second IPv4 DNS address of the specified network interface.
- [IPADDR4 InterfaceMASK](#) (int interface)
Returns the IPv4 network mask of the specified network interface.
- [IPADDR4 InterfaceGate](#) (int interface)
Returns the IPv4 gateway address of the specified network interface.
- bool [HaveActiveNetwork](#) (int interface=-1)
Returns the network routability of the specified network interface.

14.35.3.1 Detailed Description

Network Interface functions with [IPADDR4](#) return types.

```
#include< netinterface.h>
```

Note

While the recommended method for getting and setting network interface parameters is to obtain a pointer to an [InterfaceBlock](#), there are also non-member functions of the object that can be used for situations that are best served by a single function call, such as just getting a single IP address of an interface. When using those functions pay special attention to the return value type. While IPADDR should be used because it supports both [IPADDR4](#) and [IPADDR6](#) types automatically, some of these helper functions may specifically return an [IPADDR4](#) type.

14.35.3.2 Function Documentation**14.35.3.2.1 HaveActiveNetwork()**

```
bool HaveActiveNetwork (
    int interface = -1)
#include <netinterface.h>
```

Returns the network routability of the specified network interface.

Parameters

<i>interface</i>	Interface number, -1 for Any Interface

Returns

True if interface is routable, otherwise false

See also

[InterfaceBlock::LinkActive\(\)](#), [GetInterfaceLink\(\)](#)

14.35.3.2.2 InterfaceAutoIP()

```
IPADDR4 InterfaceAutoIP (  
    int interface)  
#include <netinterface.h>
```

Returns the IPv4 IP AutoIP address of the specified network interface.

Parameters

<i>interface</i>	Interface number

Returns

IPv4 IP address of type [IPADDR4](#)

14.35.3.2.3 InterfaceDNS()

```
IPADDR4 InterfaceDNS (  
    int interface)  
#include <netinterface.h>
```

Returns the IPv4 DNS address of the specified network interface.

Parameters

<i>interface</i>	Interface number

Returns

IPv4 IP address of type [IPADDR4](#)

14.35.3.2.4 InterfaceDNS2()

```
IPADDR4 InterfaceDNS2 (  
    int interface)  
#include <netinterface.h>
```

Returns the second IPv4 DNS address of the specified network interface.

Parameters

<i>interface</i>	Interface number

Returns

IPv4 IP address of type [IPADDR4](#)

14.35.3.2.5 InterfaceGate()

```
IPADDR4 InterfaceGate (  
    int interface)  
#include <netinterface.h>
```

Returns the IPv4 gateway address of the specified network interface.

Parameters

<i>interface</i>	Interface number

Returns

IPv4 IP address of type [IPADDR4](#)

14.35.3.2.6 InterfaceIP()

```
IPADDR4 InterfaceIP (
    int interface)
#include <netinterface.h>
```

Returns the IPv4 IP address of the specified network interface.

Parameters

<i>interface</i>	Interface number

Returns

IPv4 IP address of type [IPADDR4](#)

14.35.3.2.7 InterfaceMASK()

```
IPADDR4 InterfaceMASK (
    int interface)
#include <netinterface.h>
```

Returns the IPv4 network mask of the specified network interface.

Parameters

<i>interface</i>	Interface number

Returns

IPv4 network mask of type [IPADDR4](#)

14.35.4 Low Level Processing (NetDoRx)

Callback functions to add custom Ethernet handlers and pass Ethernet frames at the bottom of the TCP/IP stack.

Typedefs

- typedef int(* [netDoRXFunc](#)) ([PoolPtr](#) pp, uint16_t ocount, int if_num)
Function pointer type for custom network receive handlers.

Functions

- [netDoRXFunc SetCustomNetDoRX](#) ([netDoRXFunc](#) customFunc)
Register a custom Ethernet frame handler.
- [netDoRXFunc ClearCustomNetDoRX](#) ()
Remove the currently registered custom Ethernet handler.
- int [NetDoRX](#) ([PoolPtr](#) pp, uint16_t ocount, int if_num)
Main entry point for Ethernet frames into the TCP/IP stack.

Variables

- [netDoRXFunc CustomNetDoRX](#)

Global pointer to the currently registered custom receive handler.

14.35.4.1 Detailed Description

Callback functions to add custom Ethernet handlers and pass Ethernet frames at the bottom of the TCP/IP stack.

```
#include <netrx.h>
```

This module provides the infrastructure for inserting custom Ethernet frame handlers into the network receive path. Custom handlers run before the standard TCP/IP processing and can filter, modify, or consume frames.

Common use cases include:

- Custom protocol implementation
- Frame filtering or monitoring
- Network statistics collection
- Protocol bridging or tunneling

Note

Custom handlers require that the system is (re)compiled with the `ALLOW_CUSTOM_NET_DO_RX` macro defined.

Warning

These functions are for advanced developers only! Code written can affect all receive network processing and must handle frames efficiently to avoid packet loss or network degradation.

14.35.4.2 Typedef Documentation

14.35.4.2.1 netDoRXFunc

```
typedef int (* netDoRXFunc) (PoolPtr pp, uint16_t ocount, int if_num)
#include <netrx.h>
```

Function pointer type for custom network receive handlers.

This typedef defines the signature for custom Ethernet frame processing functions. Functions of this type are called for each received Ethernet frame before standard TCP/IP processing occurs.

The handler function should:

- Examine the frame data in the buffer
- Decide whether to process or pass the frame
- Return 1 if the frame was consumed (will be released)
- Return 0 if the frame should continue to standard processing

Parameters

<i>pp</i>	Pointer to PoolBuffer containing the received Ethernet frame. The buffer contains the complete frame including Ethernet header.
<i>ocount</i>	Total number of octets (bytes) in the received frame, including all headers and payload.
<i>if_num</i>	Network interface index that received the frame. Zero-based index identifying which physical interface the frame arrived on.

Returns

Status indicating whether the frame was handled:

- 1: Frame was processed and consumed by the handler. The buffer will be released and no further processing will occur.
- 0: Frame was not processed. Continue with standard TCP/IP stack processing.

Note

The handler must not modify the buffer or its contents unless it returns 1 to indicate it has consumed the frame.

Warning

Performance critical! This function is called for EVERY received frame. Inefficient implementations can cause packet loss and network degradation.

```
// Example: Custom handler that monitors ARP frames
int MyArpMonitor(PoolPtr pp, uint16_t ocount, int if_num) {
    // Get pointer to Ethernet header
    uint8_t *frame = (uint8_t *)pp->pData;

    // Check if this is an ARP frame (EtherType 0x0806)
    if (ocount >= 14 && frame[12] == 0x08 && frame[13] == 0x06) {
        // Log ARP frame (but don't consume it)
        iprintf("ARP frame received on interface %d\r\n", if_num);
    }

    // Don't consume - let stack process normally
    return 0;
}
```

14.35.4.3 Function Documentation**14.35.4.3.1 ClearCustomNetDoRX()**

```
netDoRXFunc ClearCustomNetDoRX () [inline]
#include <netrx.h>
```

Remove the currently registered custom Ethernet handler.

Clears the custom Ethernet frame handler by setting it to NULL. After calling this function, all received frames will go directly to standard TCP/IP processing without any custom pre-processing.

This is a convenience function equivalent to calling SetCustomNetDoRX(NULL).

Returns

Pointer to the previously registered custom handler, or NULL if no handler was registered. The returned pointer can be saved to restore the handler later if needed.

Note

This function is thread-safe and can be called at any time.

```
// Example: Temporarily disabling a handler
netDoRXFunc savedHandler = ClearCustomNetDoRX();

// ... do something without custom handler ...

// Restore the handler
if (savedHandler != NULL) {
    SetCustomNetDoRX(savedHandler);
}
```

14.35.4.3.2 NetDoRX()

```
int NetDoRX (
    PoolPtr pp,
    uint16_t ocount,
    int if_num)
#include <netrx.h>
```

Main entry point for Ethernet frames into the TCP/IP stack.

This function is the primary entry point for all received Ethernet frames into the NetBurner TCP/IP stack. It is called by the network driver when a frame has been received and is ready for processing.

The function performs the following operations:

1. If a custom handler is registered, calls it first
2. If custom handler returns 1, releases the buffer and returns
3. Otherwise, processes the frame through standard TCP/IP layers:
 - Ethernet layer processing
 - IP layer processing (if IP frame)
 - Transport layer processing (TCP/UDP/ICMP)
 - Application layer delivery

Parameters

<i>pp</i>	Pointer to PoolBuffer containing the complete Ethernet frame. The buffer includes the Ethernet header and all subsequent data. The buffer is released by this function unless an error occurs.
<i>ocount</i>	Number of octets (bytes) in the Ethernet frame, including the Ethernet header. This is the total frame size as received from the network interface.
<i>if_num</i>	Zero-based index of the network interface that received the frame. This identifies which physical network interface (Ethernet port) the frame arrived on. Used for multi-interface routing decisions.

Returns

The number of system timer ticks to wait before this function should be called again for additional processing. A return value of 0 indicates immediate readiness for more frames. This is primarily used internally by the network driver.

Note

This function is typically called only by network driver code and should not be called directly by application code.

The PoolPtr buffer is managed internally and will be released after processing unless an error occurs.

Warning

Do not call this function from application code unless implementing a custom network driver or performing specialized network operations.

14.35.4.3.3 SetCustomNetDoRX()

```
netDoRXFunc SetCustomNetDoRX (
    netDoRXFunc customFunc) [inline]
#include <netrx.h>
```

Register a custom Ethernet frame handler.

Installs a custom handler function that will be called for every received Ethernet frame before standard TCP/IP processing. Only one custom handler can be active at a time; calling this function replaces any previously registered handler.

The custom handler is invoked in the network receive interrupt context and should:

- Execute quickly to avoid blocking other network processing
- Be thread-safe if accessing shared resources
- Return 1 only if completely handling/consuming the frame
- Return 0 to pass the frame to standard processing

Parameters

<i>customFunc</i>	Pointer to the custom netDoRXFunc handler to register. Pass NULL to clear the custom handler.

Returns

Pointer to the previously registered custom handler, or NULL if no handler was previously registered. This allows chaining handlers or restoring the previous handler later.

Note

This function is thread-safe and can be called at any time.

Warning

The handler runs in interrupt context with network interrupts enabled. It must be efficient and non-blocking.

```
// Example: Installing and later restoring a handler
int MyHandler(PoolPtr pp, uint16_t ocount, int if_num) {
    // Process frame...
    return 0;
}

void InstallHandler() {
    // Save previous handler
    netDoRXFunc oldHandler = SetCustomNetDoRX(MyHandler);

    // ... later, restore previous handler ...
    SetCustomNetDoRX(oldHandler);
}
```

14.35.4.4 Variable Documentation**14.35.4.4.1 CustomNetDoRX**

```
netDoRXFunc CustomNetDoRX [extern]
#include <netrx.h>
```

Global pointer to the currently registered custom receive handler.

This external variable holds the function pointer to the active custom network receive handler. If NULL, no custom handler is registered and all frames go directly to standard TCP/IP processing.

Note

Do not modify this variable directly. Use [SetCustomNetDoRX\(\)](#) and [ClearCustomNetDoRX\(\)](#) for proper handler registration.

14.36 PEM Conversion Utilities

Privacy-Enhanced Mail (PEM) format conversion and endianness utilities.

Topics

- [PEM Data Type Identifiers](#)
Return values indicating the type of PEM data that was converted.
- [PEM Format Footers](#)
Standard PEM footer markers for different data types.
- [PEM Format Headers](#)
Standard PEM header markers for different data types.

Macros

- `#define` [CONVERT_BUFFER_LENGTH_MAX](#) `((4 * 1024) - 1)`
Maximum data buffer length in bytes for PEM conversion.

Functions

- int [ConvertPEMFormattedData](#) (puint8_t dataPtr, const char *pemEncodedData, uint32_t dataSize, puint32_t convertedDataLength, char **nextPtr=NULL)
Convert data from Privacy-Enhanced Mail (PEM) format to binary DER format.
- uint16_t [convertLittleEndianWord](#) (uint16_t hData)
Convert 16-bit data between host byte order and little-endian format.
- uint32_t [convertLittleEndianDword](#) (uint32_t hData)
Convert 32-bit data between host byte order and little-endian format.

14.36.1 Detailed Description

Privacy-Enhanced Mail (PEM) format conversion and endianness utilities.

```
#include< convert.h>
```

This module provides functions for converting between PEM-encoded cryptographic data and binary formats, as well as byte-order conversion utilities. PEM format is commonly used for storing and transmitting cryptographic keys and certificates in a text-based, Base64-encoded format.

14.36.1.1 Supported PEM Types

- X.509 Certificates
- RSA Private Keys
- DSA Private Keys
- ECDSA (Elliptic Curve) Private Keys

14.36.1.2 Typical Use Cases

- Loading SSL/TLS certificates and private keys
- Converting keys from PEM files for use with cryptographic libraries
- Preparing certificates for secure network communications
- Byte-order conversion for cross-platform data exchange

Note

- All PEM data must be NULL-terminated strings.
- Buffer sizes should be at least CONVERT_BUFFER_LENGTH_MAX bytes.

14.36.2 Macro Definition Documentation

14.36.2.1 CONVERT_BUFFER_LENGTH_MAX

```
#define CONVERT_BUFFER_LENGTH_MAX ((4 * 1024) - 1)
#include <convert.h>
```

Maximum data buffer length in bytes for PEM conversion.

This defines the maximum size of the output buffer needed for PEM conversion operations. The buffer must be at least this size to safely convert PEM data.

Value: 4095 bytes (4KB - 1)

Note

This is a practical limit based on typical certificate and key sizes. Most X.509 certificates and private keys are well under 4KB.

See also

[ConvertPEMFormattedData\(\)](#)

14.36.3 Function Documentation

14.36.3.1 convertLittleEndianDword()

```
uint32_t convertLittleEndianDword (
    uint32_t hData)
```

```
#include <convert.h>
```

Convert 32-bit data between host byte order and little-endian format.

Converts a 32-bit value to or from little-endian byte order. This function is bidirectional - it performs the same operation regardless of conversion direction. On little-endian systems (x86, ARM in little-endian mode), this is a no-op. On big-endian systems (ColdFire, PowerPC), it swaps the byte order.

14.36.3.2 Byte Order Formats

- **Little-Endian:** Least significant byte first (e.g., 0x12345678 -> [0x78, 0x56, 0x34, 0x12])
- **Big-Endian:** Most significant byte first (e.g., 0x12345678 -> [0x12, 0x34, 0x56, 0x78])

Parameters

<i>hData</i>	32-bit value to convert. Can be in either host or little-endian format.

Returns

32-bit value in the opposite byte order from input:

- If input is host order: returns little-endian
- If input is little-endian: returns host order

Note

This function is its own inverse: calling it twice returns the original value.

On little-endian platforms, this function may be optimized to a no-op.

Commonly used for reading/writing file formats that specify little-endian encoding.

See also

[convertLittleEndianWord\(\)](#) for 16-bit conversion

Expand for Example Usage

Examples

Example 1: File header with little-endian fields

```
struct FileHeader {
    uint32_t signature;    // Little-endian
    uint32_t fileSize;    // Little-endian
    uint32_t dataOffset;  // Little-endian
};

void WriteFileHeader(FILE *fp, uint32_t size, uint32_t offset) {
    FileHeader header;

    // Convert to little-endian for file storage
    header.signature = convertLittleEndianDword(0x12345678);
    header.fileSize = convertLittleEndianDword(size);
    header.dataOffset = convertLittleEndianDword(offset);

    fwrite(&header, sizeof(header), 1, fp);
    fprintf("Wrote header: size=%lu, offset=%lu\r\n", size, offset);
}

bool ReadFileHeader(FILE *fp, FileHeader *header) {
    if (fread(header, sizeof(FileHeader), 1, fp) != 1) {
        return false;
    }

    // Convert from little-endian to host
    header->signature = convertLittleEndianDword(header->signature);
    header->fileSize = convertLittleEndianDword(header->fileSize);
    header->dataOffset = convertLittleEndianDword(header->dataOffset);

    return true;
}
```


Example 2: Cross-platform data exchange

```
// Write timestamp in little-endian format
void SaveTimestamp(FILE *fp) {
    uint32_t timestamp = (uint32_t)time(NULL);
    uint32_t leTimestamp = convertLittleEndianDword(timestamp);
    fwrite(&leTimestamp, sizeof(leTimestamp), 1, fp);
}

// Read timestamp from little-endian format
time_t LoadTimestamp(FILE *fp) {
    uint32_t leTimestamp;
    fread(&leTimestamp, sizeof(leTimestamp), 1, fp);
    uint32_t timestamp = convertLittleEndianDword(leTimestamp);
    return (time_t)timestamp;
}
```

Example 3: Binary data structure conversion

```
struct SensorData {
    uint32_t timestamp;
    uint32_t sensorId;
    uint16_t temperature;
    uint16_t humidity;
};

// Convert structure to little-endian for storage/transmission
void ConvertToLittleEndian(SensorData *data) {
    data->timestamp = convertLittleEndianDword(data->timestamp);
    data->sensorId = convertLittleEndianDword(data->sensorId);
    data->temperature = convertLittleEndianWord(data->temperature);
    data->humidity = convertLittleEndianWord(data->humidity);
}

// Convert structure from little-endian to host order
void ConvertFromLittleEndian(SensorData *data) {
    data->timestamp = convertLittleEndianDword(data->timestamp);
    data->sensorId = convertLittleEndianDword(data->sensorId);
    data->temperature = convertLittleEndianWord(data->temperature);
    data->humidity = convertLittleEndianWord(data->humidity);
}
```

14.36.3.3 convertLittleEndianWord()

```
uint16_t convertLittleEndianWord (
    uint16_t hData)
```

```
#include <convert.h>
```

Convert 16-bit data between host byte order and little-endian format.

Converts a 16-bit value to or from little-endian byte order. This function is bidirectional - it performs the same operation regardless of conversion direction. On little-endian systems (x86, ARM in little-endian mode), this is a no-op. On big-endian systems (ColdFire, PowerPC), it swaps the byte order.

14.36.3.4 Byte Order Formats

- **Little-Endian:** Least significant byte first (e.g., 0x1234 -> [0x34, 0x12])
- **Big-Endian:** Most significant byte first (e.g., 0x1234 -> [0x12, 0x34])

Parameters

<i>hData</i>	16-bit value to convert. Can be in either host or little-endian format.

Returns

16-bit value in the opposite byte order from input:

- If input is host order: returns little-endian
- If input is little-endian: returns host order

Note

This function is its own inverse: calling it twice returns the original value.

On little-endian platforms, this function may be optimized to a no-op.

Commonly used for reading/writing file formats that specify little-endian encoding.

See also

[convertLittleEndianDword\(\)](#) for 32-bit conversion

Expand for Example Usage**Examples****Example 1: Converting data for file I/O**

```
// Writing data in little-endian format to file
uint16_t hostValue = 0x1234;
uint16_t fileValue = convertLittleEndianWord(hostValue);
fwrite(&fileValue, sizeof(fileValue), 1, fp);

// Reading data from little-endian file
uint16_t readValue;
fread(&readValue, sizeof(readValue), 1, fp);
uint16_t hostValue2 = convertLittleEndianWord(readValue);

iprintf("Original: 0x%04X, Converted: 0x%04X, Read back: 0x%04X\r\n",
        hostValue, fileValue, hostValue2);
```

Example 2: Network protocol with mixed endianness

```
struct LittleEndianPacket {
    uint16_t length; // Little-endian
    uint16_t checksum; // Little-endian
    uint8_t data[256];
};

void SendPacket(const uint8_t *data, uint16_t len) {
    LittleEndianPacket packet;

    // Convert to little-endian for transmission
    packet.length = convertLittleEndianWord(len);

    memcpy(packet.data, data, len);

    uint16_t chk = CalculateChecksum(data, len);
    packet.checksum = convertLittleEndianWord(chk);

    send(&packet, sizeof(packet));
}
```

14.36.3.5 ConvertPEMFormattedData()

```
int ConvertPEMFormattedData (
    uint8_t *dataPtr,
    const char * pemEncodedData,
    uint32_t dataSize,
    uint32_t convertedDataLength,
    char ** nextPtr = NULL)
```

#include <convert.h>

Convert data from Privacy-Enhanced Mail (PEM) format to binary DER format.

This function converts PEM-encoded cryptographic data (certificates or private keys) from Base64-encoded text format to binary DER (Distinguished Encoding Rules) format. It automatically detects the type of PEM data based on the header/footer markers.

The function performs:

1. PEM header/footer validation
2. Base64 decoding of the content between header and footer
3. Binary DER format output
4. Type identification

14.36.3.6 PEM Format Structure

```
-----BEGIN [TYPE]-----
[Base64 Encoded Data]
-----END [TYPE]-----
```

Parameters

out	<i>dataPtr</i>	Pointer to buffer for converted binary data. Must be at least <code>CONVERT_BUFFER_LENGTH_MAX</code> bytes. The buffer will contain the DER-encoded binary representation of the key or certificate.
in	<i>pemEncodedData</i>	Pointer to NULL-terminated string containing PEM-encoded data. May contain multiple PEM objects; function will process the first valid one found.
in	<i>dataSize</i>	Size of the output buffer pointed to by <i>dataPtr</i> , in bytes. Should be at least <code>CONVERT_BUFFER_LENGTH_MAX</code> .
out	<i>convertedDataLength</i>	Pointer to <code>uint32_t</code> where the actual length of converted data will be stored. This value will be less than or equal to <i>dataSize</i> .
out	<i>nextPtr</i>	Optional pointer to char pointer. If provided and not NULL, will be set to point to the character immediately following the converted PEM data. Useful for processing multiple PEM objects in a single string. Pass NULL if not needed.

Returns

PEM data type identifier:

Return values

<code>CONVERT_NONE</code>	Malformed data, calling parameters invalid, or conversion failed
<code>CONVERT_CERTIFICATE</code>	Successfully converted X.509 certificate
<code>CONVERT_RSA_PRIVATE_KEY</code>	Successfully converted RSA private key
<code>CONVERT_DSA_PRIVATE_KEY</code>	Successfully converted DSA private key
<code>CONVERT_ECDSA_PRIVATE_KEY</code>	Successfully converted ECDSA private key

Notes:

- The *pemEncodedData* string must be NULL-terminated.
- The output buffer (*dataPtr*) must be large enough to hold the decoded data.
- The function only processes one PEM object per call; use *nextPtr* to process multiple objects.
- The converted data is in binary DER format, suitable for use with SSL/TLS libraries and cryptographic functions.

Warnings:

- Private keys should be handled securely. Consider:
 - Zeroing buffers after use
 - Limiting key exposure in memory
 - Using secure storage when possible
- Ensure adequate buffer size to prevent overflow. If *dataSize* is too small, the function will return `CONVERT_↵_NONE`.

Expand for Example Usage

Examples

Example 1: Converting a single certificate

```

const char *pemCert =
    "-----BEGIN CERTIFICATE-----\n"
    "MIIDXTCCAkWgAwIBAgIJAKJ9VJN8ER5jMA0GCSqGSIb3DQEBCwUAMEUxCzAJBgNV\n"
    "BAYTAkFVMRMwEQYDVQIDApTb211LVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5ldCBX\n"
    "aWRnaXRzIFB0eSBMdGQwHhcNMTgwMzEyMTkyODUyWhcNMTkwMzEyMTkyODUyWjBF\n"
    "MQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwYSW50\n"
    "ZXJuZXQgV2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB\n"
    "CgKCAQEAw5TLyD3m...\n"
    "-----END CERTIFICATE-----\n";

uint8_t binaryData[CONVERT_BUFFER_LENGTH_MAX];
uint32_t dataLength;

int result = ConvertPEMFormattedData(binaryData, pemCert,
                                     sizeof(binaryData), &dataLength);

if (result == CONVERT_CERTIFICATE) {
    printf("Certificate converted successfully\r\n");
    printf("Binary data length: %lu bytes\r\n", dataLength);
    // Use binaryData with SSL/TLS functions
} else {
    printf("Failed to convert certificate\r\n");
}

```

Example 2: Converting an RSA private key

```

const char *pemKey =
    "-----BEGIN RSA PRIVATE KEY-----\n"
    "MIIEpAIBAAKCAQEAw5TLyD3m8F7xWQKLp6F5xoTUlqVJYALZ+qYQVQM8FnDtqZKr\n"
    "... \n"
    "-----END RSA PRIVATE KEY-----\n";

uint8_t keyData[CONVERT_BUFFER_LENGTH_MAX];
uint32_t keyLength;

int result = ConvertPEMFormattedData(keyData, pemKey,
                                     sizeof(keyData), &keyLength);

if (result == CONVERT_RSA_PRIVATE_KEY) {
    printf("RSA key converted: %lu bytes\r\n", keyLength);

    // Use the key with crypto library
    // ...

    // Security: Zero the key buffer when done
    memset(keyData, 0, sizeof(keyData));
} else {
    printf("Failed to convert RSA key\r\n");
}

```

Example 3: Processing multiple PEM objects

```

const char *multiPEM =
    "-----BEGIN CERTIFICATE-----\n"
    "... \n"
    "-----END CERTIFICATE-----\n"
    "-----BEGIN RSA PRIVATE KEY-----\n"
    "... \n"
    "-----END RSA PRIVATE KEY-----\n";

uint8_t buffer[CONVERT_BUFFER_LENGTH_MAX];
uint32_t length;
char *nextPEM = (char*)multiPEM;
int objectCount = 0;

while (nextPEM && *nextPEM) {
    int type = ConvertPEMFormattedData(buffer, nextPEM,
                                       sizeof(buffer), &length, &nextPEM);

    if (type == CONVERT_NONE) {
        break; // No more valid PEM data
    }

    objectCount++;

    switch (type) {
        case CONVERT_CERTIFICATE:
            printf("Object %d: Certificate (%lu bytes)\r\n",
                  objectCount, length);
            break;
        case CONVERT_RSA_PRIVATE_KEY:
            printf("Object %d: RSA Key (%lu bytes)\r\n",
                  objectCount, length);
            break;
        case CONVERT_ECDSA_PRIVATE_KEY:

```

```

        printf("Object %d: ECDSA Key (%lu bytes)\r\n",
               objectCount, length);
        break;
    }
}

printf("Processed %d PEM objects\r\n", objectCount);

```

Example 4: Loading certificate and key for SSL/TLS

```

bool LoadSSLCredentials(const char *certPEM, const char *keyPEM) {
    uint8_t certData[CONVERT_BUFFER_LENGTH_MAX];
    uint8_t keyData[CONVERT_BUFFER_LENGTH_MAX];
    uint32_t certLen, keyLen;

    // Convert certificate
    int certType = ConvertPEMFormattedData(certData, certPEM,
                                           sizeof(certData), &certLen);

    if (certType != CONVERT_CERTIFICATE) {
        printf("Error: Invalid certificate format\r\n");
        return false;
    }

    // Convert private key
    int keyType = ConvertPEMFormattedData(keyData, keyPEM,
                                           sizeof(keyData), &keyLen);

    if (keyType != CONVERT_RSA_PRIVATE_KEY &&
        keyType != CONVERT_ECDSA_PRIVATE_KEY) {
        printf("Error: Invalid private key format\r\n");
        return false;
    }

    // Load into SSL context
    if (!SSL_LoadCertificate(certData, certLen)) {
        printf("Failed to load certificate\r\n");
        memset(keyData, 0, sizeof(keyData)); // Clear sensitive data
        return false;
    }

    if (!SSL_LoadPrivateKey(keyData, keyLen)) {
        printf("Failed to load private key\r\n");
        memset(keyData, 0, sizeof(keyData));
        return false;
    }

    // Security: Clear sensitive key data
    memset(keyData, 0, sizeof(keyData));

    printf("SSL credentials loaded successfully\r\n");
    return true;
}

```

Example 5: Error handling and validation

```

int SafeConvertPEM(const char *pem, uint8_t *output,
                  uint32_t outputSize, uint32_t *outLen) {
    // Validate inputs
    if (!pem || !output || !outLen) {
        printf("Error: NULL pointer\r\n");
        return CONVERT_NONE;
    }

    if (outputSize < CONVERT_BUFFER_LENGTH_MAX) {
        printf("Error: Buffer too small (need %d bytes)\r\n",
               CONVERT_BUFFER_LENGTH_MAX);
        return CONVERT_NONE;
    }

    // Attempt conversion
    int result = ConvertPEMFormattedData(output, pem, outputSize, outLen);

    if (result == CONVERT_NONE) {
        printf("Error: PEM conversion failed\r\n");
        printf(" Check PEM format and headers/footers\r\n");
        return CONVERT_NONE;
    }

    // Validate output length
    if (*outLen == 0 || *outLen > outputSize) {
        printf("Error: Invalid output length: %lu\r\n", *outLen);
        return CONVERT_NONE;
    }

    return result;
}

```

Example 6: Reading PEM data from file

```

#include <stdio.h>

bool ConvertPEMFile(const char *filename) {
    FILE *fp = fopen(filename, "r");
    if (!fp) {
        printf("Error: Cannot open %s\r\n", filename);
        return false;
    }

    // Read entire file into buffer
    char pemBuffer[8192];
    size_t bytesRead = fread(pemBuffer, 1, sizeof(pemBuffer) - 1, fp);
    fclose(fp);

    if (bytesRead == 0) {
        printf("Error: Empty file\r\n");
        return false;
    }

    pemBuffer[bytesRead] = '\0'; // Ensure NULL termination

    // Convert PEM data
    uint8_t binaryData[CONVERT_BUFFER_LENGTH_MAX];
    uint32_t dataLen;

    int type = ConvertPEMFormattedData(binaryData, pemBuffer,
                                       sizeof(binaryData), &dataLen);

    switch (type) {
        case CONVERT_CERTIFICATE:
            printf("Converted certificate from %s\r\n", filename);
            break;
        case CONVERT_RSA_PRIVATE_KEY:
            printf("Converted RSA key from %s\r\n", filename);
            break;
        case CONVERT_ECDSA_PRIVATE_KEY:
            printf("Converted ECDSA key from %s\r\n", filename);
            break;
        default:
            printf("Error: Invalid PEM data in %s\r\n", filename);
            return false;
    }

    return true;
}

```

14.36.4 PEM Data Type Identifiers

Return values indicating the type of PEM data that was converted.

Macros

- #define **CONVERT_NONE** (0)
No valid PEM data found or conversion failed.
- #define **CONVERT_RSA_PRIVATE_KEY** (1)
RSA Private Key in PKCS#1 format.
- #define **CONVERT_DSA_PRIVATE_KEY** (2)
DSA Private Key.
- #define **CONVERT_CERTIFICATE** (3)
X.509 Certificate.
- #define **CONVERT_ECDSA_PRIVATE_KEY** (4)
ECDSA (Elliptic Curve) Private Key.

14.36.4.1 Detailed Description

Return values indicating the type of PEM data that was converted.

14.36.4.2 Macro Definition Documentation

14.36.4.2.1 CONVERT_CERTIFICATE

```
#define CONVERT_CERTIFICATE (3)
#include <convert.h>
```

X.509 Certificate.

Indicates successful conversion of an X.509 certificate. The converted binary data contains the certificate in DER format.

Note

X.509 certificates contain public keys and identity information, signed by a Certificate Authority (CA).

14.36.4.2.2 CONVERT_DSA_PRIVATE_KEY

```
#define CONVERT_DSA_PRIVATE_KEY (2)
#include <convert.h>
```

DSA Private Key.

Indicates successful conversion of a DSA (Digital Signature Algorithm) private key. The converted binary data contains the key in DER format.

Note

DSA keys are used for digital signatures only, not encryption.

14.36.4.2.3 CONVERT_ECDSA_PRIVATE_KEY

```
#define CONVERT_ECDSA_PRIVATE_KEY (4)
#include <convert.h>
```

ECDSA (Elliptic Curve) Private Key.

Indicates successful conversion of an ECDSA private key. The converted binary data contains the key in DER format.

Note

ECDSA provides equivalent security to RSA with smaller key sizes, making it efficient for embedded systems.

14.36.4.2.4 CONVERT_NONE

```
#define CONVERT_NONE (0)
#include <convert.h>
```

No valid PEM data found or conversion failed.

Returned when:

- Input data is malformed or missing PEM headers/footers
- Base64 decoding fails
- Buffer size is insufficient
- Invalid parameters passed to conversion function

14.36.4.2.5 CONVERT_RSA_PRIVATE_KEY

```
#define CONVERT_RSA_PRIVATE_KEY (1)
#include <convert.h>
```

RSA Private Key in PKCS#1 format.

Indicates successful conversion of an RSA private key. The converted binary data contains the key in DER format.

Note

RSA keys are used for asymmetric encryption and digital signatures.

14.36.5 PEM Format Footers

Standard PEM footer markers for different data types.

Macros

- **#define CONVERT_CERTIFICATE_FOOTER** "-----END CERTIFICATE-----"
PEM footer for X.509 certificates.
- **#define CONVERT_RSA_PRIVATE_KEY_FOOTER** "-----END RSA PRIVATE KEY-----"
PEM footer for RSA private keys.
- **#define CONVERT_DSA_PRIVATE_KEY_FOOTER** "-----END DSA PRIVATE KEY-----"
PEM footer for DSA private keys.
- **#define CONVERT_ECDSA_PRIVATE_KEY_FOOTER** "-----END EC PRIVATE KEY-----"
PEM footer for ECDSA private keys.

14.36.5.1 Detailed Description

Standard PEM footer markers for different data types.

14.36.6 PEM Format Headers

Standard PEM header markers for different data types.

Macros

- **#define CONVERT_CERTIFICATE_HEADER** "-----BEGIN CERTIFICATE-----"
PEM header for X.509 certificates.
- **#define CONVERT_RSA_PRIVATE_KEY_HEADER** "-----BEGIN RSA PRIVATE KEY-----"
PEM header for RSA private keys.
- **#define CONVERT_DSA_PRIVATE_KEY_HEADER** "-----BEGIN DSA PRIVATE KEY-----"
PEM header for DSA private keys.
- **#define CONVERT_ECDSA_PRIVATE_KEY_HEADER** "-----BEGIN EC PRIVATE KEY-----"
PEM header for ECDSA private keys.

14.36.6.1 Detailed Description

Standard PEM header markers for different data types.

14.37 POP3 - Post Office Protocol

Topics

- [POP3 Return Codes](#)

Functions

- int [POPGetResultCode](#) (int fd, uint32_t timeout)
Returns the result code of the previous POP3 operation.
- int [POP3_InitializeSession](#) (IPADDR server_address, uint16_t port, PCSTR UserName, PCSTR Password, uint32_t timeout)
Create a connection to the POP3 server and log in.
- int [POP3_CloseSession](#) (int session)
Close a POP3 session.
- int [POP3_StatCmd](#) (int session, uint32_t *num_messages, uint32_t *total_bytes, uint32_t timeout)
Returns the status of the mailstore on the POP3 server.
- int [POP3_ListCmd](#) (int session, uint32_t message_number, uint32_t *total_bytes, uint32_t timeout)

Get the size of a message on the server.

- int [POP3_DeleteCmd](#) (int session, uint32_t message_number, uint32_t timeout)

Delete a pending message on the server.

- int [POP3_RetrieveMessage](#) (int session, uint32_t message_number, char *buffer, char **subject_ptr, char **body_ptr, int max_bufferlen, uint32_t timeout)

Retrieve a message from the server.

- PCSTR [GetPOPErrorString](#) (int err)

Returns the error text for an error code.

- int [SSL_POP3_InitializeSession](#) ([IPADDR](#) server_address, uint16_t port, PCSTR UserName, PCSTR Password, uint32_t time_out)

Initialize Post Office Protocol Version 3 (POP3) on a SSL socket.

14.37.1 Detailed Description

```
#include< pop3.h>
```

14.37.2 Function Documentation

14.37.2.1 GetPOPErrorString()

```
PCSTR GetPOPErrorString (
    int err)
```

```
#include <pop3.h>
```

Returns the error text for an error code.

Parameters

<i>err</i>	Error number to decode

Returns

Pointer to a string containing the error code text

14.37.2.2 POP3_CloseSession()

```
int POP3_CloseSession (
    int session)
```

```
#include <pop3.h>
```

Close a POP3 session.

Parameters

<i>session</i>	The session number returned by POP3_InitializeSession()

Returns

[POP3 Return Codes](#)

14.37.2.3 POP3_DeleteCmd()

```
int POP3_DeleteCmd (  
    int session,  
    uint32_t message_number,  
    uint32_t timeout)
```

```
#include <pop3.h>
```

Delete a pending message on the server.

Note: the server will not actually delete the message until the session is closed.

Parameters

<i>session</i>	The session number returned by POP3_InitializeSession() .
<i>message_number</i>	The message number to delete. The message number can be obtained by POP3_StatCmd() .
<i>timeout</i>	Timeout in system Time Ticks.

Returns

Return code [POP3 Return Codes](#)

14.37.2.4 POP3_InitializeSession()

```
int POP3_InitializeSession (
    IPADDR server_address,
    uint16_t port,
    PCSTR UserName,
    PCSTR Password,
    uint32_t timeout)
#include <pop3.h>
Create a connection to the POP3 server and log in.
```

Parameters

<i>server_address</i>	Server IP address
<i>port</i>	Server port number
<i>UserName</i>	Account username
<i>Password</i>	Account password
<i>timeout</i>	Timeout in system Time Ticks

Returns

The POP3 session number if greater than 0, otherwise error code [POP3 Return Codes](#)

See also

[SSL_POP3_InitializeSession](#)

14.37.2.5 POP3_ListCmd()

```
int POP3_ListCmd (
    int session,
    uint32_t message_number,
    uint32_t * total_bytes,
    uint32_t timeout)
#include <pop3.h>
Get the size of a message on the server.
```

Parameters

<i>session</i>	The session number returned by POP3_InitializeSession() .
<i>message_number</i>	The message number to query. The message number can be obtained by POP3_StatCmd() .
<i>total_bytes</i>	Pointer to the variable in which to store the total number of bytes of the pending message.
<i>timeout</i>	Timeout in system Time Ticks.

Returns

Return code [POP3 Return Codes](#)

14.37.2.6 POP3_RetrieveMessage()

```
int POP3_RetrieveMessage (
    int session,
    uint32_t message_number,
    char * buffer,
    char ** subject_ptr,
    char ** body_ptr,
    int max_bufferlen,
    uint32_t timeout)
#include <pop3.h>
```

Retrieve a message from the server.

The message is retrieved as a block of bytes including the header, subject, etc. The headers will appear first. Note that the message is left on the server and will not be deleted until the [POP3_DeleteCmd\(\)](#) function is called for the message number.

Parameters

<i>session</i>	The session number returned by POP3_InitializeSession() .
<i>message_number</i>	The message number to retrieve. The message number can be obtained by POP3_StatCmd() .
<i>buffer</i>	Pointer to the buffer to store the entire message, including headers
<i>subject_ptr</i>	If not NULL, a pointer to the start of the subject in the message buffer
<i>body_ptr</i>	If not NULL, a pointer to the start of the body in the message buffer
<i>max_bufferlen</i>	Maximum length of the buffer
<i>timeout</i>	Timeout in system Time Ticks.

Returns

Return code [POP3 Return Codes](#)

14.37.2.7 POP3_StatCmd()

```
int POP3_StatCmd (
    int session,
    uint32_t * num_messages,
    uint32_t * total_bytes,
    uint32_t timeout)
#include <pop3.h>
```

Returns the status of the mailstore on the POP3 server.

Parameters

<i>session</i>	The session number returned by POP3_InitializeSession()
<i>num_messages</i>	Pointer to the variable in which to store the number of pending messages
<i>total_bytes</i>	Pointer to the variable in which to store the total number of bytes of the pending messages
<i>timeout</i>	Timeout in system Time Ticks

Returns

Return code [POP3 Return Codes](#)

14.37.2.8 POPGetResultCode()

```
int POPGetResultCode (
    int fd,
    uint32_t timeout)
```

```
#include <pop3.h>
```

Returns the result code of the previous POP3 operation.

Parameters

<i>fd</i>	TCP socket file descriptor for the active session
<i>timeout</i>	Timeout in system Time Ticks

Returns

[POP3 Return Codes](#)

14.37.2.9 SSL_POP3_InitializeSession()

```
int SSL_POP3_InitializeSession (
    IPADDR server_address,
    uint16_t port,
    PCSTR UserName,
    PCSTR PassWord,
    uint32_t time_out)
```

```
#include <ssl_pop3.h>
```

Initialize Post Office Protocol Version 3 (POP3) on a SSL socket.

Note: Routine makes the connection to the POP server and logs in with the provided UserName and PassWord.

Session is release with POP3_CloseSession in [pop3.h](#).

Post Office Protocol, Version 3 (POP3) is defined in RFC1939.

Parameters

<i>server_address</i>	IP address of the server.
<i>port</i>	Port to connect to the server.
<i>UserName</i>	Account user name string.
<i>PassWord</i>	Account password string.
<i>time_out</i>	Number of ticks to wait.

Return values

<i>>=0</i>	The session handle
<i><0</i>	Error code from POP3 Return Codes

See also

[POP3_InitializeSession](#)

14.37.3 POP3 Return Codes**Macros**

- #define **POP_OK** (0)
No errors occurred.
- #define **POP_TIMEOUT** (-1)
Time out.

- **#define POP_PASSWORDERROR** (-2)
Password error.
- **#define POP_CONNECTFAIL** (-3)
Connection failed.
- **#define POP_COMMANDFAIL** (-4)
Command failed.
- **#define POP_BADSESSION** (-5)
Session error.
- **#define POP_NETWORKERROR** (-6)
Network error.
- **#define POP_BUFFER_FULL** (-7)
Receive buffer full.

14.37.3.1 Detailed Description

```
#include< pop3.h>
```

14.38 PPP - Point to Point Protocol

Topics

- [PPP Connection State](#)
- [PPP Return Codes](#)

14.38.1 Detailed Description

```
#include< ppp.h>
```

Like Ethernet, Point to Point Protocol (PPP) is a data link layer which transfers data between nodes on a network. The main difference is that PPP is used between two end points, whereas Ethernet is a multidrop configuration. PPP can be used over many types of physical networks, including: fiber, phone line, serial cable and cellular telephone. Once you have created a PPP connection to or from your NetBurner device you can use it with all the standard network applications such as the web server, tcp, udp, ftp, email, etc. Internet Service Providers (ISP) can implement PPP with many different options that vary from ISP to ISP. The PPP example program included in the development kit examples directory specifies the most common ISP options and can be used as a code example to specify any specific options you may need in your application.

14.38.2 PPP Connection State

Enumerations

- enum [enum_PPPState](#) {
[eClosed](#) , [eInitializingModem](#) , [eDialing](#) , [eWait4Ring](#) ,
[eAnswering](#) , [eWaitForTrain](#) , [eLCPNegotiate](#) , [ePAPAuthenticate](#) ,
[eCHAPAuthenticate](#) , [eNCPNegotiate](#) , [eOpen](#) , [eClosing](#) }
PPP States.

14.38.2.1 Detailed Description

```
#include< ppp.h>
```

14.38.2.2 Enumeration Type Documentation

14.38.2.2.1 enum_PPPState

```
enum enum_PPPState
#include <ppp.h>
PPP States.
```

Enumerator

eClosed	Connection closed.
eInitializingModem	Initializing modem.
eDialing	Dialing.
eWait4Ring	Waiting for ring indicator.
eAnswering	Answering incoming connection.
eWaitForTrain	Waiting for train.
eLCPNegotiate	LCP negotiation.
ePAPAuthenticate	PAP authentication.
eCHAPAuthenticate	CHAP authentication.
eNCPNegotiate	NCP negotiation.
eOpen	Connection open.
eClosing	Connection closing.

14.38.3 PPP Return Codes

Macros

- #define **ERR_PPP_SUCCESS** (0)
Success.
- #define **ERR_PPP_ALREADY_OPEN** (-1)
A session is already open.
- #define **ERR_PPP_NO_DIALTONE** (-2)
No dial tone.
- #define **ERR_PPP_NO_ANSWER** (-3)
The remote client did not answer.
- #define **ERR_PPP_BUSY** (-4)
The remote client is sending a busy signal.
- #define **ERR_PPP_FAIL** (-5)
The attempted action has failed.
- #define **ERR_PPP_PASSFAIL** (-6)
Pass/Fail.
- #define **ERR_PPP_LOSTCARRIER** (-7)
Lost connection carrier signal.
- #define **ERR_PPP_NO_MODEM** (-8)
No modem detected.
- #define **ERR_PPP_LCP_FAILED** (-9)
LCP negotiation has failed.
- #define **ERR_PPP_CHAPFAIL** (-10)
CHAP negotiation has failed.

14.38.3.1 Detailed Description

```
#include< ppp.h>
```

14.39 Pit

Classes

- struct [_pit_config](#)
PIT configuration structure.

Typedefs

- typedef enum [_pit_chnl](#) [pit_chnl_t](#)
List of PIT channels.
- typedef enum [_pit_interrupt_enable](#) [pit_interrupt_enable_t](#)
List of PIT interrupts.
- typedef enum [_pit_status_flags](#) [pit_status_flags_t](#)
List of PIT status flags.
- typedef struct [_pit_config](#) [pit_config_t](#)
PIT configuration structure.

Enumerations

- enum [_pit_chnl](#) { [kPIT_Chnl_0](#) = 0U , [kPIT_Chnl_1](#) , [kPIT_Chnl_2](#) , [kPIT_Chnl_3](#) }
List of PIT channels.
- enum [_pit_interrupt_enable](#) { [kPIT_TimerInterruptEnable](#) = PIT_TCTRL_TIE_MASK }
List of PIT interrupts.
- enum [_pit_status_flags](#) { [kPIT_TimerFlag](#) = PIT_TFLG_TIF_MASK }
List of PIT status flags.

Initialization and deinitialization

- void [PIT_Init](#) (PIT_Type *base, const [pit_config_t](#) *config)
Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.
- void [PIT_Deinit](#) (PIT_Type *base)
Gates the PIT clock and disables the PIT module.

Driver version

- #define [FSL_PIT_DRIVER_VERSION](#) (MAKE_VERSION(2, 0, 5))
PIT Driver Version 2.0.5.

14.39.1 Detailed Description

14.39.2 Typedef Documentation

14.39.2.1 pit_chnl_t

```
typedef enum _pit_chnl pit_chnl_t
#include <fsl_pit.h>
List of PIT channels.
```


Note

Actual number of available channels is SoC dependent

14.39.2.2 pit_config_t

```
typedef struct _pit_config pit_config_t
#include <fsl_pit.h>
```

PIT configuration structure.

This structure holds the configuration settings for the PIT peripheral. To initialize this structure to reasonable defaults, call the PIT_GetDefaultConfig() function and pass a pointer to your config structure instance.

The configuration structure can be made constant so it resides in flash.

14.39.3 Enumeration Type Documentation**14.39.3.1 _pit_chnl**

```
enum _pit_chnl
#include <fsl_pit.h>
```

List of PIT channels.

Note

Actual number of available channels is SoC dependent

Enumerator

kPIT_Chnl↔ _0	PIT channel number 0
kPIT_Chnl↔ _1	PIT channel number 1
kPIT_Chnl↔ _2	PIT channel number 2
kPIT_Chnl↔ _3	PIT channel number 3

14.39.3.2 _pit_interrupt_enable

```
enum _pit_interrupt_enable
#include <fsl_pit.h>
```

List of PIT interrupts.

Enumerator

kPIT_TimerInterruptEnable	Timer interrupt enable

14.39.3.3 _pit_status_flags

```
enum _pit_status_flags
#include <fsl_pit.h>
```

List of PIT status flags.

Enumerator

kPIT_TimerFlag	Timer flag

14.39.4 Function Documentation

14.39.4.1 PIT_Deinit()

```
void PIT_Deinit (
    PIT_Type * base)
#include <fsl_pit.h>
```

Gates the PIT clock and disables the PIT module.

Parameters

<i>base</i>	PIT peripheral base address

brief Gates the PIT clock and disables the PIT module.

param base PIT peripheral base address

14.39.4.2 PIT_Init()

```
void PIT_Init (
    PIT_Type * base,
    const pit_config_t * config)
#include <fsl_pit.h>
```

Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.

Note

This API should be called at the beginning of the application using the PIT driver.

Parameters

<i>base</i>	PIT peripheral base address
<i>config</i>	Pointer to the user's PIT config structure

brief Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.

note This API should be called at the beginning of the application using the PIT driver.

param base PIT peripheral base address param config Pointer to the user's PIT config structure

14.40 Platform Specific API

Topics

- [MOD54415, MOD54417, NANO54415, SB800EX](#)
- [MODM7AE70, SBE70LC](#)
- [SOMRT1061](#)

14.40.1 Detailed Description

APIs applicable to specic NetBurner platforms such as:

- MOD54415, MOD54417
- MODM7AE70
- MODRT1171
- SBE70LC
- SB800EX
- SOMRT1061

14.40.2 MOD54415, MOD54417, NANO54415, SB800EX

Topics

- [Interrupt Macro: MOD5441x,NANO54415, SB800EX](#)

14.40.2.1 Detailed Description

APIs applicable to MOD54415, MOD54417, NANO54415, SB800EX

14.40.2.2 Interrupt Macro: MOD5441x,NANO54415, SB800EX

Topics

- [ColdFire Exception and Interrupt Vectors](#)

Vector offset definitions for predefined ColdFire exception and interrupt vectors.

Functions

- `uint16_t GetSR_IntLevel ()`
Get the current interrupt mask level from the Status Register (SR).
- `void SetSR_IntLevel (uint16_t sv)`
Set the interrupt mask level in the Status Register (SR).

14.40.2.2.1 Detailed Description

14.40.2.2.2 Function Documentation

14.40.2.2.2.1 GetSR_IntLevel()

```
uint16_t GetSR_IntLevel ()
#include <cfinter.h>
```

Get the current interrupt mask level from the Status Register (SR).

Retrieves the interrupt priority mask from bits 8-10 of the ColdFire Status Register. The interrupt mask determines which interrupt priority levels are currently blocked. Only interrupts with priority higher than the current mask level can be serviced.

The returned value represents the interrupt mask level:

- 0x0000 (0): All interrupts enabled (no masking)
- 0x0100 (1): Block priority 1 and lower
- 0x0200 (2): Block priority 2 and lower
- 0x0300 (3): Block priority 3 and lower
- 0x0400 (4): Block priority 4 and lower
- 0x0500 (5): Block priority 5 and lower
- 0x0600 (6): Block priority 6 and lower
- 0x0700 (7): All interrupts masked (except level 7 NMI)

Returns

Current interrupt mask level from SR register (bits 8-10), formatted as 0x0X00 where X is the priority level (0-7).

See also

[SetSR_IntLevel\(\)](#)

Expand for Example Usage

Examples

Example 1: Reading and displaying current interrupt mask level

```

void DisplayInterruptStatus() {
    uint16_t currentLevel = GetSR_IntLevel();

    printf("Current interrupt mask level: 0x%04X\r\n", currentLevel);

    // Extract just the level bits
    uint8_t level = (currentLevel » 8) & 0x07;
    printf("Masking interrupts at priority %u and below\r\n", level);

    if (level == 0) {
        printf("All interrupts enabled\r\n");
    } else if (level == 7) {
        printf("All maskable interrupts disabled\r\n");
    } else {
        printf("Interrupts %u and lower are masked\r\n", level);
    }
}

```

14.40.2.2.2.2 SetSR_IntLevel()

```

void SetSR_IntLevel (
    uint16_t sv)
#include <cfinter.h>

```

Set the interrupt mask level in the Status Register (SR).

Sets the interrupt priority mask in bits 8-10 of the ColdFire Status Register. The interrupt mask controls which interrupt priority levels are blocked. Only interrupts with priority strictly higher than the mask level can be serviced. Valid interrupt mask values:

- 0x0000 (0): Enable all interrupts (no masking)
- 0x0100 (1): Block priority 1, allow 2-7
- 0x0200 (2): Block priority 1-2, allow 3-7
- 0x0300 (3): Block priority 1-3, allow 4-7
- 0x0400 (4): Block priority 1-4, allow 5-7
- 0x0500 (5): Block priority 1-5, allow 6-7
- 0x0600 (6): Block priority 1-6, allow 7 only
- 0x0700 (7): Block all maskable interrupts (only level 7 NMI allowed)

Parameters

sv	The interrupt mask level to set, formatted as 0x0X00 where X is the priority level (0-7). Only bits 8-10 are used; other bits in the SR are preserved.

Note

Setting the mask to 0x0700 disables all maskable interrupts. Use with caution as this can affect RTOS operation and real-time responsiveness.

Level 7 interrupts (NMI) cannot be masked and will always be serviced.

Warning

Improper use of this function can cause:

- Missed real-time deadlines
- RTOS scheduler dysfunction
- System deadlock if interrupts remain disabled
- Communication timeouts

Always save the previous interrupt level with [GetSR_IntLevel\(\)](#) before modifying it, and restore the saved level when done.

See also

[GetSR_IntLevel\(\)](#)

Expand for Example Usage

Examples

Example 1: Selective interrupt masking for priority management

```
void MediumPriorityWork() {
    uint16_t savedLevel = GetSR_IntLevel();

    // Block interrupts priority 2 and lower, allow 3+
    SetSR_IntLevel(0x0200);

    // This work can be interrupted by priority 3+ interrupts
    // but not by lower priority interrupts
    ProcessMediumPriorityData();

    SetSR_IntLevel(savedLevel);
}

void HighPriorityWork() {
    uint16_t savedLevel = GetSR_IntLevel();

    // Block interrupts priority 5 and lower, allow 6-7
    SetSR_IntLevel(0x0500);

    // Only highest priority interrupts can preempt this
    ProcessHighPriorityData();

    SetSR_IntLevel(savedLevel);
}
```

14.40.2.2.3 ColdFire Exception and Interrupt Vectors

Vector offset definitions for predefined ColdFire exception and interrupt vectors.

Macros

- #define [CF_ACCESSERROR_VECTOR](#) 2
Access Error Exception Vector (Vector 2)
- #define [CF_ADDRESSERROR_VECTOR](#) 3
Address Error Exception Vector (Vector 3)
- #define [CF_ILLEGAL_INSTRUCTION_VECTOR](#) 4
Illegal Instruction Exception Vector (Vector 4)
- #define [CF_PRIVLEDGE_VIOLATION_VECTOR](#) 8
Privilege Violation Exception Vector (Vector 8)
- #define [CF_TRACE_VECTOR](#) 9
Trace Exception Vector (Vector 9)
- #define [CF_UNIMPLMENTED_A_VECTOR](#) 10
Unimplemented Line-A Opcode Exception Vector (Vector 10)
- #define [CF_UNIMPLMENTED_F_VECTOR](#) 11
Unimplemented Line-F Opcode Exception Vector (Vector 11)
- #define [CF_DEBUG_INTERRUPT](#) 12
Debug Interrupt Exception Vector (Vector 12)
- #define [CF_FORMAT_ERR_VECTOR](#) 14
Format Error Exception Vector (Vector 14)
- #define [CF_UNINITIALIZED_VECTOR](#) 15
Uninitialized Interrupt Exception Vector (Vector 15)
- #define [CF_SPURIOUS_INT_VECTOR](#) 24
Spurious Interrupt Exception Vector (Vector 24)
- #define [CF_AUTOVECTOR_IRQ1](#) 25
Autovector IRQ Level 1 (Vector 25)
- #define [CF_AUTOVECTOR_IRQ2](#) 26
Autovector IRQ Level 2 (Vector 26)

- #define [CF_AUTOVECTOR_IRQ3](#) 27
Autovector IRQ Level 3 (Vector 27)
- #define [CF_AUTOVECTOR_IRQ4](#) 28
Autovector IRQ Level 4 (Vector 28)
- #define [CF_AUTOVECTOR_IRQ5](#) 29
Autovector IRQ Level 5 (Vector 29)
- #define [CF_AUTOVECTOR_IRQ6](#) 30
Autovector IRQ Level 6 (Vector 30)
- #define [CF_AUTOVECTOR_IRQ7](#) 31
Autovector IRQ Level 7 (Vector 31)
- #define [CF_TRAP0_VECTOR](#) 32
TRAP #0 Instruction Vector (Vector 32)
- #define [CF_TRAP1_VECTOR](#) 33
TRAP #1 Instruction Vector (Vector 33)
- #define [CF_TRAP2_VECTOR](#) 34
TRAP #2 Instruction Vector (Vector 34)
- #define [CF_TRAP3_VECTOR](#) 35
TRAP #3 Instruction Vector (Vector 35)
- #define [CF_TRAP4_VECTOR](#) 36
TRAP #4 Instruction Vector (Vector 36)
- #define [CF_TRAP5_VECTOR](#) 37
TRAP #5 Instruction Vector (Vector 37)
- #define [CF_TRAP6_VECTOR](#) 38
TRAP #6 Instruction Vector (Vector 38)
- #define [CF_TRAP7_VECTOR](#) 39
TRAP #7 Instruction Vector (Vector 39)
- #define [CF_TRAP8_VECTOR](#) 40
TRAP #8 Instruction Vector (Vector 40)
- #define [CF_TRAP9_VECTOR](#) 41
TRAP #9 Instruction Vector (Vector 41)
- #define [CF_TRAP10_VECTOR](#) 42
TRAP #10 Instruction Vector (Vector 42)
- #define [CF_TRAP11_VECTOR](#) 43
TRAP #11 Instruction Vector (Vector 43)
- #define [CF_TRAP12_VECTOR](#) 44
TRAP #12 Instruction Vector (Vector 44)
- #define [CF_TRAP13_VECTOR](#) 45
TRAP #13 Instruction Vector (Vector 45)
- #define [CF_TRAP14_VECTOR](#) 46
TRAP #14 Instruction Vector (Vector 46)
- #define [CF_TRAP15_VECTOR](#) 47
TRAP #15 Instruction Vector (Vector 47)
- #define [CF_USER_BASE_VECTOR](#) 64
User-Defined Interrupt Vector Base (Vector 64)

14.40.2.2.3.1 Detailed Description

Vector offset definitions for predefined ColdFire exception and interrupt vectors.

These definitions specify the vector table offsets for various ColdFire processor exceptions, traps, and interrupt sources. The vector table is an array of function pointers where each entry corresponds to a specific exception or interrupt handler.

Vector Table Organization

The ColdFire vector table is organized into several regions:

- **Vectors 0-15:** System exceptions and errors
- **Vectors 24-31:** Autovectored interrupts and spurious interrupt
- **Vectors 32-47:** TRAP instructions (software interrupts)
- **Vectors 64+:** User-defined peripheral interrupts

Usage

Vector offsets are used to install interrupt handlers in the vector table:

```
extern void myHandler();  
vectors[CF_AUTOVECTOR_IRQ5] = myHandler;
```

Note

The actual vector table base address is stored in the Vector Base Register (VBR).

Vectors 0-1 are reserved for the initial stack pointer and program counter.

See also

INTERRUPT

14.40.2.2.3.2 Macro Definition Documentation

CF_ACCESSERROR_VECTOR

```
#define CF_ACCESSERROR_VECTOR 2  
#include <cfinter.h>
```

Access Error Exception Vector (Vector 2)

Triggered when an illegal memory access occurs, such as:

- Attempting to write to read-only memory
- Accessing unmapped memory regions
- Bus timeout or bus error
- Memory protection violations

Note

This is a critical exception that typically indicates a hardware fault or programming error such as dereferencing a NULL or invalid pointer.

Warning

If this exception occurs, the system state may be unstable. The handler should log diagnostic information and consider system reset.

Expand for Example Usage

Examples

Example: Installing an access error handler

```

void AccessErrorHandler() {
    // Log error information
    printf("FATAL: Access Error Exception\r\n");
    printf("PC: 0x%08lX\r\n", GetExceptionPC());

    // Optionally reset the system
    ForceReboot();
}

void UserMain(void *pd) {
    init();

    // Install access error handler
    extern void AccessErrorHandler();
    vectors[CF_ACCESSERROR_VECTOR] = AccessErrorHandler;

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

CF_ADDRESSERROR_VECTOR

```

#define CF_ADDRESSERROR_VECTOR 3
#include <cfinter.h>

```

Address Error Exception Vector (Vector 3)

Triggered when the processor attempts to access a misaligned address or perform an invalid address operation:

- Word access (16-bit) on odd address
- Long word access (32-bit) on non-aligned address
- Invalid instruction fetch address

Note

ColdFire processors require proper data alignment. Word accesses must be word-aligned, and long word accesses must be long-word aligned.

Warning

Address errors often indicate pointer arithmetic errors or structure packing issues. Check your data structure alignment.

CF_AUTOVECTOR_IRQ1

```

#define CF_AUTOVECTOR_IRQ1 25
#include <cfinter.h>

```

Autovector IRQ Level 1 (Vector 25)

Autovector interrupt request level 1 (lowest priority maskable interrupt).

Note

Autovector interrupt requests provide automatic vector generation without requiring the interrupting device to supply a vector number.

See also

[CF_AUTOVECTOR_IRQ2](#) through [CF_AUTOVECTOR_IRQ7](#) for higher priority levels

CF_AUTOVECTOR_IRQ2

```

#define CF_AUTOVECTOR_IRQ2 26
#include <cfinter.h>

```

Autovector IRQ Level 2 (Vector 26)

Autovector interrupt request level 2.

CF_AUTOVECTOR_IRQ3

```
#define CF_AUTOVECTOR_IRQ3 27
#include <cfinter.h>
Autovector IRQ Level 3 (Vector 27)
Autovectored interrupt request level 3.
```

CF_AUTOVECTOR_IRQ4

```
#define CF_AUTOVECTOR_IRQ4 28
#include <cfinter.h>
Autovector IRQ Level 4 (Vector 28)
Autovectored interrupt request level 4.
```

CF_AUTOVECTOR_IRQ5

```
#define CF_AUTOVECTOR_IRQ5 29
#include <cfinter.h>
Autovector IRQ Level 5 (Vector 29)
Autovectored interrupt request level 5.
```

CF_AUTOVECTOR_IRQ6

```
#define CF_AUTOVECTOR_IRQ6 30
#include <cfinter.h>
Autovector IRQ Level 6 (Vector 30)
Autovectored interrupt request level 6.
```

CF_AUTOVECTOR_IRQ7

```
#define CF_AUTOVECTOR_IRQ7 31
#include <cfinter.h>
Autovector IRQ Level 7 (Vector 31)
Autovectored interrupt request level 7 (highest priority, non-maskable).
```

Note

Level 7 interrupts cannot be masked by the interrupt priority mask in the status register and will always be serviced.

Warning

Use level 7 only for critical, non-maskable events such as power failure or critical hardware faults.

CF_DEBUG_INTERRUPT

```
#define CF_DEBUG_INTERRUPT 12
#include <cfinter.h>
Debug Interrupt Exception Vector (Vector 12)
Triggered by the on-chip debug module for hardware breakpoints and debugging. Used by BDM (Background Debug Mode) and JTAG debuggers.
```

Note

This vector is typically managed by the debugging tools and should not be modified by application code.

CF_FORMAT_ERR_VECTOR

```
#define CF_FORMAT_ERR_VECTOR 14
```

```
#include <cfinter.h>
```

Format Error Exception Vector (Vector 14)

Triggered when the processor encounters an invalid stack frame format during an exception return (RTE instruction).

Note

This typically indicates stack corruption or an attempt to return from an exception with an improperly formatted exception frame.

Warning

Often indicates serious stack corruption issues.

CF_ILLEGAL_INSTRUCTION_VECTOR

```
#define CF_ILLEGAL_INSTRUCTION_VECTOR 4
```

```
#include <cfinter.h>
```

Illegal Instruction Exception Vector (Vector 4)

Triggered when the processor attempts to execute:

- An undefined or invalid opcode
- A corrupted instruction
- Code from a data region

Note

This can occur due to:

- Executing data as code (bad function pointer)
- Stack corruption causing return to invalid address
- Flash memory corruption

Warning

Often indicates serious firmware corruption or programming errors.

CF_PRIVLEDGE_VIOLATION_VECTOR

```
#define CF_PRIVLEDGE_VIOLATION_VECTOR 8
```

```
#include <cfinter.h>
```

Privilege Violation Exception Vector (Vector 8)

Triggered when user mode code attempts to execute a privileged instruction or access a privileged resource that requires supervisor mode.

Privileged operations include:

- Modifying the status register (SR)
- Accessing certain control registers
- Executing STOP, RESET, or RTE instructions

Note

Most NetBurner applications run in supervisor mode, so this exception is uncommon unless explicitly using user/supervisor mode separation.

CF_SPURIOUS_INT_VECTOR

```
#define CF_SPURIOUS_INT_VECTOR 24
```

```
#include <cfinter.h>
```

Spurious Interrupt Exception Vector (Vector 24)

Triggered when an interrupt is acknowledged but no interrupt source is found. This can occur due to:

- Interrupt line noise or glitches
- Interrupt cleared between assertion and acknowledgment
- Hardware malfunction

Note

A spurious interrupt handler should simply return without taking action.

Frequent spurious interrupts may indicate hardware problems.

Expand for Example Usage**Examples**

Example: Spurious interrupt handler with counter

```
volatile uint32_t spuriousIntCount = 0;

INTERRUPT(spuriousIntHandler, 0x2000)
{
    spuriousIntCount++;
    // Just return - no action needed
}

void UserMain(void *pd) {
    init();

    extern void spuriousIntHandler();
    vectors[CF_SPURIOUS_INT_VECTOR] = spuriousIntHandler;

    while(1) {
        if (spuriousIntCount > 100) {
            printf("WARNING: Excessive spurious interrupts: %lu\r\n",
                  spuriousIntCount);
        }
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

CF_TRACE_VECTOR

```
#define CF_TRACE_VECTOR 9
```

```
#include <cfinter.h>
```

Trace Exception Vector (Vector 9)

Triggered after each instruction when trace mode is enabled in the status register. Used for single-stepping through code during debugging.

Note

This is primarily used by debuggers and development tools.

The T0 and T1 bits in the SR control trace mode operation.

CF_TRAP0_VECTOR

```
#define CF_TRAP0_VECTOR 32
```

```
#include <cfinter.h>
```

TRAP #0 Instruction Vector (Vector 32)

Software interrupt triggered by the TRAP #0 instruction. TRAP instructions provide a mechanism for software-generated exceptions and system calls.

Note

TRAP instructions are often used to implement system call interfaces or breakpoints in debugging.

Expand for Example Usage**Examples****Example: Using TRAP for system calls**

```
// System call handler
INTERRUPT(trap0Handler, 0x2700)
{
    // Read D0 register to determine system call number
    // Implement system call dispatch logic
    HandleSystemCall();
}

void UserMain(void *pd) {
    init();

    extern void trap0Handler();
    vectors[CF_TRAP0_VECTOR] = trap0Handler;

    // Application can now use: asm("trap #0");

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

CF_TRAP10_VECTOR

```
#define CF_TRAP10_VECTOR 42
```

```
#include <cfinter.h>
```

TRAP #10 Instruction Vector (Vector 42)

Software interrupt triggered by the TRAP #10 instruction.

CF_TRAP11_VECTOR

```
#define CF_TRAP11_VECTOR 43
```

```
#include <cfinter.h>
```

TRAP #11 Instruction Vector (Vector 43)

Software interrupt triggered by the TRAP #11 instruction.

CF_TRAP12_VECTOR

```
#define CF_TRAP12_VECTOR 44
```

```
#include <cfinter.h>
```

TRAP #12 Instruction Vector (Vector 44)

Software interrupt triggered by the TRAP #12 instruction.

CF_TRAP13_VECTOR

```
#define CF_TRAP13_VECTOR 45
```

```
#include <cfinter.h>
```

TRAP #13 Instruction Vector (Vector 45)

Software interrupt triggered by the TRAP #13 instruction.

CF_TRAP14_VECTOR

```
#define CF_TRAP14_VECTOR 46
```

```
#include <cfinter.h>
```

TRAP #14 Instruction Vector (Vector 46)

Software interrupt triggered by the TRAP #14 instruction.

CF_TRAP15_VECTOR

```
#define CF_TRAP15_VECTOR 47
```

```
#include <cfinter.h>
```

TRAP #15 Instruction Vector (Vector 47)

Software interrupt triggered by the TRAP #15 instruction.

CF_TRAP1_VECTOR

```
#define CF_TRAP1_VECTOR 33
```

```
#include <cfinter.h>
```

TRAP #1 Instruction Vector (Vector 33)

Software interrupt triggered by the TRAP #1 instruction.

CF_TRAP2_VECTOR

```
#define CF_TRAP2_VECTOR 34
```

```
#include <cfinter.h>
```

TRAP #2 Instruction Vector (Vector 34)

Software interrupt triggered by the TRAP #2 instruction.

CF_TRAP3_VECTOR

```
#define CF_TRAP3_VECTOR 35
```

```
#include <cfinter.h>
```

TRAP #3 Instruction Vector (Vector 35)

Software interrupt triggered by the TRAP #3 instruction.

CF_TRAP4_VECTOR

```
#define CF_TRAP4_VECTOR 36
```

```
#include <cfinter.h>
```

TRAP #4 Instruction Vector (Vector 36)

Software interrupt triggered by the TRAP #4 instruction.

CF_TRAP5_VECTOR

```
#define CF_TRAP5_VECTOR 37
```

```
#include <cfinter.h>
```

TRAP #5 Instruction Vector (Vector 37)

Software interrupt triggered by the TRAP #5 instruction.

CF_TRAP6_VECTOR

```
#define CF_TRAP6_VECTOR 38
```

```
#include <cfinter.h>
```

TRAP #6 Instruction Vector (Vector 38)

Software interrupt triggered by the TRAP #6 instruction.

CF_TRAP7_VECTOR

```
#define CF_TRAP7_VECTOR 39
```

```
#include <cfinter.h>
```

TRAP #7 Instruction Vector (Vector 39)

Software interrupt triggered by the TRAP #7 instruction.

CF_TRAP8_VECTOR

```
#define CF_TRAP8_VECTOR 40
```

```
#include <cfinter.h>
```

TRAP #8 Instruction Vector (Vector 40)

Software interrupt triggered by the TRAP #8 instruction.

CF_TRAP9_VECTOR

```
#define CF_TRAP9_VECTOR 41
#include <cfinter.h>
TRAP #9 Instruction Vector (Vector 41)
Software interrupt triggered by the TRAP #9 instruction.
```

CF_UNIMPLMENTED_A_VECTOR

```
#define CF_UNIMPLMENTED_A_VECTOR 10
#include <cfinter.h>
Unimplemented Line-A Opcode Exception Vector (Vector 10)
Triggered when the processor encounters an instruction with bits 15-12 = 1010 (0xA). These opcodes are reserved and unimplemented on ColdFire processors.
```

Note

Can be used to implement software emulation of missing instructions or for custom instruction extensions.

CF_UNIMPLMENTED_F_VECTOR

```
#define CF_UNIMPLMENTED_F_VECTOR 11
#include <cfinter.h>
Unimplemented Line-F Opcode Exception Vector (Vector 11)
Triggered when the processor encounters an instruction with bits 15-12 = 1111 (0xF). These opcodes are reserved and unimplemented on ColdFire processors.
```

Note

Can be used to implement co-processor emulation or custom extensions.

CF_UNINITIALIZED_VECTOR

```
#define CF_UNINITIALIZED_VECTOR 15
#include <cfinter.h>
Uninitialized Interrupt Exception Vector (Vector 15)
Triggered when an interrupt occurs but the interrupt controller has not been properly initialized or the interrupt vector has not been programmed.
```

Note

This can occur during system startup if interrupts are enabled before the vector table is fully initialized.

Warning

All interrupt vectors should be initialized before enabling interrupts to avoid this exception.

CF_USER_BASE_VECTOR

```
#define CF_USER_BASE_VECTOR 64
#include <cfinter.h>
User-Defined Interrupt Vector Base (Vector 64)
Starting vector number for user-defined peripheral interrupts. Vectors 64 and above are available for assignment to custom interrupt sources such as:
```

- Timers
- UART/Serial ports
- GPIO interrupts

- DMA controllers
- ADC converters
- Custom peripherals

The actual vector number for a peripheral interrupt is typically: `CF_USER_BASE_VECTOR + peripheral_`
`irq_number`

Note

The specific IRQ numbers and mappings are platform-dependent. Consult your NetBurner platform documentation for peripheral interrupt assignments.

Vectors 48-63 are reserved by Freescale/NXP and should not be used.

Expand for Example Usage

Examples

Example 1: Installing a timer interrupt handler

```
#define TIMER0_IRQ 0 // Platform-specific IRQ number

INTERRUPT(timer0Handler, 0x2300)
{
    // Timer interrupt service routine
    static uint32_t timerTicks = 0;
    timerTicks++;

    // Clear timer interrupt flag
    TIMER0_STATUS_REG |= TIMER_INT_FLAG;
}

void UserMain(void *pd) {
    init();

    // Install handler at user vector base + IRQ number
    extern void timer0Handler();
    vectors[CF_USER_BASE_VECTOR + TIMER0_IRQ] = timer0Handler;

    // Enable timer interrupt
    TIMER0_CONTROL_REG |= TIMER_ENABLE | TIMER_INT_ENABLE;

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

Example 2: Multiple peripheral interrupts

```
// Platform-specific IRQ assignments
#define UART0_IRQ 0
#define UART1_IRQ 1
#define TIMER0_IRQ 2
#define GPIO_IRQ 3

INTERRUPT(uart0Handler, 0x2300)
{
    HandleUART0Interrupt();
}

INTERRUPT(uart1Handler, 0x2300)
{
    HandleUART1Interrupt();
}

INTERRUPT(timer0Handler, 0x2400)
{
    HandleTimer0Interrupt();
}

INTERRUPT(gpioHandler, 0x2200)
{
    HandleGPIOInterrupt();
}

void InitializeInterrupts() {
    extern void uart0Handler();
    extern void uart1Handler();
    extern void timer0Handler();
    extern void gpioHandler();

    // Install all handlers
    vectors[CF_USER_BASE_VECTOR + UART0_IRQ] = uart0Handler;
```

```

    vectors[CF_USER_BASE_VECTOR + UART1_IRQ] = uart1Handler;
    vectors[CF_USER_BASE_VECTOR + TIMER0_IRQ] = timer0Handler;
    vectors[CF_USER_BASE_VECTOR + GPIO_IRQ] = gpioHandler;

    printf("All interrupt handlers installed\r\n");
}

```

Example 3: Vector table dump utility

```

void DumpVectorTable() {
    printf("\r\n=== ColdFire Vector Table ===\r\n");

    printf("\r\nException Vectors:\r\n");
    printf("  Access Error:      %d (0x%08lX)\r\n",
           CF_ACCESSERROR_VECTOR, (uint32_t)vectors[CF_ACCESSERROR_VECTOR]);
    printf("  Address Error:      %d (0x%08lX)\r\n",
           CF_ADDRESSERROR_VECTOR, (uint32_t)vectors[CF_ADDRESSERROR_VECTOR]);
    printf("  Illegal Instruction: %d (0x%08lX)\r\n",
           CF_ILLEGAL_INSTRUCTION_VECTOR, (uint32_t)vectors[CF_ILLEGAL_INSTRUCTION_VECTOR]);

    printf("\r\nAutovector IRQs:\r\n");
    for (int i = 1; i <= 7; i++) {
        printf("  IRQ%d: Vector %d (0x%08lX)\r\n",
               i, CF_AUTOVECTOR_IRQ1 + (i-1),
               (uint32_t)vectors[CF_AUTOVECTOR_IRQ1 + (i-1)]);
    }

    printf("\r\nTRAP Vectors:\r\n");
    for (int i = 0; i <= 15; i++) {
        printf("  TRAP #d: Vector %d (0x%08lX)\r\n",
               i, CF_TRAP0_VECTOR + i,
               (uint32_t)vectors[CF_TRAP0_VECTOR + i]);
    }

    printf("\r\nUser Vectors (first 10):\r\n");
    for (int i = 0; i < 10; i++) {
        printf("  Vector %d: 0x%08lX\r\n",
               CF_USER_BASE_VECTOR + i,
               (uint32_t)vectors[CF_USER_BASE_VECTOR + i]);
    }
}

```

Example 4: Safe vector installation with validation

```

bool InstallUserInterruptHandler(uint8_t irqNum, void (*handler)(),
                                const char *name)
{
    if (handler == NULL) {
        printf("Error: NULL handler for %s\r\n", name);
        return false;
    }

    uint32_t vectorNum = CF_USER_BASE_VECTOR + irqNum;

    if (vectorNum >= 256) {
        printf("Error: Vector %lu out of range for %s\r\n", vectorNum, name);
        return false;
    }

    // Check if vector already assigned
    if (vectors[vectorNum] != NULL && vectors[vectorNum] != handler) {
        printf("Warning: Vector %lu already assigned for %s\r\n",
               vectorNum, name);
    }

    vectors[vectorNum] = handler;
    printf("Installed %s at vector %lu (IRQ %u)\r\n",
           name, vectorNum, irqNum);

    return true;
}

void UserMain(void *pd) {
    init();

    extern void timer0Handler();
    extern void uart0Handler();

    InstallUserInterruptHandler(0, timer0Handler, "Timer0");
    InstallUserInterruptHandler(1, uart0Handler, "UART0");

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```


14.40.3 MODM7AE70, SBE70LC

Topics

- [EBI - External Bus Interface](#)

14.40.3.1 Detailed Description

APIs applicable to MODM7AE70, SBE70LC

14.40.3.2 EBI - External Bus Interface

Classes

- struct [EBI_CS_cfg_t](#)
Configuration structure for an External Bus Interface (EBI) chip select.

Enumerations

- enum [EBI_CS_BusWidth_t](#) { [EBI_BUS_WIDTH_8](#) = 0 , [EBI_BUS_WIDTH_16](#) = 1 }
Data bus width configuration for External Bus Interface.
- enum [EBI_CS_ByteAccess_t](#) { [EBI_BYTE_ACCESS_SELECT](#) = 0 , [EBI_BYTE_ACCESS_WRITE](#) = 1 }
Byte access mode for 16-bit bus configurations.
- enum [EBI_CS_NWait_t](#) { [EBI_NWAIT_DISABLED](#) = 0 , [EBI_NWAIT_FROZEN](#) = 2 , [EBI_NWAIT_READY](#) = 3 }
NWAIT signal configuration for bus cycle extension.
- enum [EBI_CS_WrMode_t](#) { [EBI_WRITE_MODE_NCS](#) = 0 , [EBI_WRITE_MODE_NWE](#) = 1 }
Write control signal selection.
- enum [EBI_CS_RdMode_t](#) { [EBI_READ_MODE_NCS](#) = 0 , [EBI_READ_MODE_NRD](#) = 1 }
Read control signal selection.

Functions

- void [ConfigureEBI_CSPin](#) (int csNum)
Configure the I/O pin for a given Chip Select for the external data bus.
- void [ConfigureEBI_NWRPin](#) ()
Configure the I/O pin for the active low write/read (NWR) bus signal.
- void [ConfigureEBI_NRDPin](#) ()
Configure the I/O pin for the active low read (NRD) bus signal.
- void [ConfigureEBI_CS](#) (uint32_t csNum, const [EBI_CS_cfg_t](#) &&cfg)
Configure the given Chip Select for the external data bus (rvalue reference version).
- void [ConfigureEBI_CS](#) (uint32_t csNum, const [EBI_CS_cfg_t](#) &cfg)
Configure the given Chip Select for the external data bus (lvalue reference version).

Variables

- uint8_t [EBI_CS_cfg_t::ncs_rd_setup](#)
- uint8_t [EBI_CS_cfg_t::nrd_setup](#)
- uint8_t [EBI_CS_cfg_t::ncs_wr_setup](#)
- uint8_t [EBI_CS_cfg_t::nwe_setup](#)
- uint8_t [EBI_CS_cfg_t::ncs_rd_pulse](#)
- uint8_t [EBI_CS_cfg_t::nrd_pulse](#)
- uint8_t [EBI_CS_cfg_t::ncs_wr_pulse](#)
- uint8_t [EBI_CS_cfg_t::nwe_pulse](#)
- uint16_t [EBI_CS_cfg_t::nrd_cycles](#)
- uint16_t [EBI_CS_cfg_t::nwe_cycles](#)

- [uint8_t EBI_CS_cfg_t::tdf_cycles](#)
- [EBI_CS_BusWidth_t EBI_CS_cfg_t::busWidth](#)
- [EBI_CS_ByteAccess_t EBI_CS_cfg_t::byteAccess](#)
- [EBI_CS_NWait_t EBI_CS_cfg_t::nWait](#)
- [EBI_CS_WrMode_t EBI_CS_cfg_t::wrMode](#)
- [EBI_CS_RdMode_t EBI_CS_cfg_t::rdMode](#)

14.40.3.2.1 Detailed Description

Supported Platforms: MODM7AE70

The Microchip SAME70 EBI peripheral is a parallel communication port that can only operate in "host" mode to communicate with "slave" devices. The EBI peripheral consists of a set of up to 16 data signals, up to 24 address signals and up to 10 different control signals. Some of the benefits of using the EBI bus are that it has high performance in both throughput and latency. It is directly memory mapped to the processor so it is very easy to use in software once configured. The downsides are mostly that it has many signals which will greatly complicate hardware designs and increase the size of external peripheral chips.

The EBI peripheral is highly configurable to fit the specifications of many types of external devices.

A consequence to this high configurability is that there are many timing parameters which makes configuration complicated. The NetBurner examples specify a conservative set of timing parameters with the goal of being compatible with a wide range of peripherals such as FPGAs, CPLDs, buffers, latches displays, flash memory and SRAM. Once successful communication is obtained with the slave target device, these default settings should be refined to increase the performance to match the device you are interfacing with.

Advanced EBI functionality is currently not supported by this examples or this driver. Advanced functionality includes interfacing with SDRAM or NAND memories. Page mode, burst operations and scrambling functionality are all not currently configurable via this driver.

In most applications you will only need to verify the last 5 parameters meet your peripheral requirements: bus width (8 or 16), byte Access 16-bit mode only), timing parameters, read mode (chip select or read signal) and write mode (chip select or write signal).

There are 3 sources of information that are very useful for understanding the EBI:

1. The EBI section of the Microchip SAME70 manual located in "`\nburn\docs`". It is very important to do this as the first step to understand the signal terminology in the examples and header files. Additionally the Static Memory Controller chapter in this processor manual is essential as it explains all the settings and timing parameters exposed by this driver.
2. The NetBurner EBI examples described in the manual and located in "`\nburn\examples\PlatformSpecific\↵MODM7AE70`"
3. The SAME70 [ebi.h](#) header file located in "`\nburn\arch\cortex-m7\cpu\SAME70\include`"

The EBI supports Chip Selects as well as Read and Write signals to interface with peripherals. The basic procedure for configuring the bus is:

- As noted in the examples, the MODM7AE70 has onboard bus buffers to reduce noise and help signal integrity. The following 4 signals must always be configured for EBI functionality as they are control the enable/disable and direction control of the bus buffer ICs:

```
P1[5].function(PINP1_5_NCS2);    // chip select 2, active low
P1[6].function(PINP1_6_NCS0);    // chip select 0, active low
P1[7].function(PINP1_7_NCS3);    // chip select 3, active low
P1[8].function(PINP1_8_NRD);     // Bus read, active low
```

- Declare a Configuration Structure of type [EBI_CS_cfg_t](#). An easy way is to copy one from the examples. Then modify any parameters necessary to meet your specific peripheral requirements, such as bus width, byte access in 16-bit mode, timing parameters, etc.
- Configure the desired chip select using the [ConfigureEBI_CS\(\)](#) function. Note that when enabling the EBI, the MODM7AE70 module requires all three chip selects (NCS0, NCS02, NCS03) to be assigned as chip selects. Additionally the NRD signal must also be configured for NRD bus operation. All of these signals are used for controlling the enable/direction logic of the external bus buffer hardware - they cannot use their alternate functions. Each chip select has a predefined fixed 16MB address range ([MODM7AE70 Memory Map](#)). The

NetBurner API supplied predefined base address variables to make access easier as demonstrated in the EBI examples:

```
extern volatile uint8_t ebi_0_base[]; // Chip select 0
extern volatile uint8_t ebi_1_base[]; // Chip select 1
extern volatile uint8_t ebi_2_base[]; // Chip select 2
extern volatile uint8_t ebi_3_base[]; // Chip select 3
```

In this way you can do a bus write with code such as (please refer to EBI examples for latest code):

```
const uint8_t  EBITestData8 = 0x5A;           // Data byte for testing
const uint32_t EBITestAddressOffset = 0xA5A5; // Address offset from chip select base address

ebi_0_base[EBITestAddressOffset] = EBITestData8; // Write data to EBI, chip select 0
ebi_0_base[0xA5A5] = 0x5A;                     // Same write with hard coded values

uint32_t EBIdata = ebi_0_base[EBITestAddressOffset]; // Read data from EBI, chip select 0
```

Warning

When enabling the EBI for the MODM7AE70 module, NCS0, NCS2, NCS3 and NRD must all be configured for their bus functionality as chip selects or nRead signals, even if not used by your application - they cannot be assigned their alternate functions. Additionally the "void EnableExtBusBuff(bool enable);" function from bsp.h must be used to enable the external bus buffer hardware.

14.40.3.2.2 Enumeration Type Documentation

14.40.3.2.2.1 EBI_CS_BusWidth_t

```
enum EBI_CS_BusWidth_t
#include <ebi.h>
```

Data bus width configuration for External Bus Interface.

Specifies the data bus width for communication with external devices on the EBI. The bus width must match the physical connection and capabilities of the external device connected to the chip select.

Note

The bus width setting affects how data is transferred and accessed on the external bus. Ensure the external device supports the selected width.

See also

[EBI_CS_ByteAccess_t](#)

Enumerator

EBI_BUS_WIDTH_8	8-bit data bus width (D0-D7)
EBI_BUS_WIDTH_16	16-bit data bus width (D0-D15)

14.40.3.2.2.2 EBI_CS_ByteAccess_t

```
enum EBI_CS_ByteAccess_t
#include <ebi.h>
```

Byte access mode for 16-bit bus configurations.

Controls how 8-bit accesses are handled when the bus is configured for 16-bit mode. This setting determines which control signal (chip select or write enable) is used to select the high or low byte during 8-bit transfers on a 16-bit bus.

Note

This setting is only relevant when using EBI_BUS_WIDTH_16. It has no effect in 8-bit bus mode.

Warning

Incorrect byte access configuration can result in data corruption when performing byte-level operations on a 16-bit bus.

See also

[EBI_CS_BusWidth_t](#)

Expand for Example Usage

Examples

Example 1: Configuring byte access using chip select

```
void ConfigureEBI_16BitWithByteAccess() {
    EBI_CS_BusWidth_t busWidth = EBI_BUS_WIDTH_16;
    EBI_CS_ByteAccess_t byteAccess = EBI_BYTE_ACCESS_SELECT;

    // Configure for 16-bit bus with CS-controlled byte access
    // This allows upper/lower byte selection via chip select

    iprintf("16-bit bus configured with CS byte select\r\n");
}
```

Example 2: Configuring byte access using write signal

```
void ConfigureEBI_16BitWithWriteByteAccess() {
    EBI_CS_BusWidth_t busWidth = EBI_BUS_WIDTH_16;
    EBI_CS_ByteAccess_t byteAccess = EBI_BYTE_ACCESS_WRITE;

    // Configure for 16-bit bus with write-controlled byte access
    // This uses the write signal to select upper/lower byte

    iprintf("16-bit bus configured with write signal byte select\r\n");
}
```

Enumerator

EBI_BYTE_ACCESS_SELECT	Use chip select signal to access individual bytes on 16-bit bus.
EBI_BYTE_ACCESS_WRITE	Use write enable signal to access individual bytes on 16-bit bus.

14.40.3.2.2.3 EBI_CS_NWait_t

```
enum EBI_CS_NWait_t
#include <ebi.h>
```

NWAIT signal configuration for bus cycle extension.

Controls how the external NWAIT signal is used to extend read/write cycles when interfacing with slow external devices. The NWAIT signal allows external devices to insert wait states, extending the access time as needed.

This is commonly used with:

- Slow flash memory devices
- Variable-speed peripherals
- Devices that need dynamic timing adjustment

Note

When disabled, the NWAIT signal is ignored and all timing is determined by the configured setup, pulse, and hold times.

In Frozen mode, the NWAIT signal can extend the access time indefinitely while the signal is asserted.

In Ready mode, the NWAIT signal is sampled and can control cycle extension based on device readiness.

Warning

Using NWAIT modes with devices that don't properly drive the NWAIT signal can result in bus hangs or indefinite waits.

Expand for Example Usage

Examples

Example 1: Disabled NWAIT for fast SRAM

```
void ConfigureFastSRAM() {
    EBI_CS_NWait_t nwaitMode = EBI_NWAIT_DISABLED;

    // Fast SRAM doesn't need wait states
    // All timing controlled by setup/pulse/hold registers

    iprintf("EBI configured for fast SRAM (NWAIT disabled)\r\n");
}
```

Example 2: Frozen mode for slow flash memory

```
void ConfigureSlowFlash() {
    EBI_CS_NWait_t nwaitMode = EBI_NWAIT_FROZEN;

    // Flash can extend cycles as needed via NWAIT
    // Useful for write operations that need variable time

    iprintf("EBI configured for flash with frozen NWAIT mode\r\n");
}
```

Example 3: Ready mode for variable-speed peripheral

```
void ConfigureVariableSpeedDevice() {
    EBI_CS_NWait_t nwaitMode = EBI_NWAIT_READY;

    // Device can signal when it's ready via NWAIT
    // Allows dynamic timing based on device state

    iprintf("EBI configured with ready mode NWAIT\r\n");
}
```

Example 4: Comparing NWAIT modes for different device types

```
void ConfigureMultipleDevices() {
    // CS0: Fast SRAM - no wait states needed
    EBI_CS_NWait_t cs0_nwait = EBI_NWAIT_DISABLED;

    // CS1: Flash memory - can hold bus during programming
    EBI_CS_NWait_t cs1_nwait = EBI_NWAIT_FROZEN;

    // CS2: Variable speed FPGA - signals readiness
    EBI_CS_NWait_t cs2_nwait = EBI_NWAIT_READY;

    iprintf("Configured 3 devices with different NWAIT modes\r\n");
}
```

Enumerator

EBI_NWAIT_DISABLED	NWAIT signal ignored, all timing from configuration registers.
EBI_NWAIT_FROZEN	NWAIT can freeze/extend the access cycle indefinitely.
EBI_NWAIT_READY	NWAIT sampled to determine device ready state.

14.40.3.2.2.4 EBI_CS_RdMode_t

```
enum EBI_CS_RdMode_t
```

```
#include <ebi.h>
```

Read control signal selection.

Determines which signal controls read operations from external devices on the EBI. This setting affects how read cycles are generated and controlled.

- **NCS mode:** Read cycle controlled by chip select assertion
- **NRD mode:** Read cycle controlled by dedicated read enable signal

Note

NRD (Read Enable) mode provides more precise read timing control and is commonly used with memory devices that have separate read enable inputs.

NCS mode may be suitable for simpler devices or when minimizing control signal requirements.

See also

[EBI_CS_WrMode_t](#)

[ConfigureEBI_NRDPin\(\)](#)

Expand for Example Usage

Examples

Example 1: Using NRD for standard SRAM

```
void ConfigureSRAM_WithReadEnable() {
    EBI_CS_RdMode_t readMode = EBI_READ_MODE_NRD;

    // Standard SRAM with dedicated OE/RD pin
    // Provides precise read pulse timing
    ConfigureEBI_NRDPin(); // Configure the read enable pin

    iprintf("SRAM configured with NRD read control\r\n");
}
```

Example 2: Using NCS for simple peripheral

```
void ConfigureSimpleReadDevice() {
    EBI_CS_RdMode_t readMode = EBI_READ_MODE_NCS;

    // Simple device controlled only by chip select
    // Data output when CS is asserted

    iprintf("Device configured with NCS read control\r\n");
}
```

Example 3: Complete read/write configuration

```
void ConfigureExternalMemory() {
    EBI_CS_WrMode_t writeMode = EBI_WRITE_MODE_NWE;
    EBI_CS_RdMode_t readMode = EBI_READ_MODE_NRD;

    // Configure pins for read and write operations
    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();

    iprintf("Memory configured with NRD/NWE control signals\r\n");
}
```

Example 4: Mixed control modes for different chip selects

```
void ConfigureMixedDevices() {
    // CS0: Memory with dedicated control signals
    EBI_CS_RdMode_t cs0_read = EBI_READ_MODE_NRD;
    EBI_CS_WrMode_t cs0_write = EBI_WRITE_MODE_NWE;

    // CS1: Simple I/O using only chip select
    EBI_CS_RdMode_t cs1_read = EBI_READ_MODE_NCS;
    EBI_CS_WrMode_t cs1_write = EBI_WRITE_MODE_NCS;

    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();

    iprintf("CS0: Full control signals (NRD/NWE)\r\n");
    iprintf("CS1: Chip select only (NCS)\r\n");
}
```

Enumerator

EBI_READ_MODE_NCS	Read controlled by chip select (NCS) assertion.
EBI_READ_MODE_NRD	Read controlled by read enable (NRD) signal.

14.40.3.2.2.5 EBI_CS_WrMode_t

```
enum EBI_CS_WrMode_t
```

```
#include <ebi.h>
```

Write control signal selection.

Determines which signal controls write operations to external devices on the EBI. This setting affects how write cycles are generated and controlled.

- **NCS mode:** Write cycle controlled by chip select assertion

- **NWE mode:** Write cycle controlled by dedicated write enable signal

Note

NWE (Write Enable) mode provides more precise write timing control and is commonly used with memory devices that have separate write enable inputs.

NCS mode may be suitable for simpler devices or when minimizing control signal requirements.

See also

[EBI_CS_RdMode_t](#)

[ConfigureEBI_NWRPin\(\)](#)

Expand for Example Usage**Examples****Example 1: Using NWE for standard SRAM**

```
void ConfigureSRAM_WithWriteEnable() {
    EBI_CS_WrMode_t writeMode = EBI_WRITE_MODE_NWE;

    // Standard SRAM with dedicated WE pin
    // Provides precise write pulse timing
    ConfigureEBI_NWRPin(); // Configure the write enable pin

    iprintf("SRAM configured with NWE write control\r\n");
}
```

Example 2: Using NCS for simple peripheral

```
void ConfigureSimplePeripheral() {
    EBI_CS_WrMode_t writeMode = EBI_WRITE_MODE_NCS;

    // Simple device controlled only by chip select
    // Write occurs when CS is asserted

    iprintf("Peripheral configured with NCS write control\r\n");
}
```

Example 3: Comparing write modes for different devices

```
void ConfigureMultipleDeviceWrites() {
    // Memory device: use dedicated write enable
    EBI_CS_WrMode_t sram_write = EBI_WRITE_MODE_NWE;

    // Simple I/O device: use chip select
    EBI_CS_WrMode_t io_write = EBI_WRITE_MODE_NCS;

    iprintf("CS0: NWE mode for memory\r\n");
    iprintf("CS1: NCS mode for I/O\r\n");
}
```

Enumerator

EBI_WRITE_MODE_NCS	Write controlled by chip select (NCS) assertion.
EBI_WRITE_MODE_NWE	Write controlled by write enable (NWE) signal.

14.40.3.2.3 Function Documentation**14.40.3.2.3.1 ConfigureEBI_CS() [1/2]**

```
void ConfigureEBI_CS (
    uint32_t csNum,
    const EBI_CS_cfg_t && cfg)

#include <ebi.h>
```

Configure the given Chip Select for the external data bus (rvalue reference version).

Configures an External Bus Interface (EBI) chip select with the specified timing and control parameters. This overload accepts an rvalue reference, allowing efficient configuration with temporary configuration objects.

This function sets up all timing parameters (setup, pulse, hold times), control signal modes, bus width, and wait state configuration for the specified chip select line.

Parameters

<i>csNum</i>	The EBI Chip Select number to configure (typically 0-3, platform-dependent).
<i>cfg</i>	Rvalue reference to the configuration structure. The configuration is moved into the function, making this efficient for temporary configuration objects.

Note

This function should be called after pin configuration functions:

- [ConfigureEBI_CSPin\(\)](#)
- [ConfigureEBI_NWRPin\(\)](#) (if using write operations)
- [ConfigureEBI_NRDPin\(\)](#) (if using read operations)

All timing parameters in the configuration are specified in EBI clock cycles. Calculate the required values based on your EBI clock frequency and device datasheet.

Warning

Incorrect timing configuration can cause data corruption or device malfunction. Always verify timing parameters against the external device datasheet.

Ensure the chip select number is valid for your platform. Invalid values may result in undefined behavior.

See also

[ConfigureEBI_CS\(uint32_t, const EBI_CS_cfg_t&\)](#)

[ConfigureEBI_CSPin\(\)](#)

[EBI_CS_cfg_t](#)

Expand for Example Usage

Examples

Example 1: Configure CS0 with temporary configuration object

```
void InitializeExternalSRAM() {
    // Configure the chip select pin first
    ConfigureEBI_CSPin(0);
    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();

    // Configure CS0 using temporary configuration (rvalue)
    // This overload is efficient when creating config inline
    ConfigureEBI_CS(0, EBI_CS_cfg_t{
        .ncs_rd_setup = 2,
        .nrd_setup = 2,
        .ncs_rd_pulse = 10,
        .nrd_pulse = 10,
        .nrd_cycles = 15,
        .ncs_wr_setup = 2,
        .nwe_setup = 2,
        .ncs_wr_pulse = 10,
        .nwe_pulse = 10,
        .nwe_cycles = 15,
        .tdf_cycles = 2,
        .busWidth = EBI_BUS_WIDTH_16,
        .byteAccess = EBI_BYTE_ACCESS_WRITE,
        .nWait = EBI_NWAIT_DISABLED,
        .wrMode = EBI_WRITE_MODE_NWE,
        .rdMode = EBI_READ_MODE_NRD
    });

    iprintf("CS0 configured for SRAM\r\n");
}
```


Example 2: Using a factory function that returns configuration

```
EBI_CS_cfg_t CreateFastSRAMConfig() {
    return EBI_CS_cfg_t{
        .ncs_rd_setup = 2, .nrd_setup = 2,
        .ncs_rd_pulse = 10, .nrd_pulse = 10,
        .nrd_cycles = 15,
        .ncs_wr_setup = 2, .nwe_setup = 2,
        .ncs_wr_pulse = 10, .nwe_pulse = 10,
        .nwe_cycles = 15,
        .tdf_cycles = 2,
        .busWidth = EBI_BUS_WIDTH_16,
        .byteAccess = EBI_BYTE_ACCESS_WRITE,
        .nWait = EBI_NWAIT_DISABLED,
        .wrMode = EBI_WRITE_MODE_NWE,
        .rdMode = EBI_READ_MODE_NRD
    };
}

void UserMain(void *pd) {
    init();

    ConfigureEBI_CSPin(0);
    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();

    // Rvalue overload called with returned temporary
    ConfigureEBI_CS(0, CreateFastSRAMConfig());

    iprintf("CS0 configured using factory function\r\n");

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

Example 3: Chain configuration with std::move

```
void ConfigureMultipleChipSelects() {
    ConfigureEBI_CSPin(0);
    ConfigureEBI_CSPin(1);
    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();

    EBI_CS_cfg_t baseConfig = {
        .ncs_rd_setup = 2, .nrd_setup = 2,
        .ncs_rd_pulse = 10, .nrd_pulse = 10,
        .nrd_cycles = 15,
        .ncs_wr_setup = 2, .nwe_setup = 2,
        .ncs_wr_pulse = 10, .nwe_pulse = 10,
        .nwe_cycles = 15,
        .tdf_cycles = 2,
        .busWidth = EBI_BUS_WIDTH_16,
        .byteAccess = EBI_BYTE_ACCESS_WRITE,
        .nWait = EBI_NWAIT_DISABLED,
        .wrMode = EBI_WRITE_MODE_NWE,
        .rdMode = EBI_READ_MODE_NRD
    };

    // Configure CS0 with the base config
    ConfigureEBI_CS(0, baseConfig); // Uses lvalue overload

    // Modify for CS1 (slower device)
    baseConfig.nrd_cycles = 20;
    baseConfig.nwe_cycles = 20;

    // Move to CS1 configuration
    ConfigureEBI_CS(1, std::move(baseConfig)); // Uses rvalue overload

    iprintf("CS0 and CS1 configured\r\n");
}
```

14.40.3.2.3.2 ConfigureEBI_CS() [2/2]

```
void ConfigureEBI_CS (
    uint32_t csNum,
    const EBI_CS_cfg_t & cfg)
```

```
#include <ebi.h>
```

Configure the given Chip Select for the external data bus (lvalue reference version).

Configures an External Bus Interface (EBI) chip select with the specified timing and control parameters. This overload accepts an lvalue reference, allowing configuration with existing configuration objects that may be reused.

This function sets up all timing parameters (setup, pulse, hold times), control signal modes, bus width, and wait state configuration for the specified chip select line.

Parameters

<i>csNum</i>	The EBI Chip Select number to configure (typically 0-3, platform-dependent).
<i>cfg</i>	Const reference to the configuration structure. The configuration is copied, allowing the original to be reused for other chip selects.

Note

This function should be called after pin configuration functions:

- [ConfigureEBI_CSPin\(\)](#)
- [ConfigureEBI_NWRPin\(\)](#) (if using write operations)
- [ConfigureEBI_NRDPin\(\)](#) (if using read operations)

All timing parameters in the configuration are specified in EBI clock cycles. Calculate the required values based on your EBI clock frequency and device datasheet.

Warning

Incorrect timing configuration can cause data corruption or device malfunction. Always verify timing parameters against the external device datasheet.

Ensure the chip select number is valid for your platform. Invalid values may result in undefined behavior.

See also

[ConfigureEBI_CS\(uint32_t, const EBI_CS_cfg_t&\)](#)
[ConfigureEBI_CSPin\(\)](#)
[EBI_CS_cfg_t](#)

Expand for Example Usage

Examples

Example 1: Basic CS configuration with persistent config object

```
void InitializeExternalMemory() {
    // Configure the necessary pins
    ConfigureEBI_CSPin(0);
    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();

    // Create configuration object
    EBI_CS_cfg_t sramConfig = {
        .ncs_rd_setup = 2,
        .nrd_setup = 2,
        .ncs_rd_pulse = 10,
        .nrd_pulse = 10,
        .nrd_cycles = 15,
        .ncs_wr_setup = 2,
        .nwe_setup = 2,
        .ncs_wr_pulse = 10,
        .nwe_pulse = 10,
        .nwe_cycles = 15,
        .tdf_cycles = 2,
        .busWidth = EBI_BUS_WIDTH_16,
        .byteAccess = EBI_BYTE_ACCESS_WRITE,
        .nWait = EBI_NWAIT_DISABLED,
        .wrMode = EBI_WRITE_MODE_NWE,
        .rdMode = EBI_READ_MODE_NRD
    };

    // Configure CS0 using lvalue reference
    ConfigureEBI_CS(0, sramConfig);

    iprintf("CS0 configured for SRAM\r\n");
}
```

Example 2: Reusing configuration for multiple chip selects

```

void ConfigureMultipleSimilarDevices() {
    // Configure pins for all chip selects
    ConfigureEBI_CSPin(0);
    ConfigureEBI_CSPin(1);
    ConfigureEBI_CSPin(2);
    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();

    // Base configuration for similar devices
    EBI_CS_cfg_t baseConfig = {
        .ncs_rd_setup = 2, .nrd_setup = 2,
        .ncs_rd_pulse = 10, .nrd_pulse = 10,
        .nrd_cycles = 15,
        .ncs_wr_setup = 2, .nwe_setup = 2,
        .ncs_wr_pulse = 10, .nwe_pulse = 10,
        .nwe_cycles = 15,
        .tdf_cycles = 2,
        .busWidth = EBI_BUS_WIDTH_16,
        .byteAccess = EBI_BYTE_ACCESS_WRITE,
        .nWait = EBI_NWAIT_DISABLED,
        .wrMode = EBI_WRITE_MODE_NWE,
        .rdMode = EBI_READ_MODE_NRD
    };

    // Configure multiple chip selects with same timing
    ConfigureEBI_CS(0, baseConfig); // Config object still valid
    ConfigureEBI_CS(1, baseConfig); // Reused for CS1
    ConfigureEBI_CS(2, baseConfig); // Reused for CS2

    iprintf("CS0, CS1, CS2 configured with identical timing\r\n");
}

```

Example 3: Configuration with device-specific variations

```

void ConfigureMixedDevices() {
    ConfigureEBI_CSPin(0);
    ConfigureEBI_CSPin(1);
    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();

    // Fast SRAM on CS0
    EBI_CS_cfg_t fastConfig = {
        .ncs_rd_setup = 2, .nrd_setup = 2,
        .ncs_rd_pulse = 10, .nrd_pulse = 10,
        .nrd_cycles = 15,
        .ncs_wr_setup = 2, .nwe_setup = 2,
        .ncs_wr_pulse = 10, .nwe_pulse = 10,
        .nwe_cycles = 15,
        .tdf_cycles = 2,
        .busWidth = EBI_BUS_WIDTH_16,
        .byteAccess = EBI_BYTE_ACCESS_WRITE,
        .nWait = EBI_NWAIT_DISABLED,
        .wrMode = EBI_WRITE_MODE_NWE,
        .rdMode = EBI_READ_MODE_NRD
    };

    // Slow flash on CS1
    EBI_CS_cfg_t slowConfig = {
        .ncs_rd_setup = 3, .nrd_setup = 3,
        .ncs_rd_pulse = 15, .nrd_pulse = 15,
        .nrd_cycles = 22,
        .ncs_wr_setup = 3, .nwe_setup = 3,
        .ncs_wr_pulse = 20, .nwe_pulse = 20,
        .nwe_cycles = 28,
        .tdf_cycles = 3,
        .busWidth = EBI_BUS_WIDTH_8,
        .byteAccess = EBI_BYTE_ACCESS_SELECT,
        .nWait = EBI_NWAIT_FROZEN,
        .wrMode = EBI_WRITE_MODE_NWE,
        .rdMode = EBI_READ_MODE_NRD
    };

    ConfigureEBI_CS(0, fastConfig);
    ConfigureEBI_CS(1, slowConfig);

    iprintf("CS0: Fast SRAM, CS1: Slow Flash\r\n");
}

```

Example 4: Complete initialization with error checking

```

bool InitializeEBI_Device(uint32_t csNum, const EBI_CS_cfg_t &config) {
    // Validate chip select number
    if (csNum > 3) {
        iprintf("Error: Invalid chip select %lu\r\n", csNum);
        return false;
    }

    // Validate TDF cycles
    if (config.tdf_cycles > 15) {
        iprintf("Error: TDF cycles must be 0-15\r\n");
        return false;
    }

    // Configure the hardware
    ConfigureEBI_CS_Pin(csNum);

    if (config.wrMode == EBI_WRITE_MODE_NWE) {
        ConfigureEBI_NWRP_Pin();
    }

    if (config.rdMode == EBI_READ_MODE_NRD) {
        ConfigureEBI_NRP_Pin();
    }

    // Apply configuration
    ConfigureEBI_CS(csNum, config);

    iprintf("CS%lu configured successfully\r\n", csNum);
    return true;
}

void UserMain(void *pd) {
    init();

    EBI_CS_cfg_t myConfig = {
        .ncs_rd_setup = 2, .nrd_setup = 2,
        .ncs_rd_pulse = 10, .nrd_pulse = 10,
        .nrd_cycles = 15,
        .ncs_wr_setup = 2, .nwe_setup = 2,
        .ncs_wr_pulse = 10, .nwe_pulse = 10,
        .nwe_cycles = 15,
        .tdf_cycles = 2,
        .busWidth = EBI_BUS_WIDTH_16,
        .byteAccess = EBI_BYTE_ACCESS_WRITE,
        .nWait = EBI_NWAIT_DISABLED,
        .wrMode = EBI_WRITE_MODE_NWE,
        .rdMode = EBI_READ_MODE_NRD
    };

    if (InitializeEBI_Device(0, myConfig)) {
        iprintf("Device ready for operations\r\n");
    }

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Example 5: Configuration storage and retrieval

```

// Global storage for configurations
EBI_CS_cfg_t g_csConfigs[4];

void StoreConfiguration(uint32_t csNum, const EBI_CS_cfg_t &config) {
    if (csNum < 4) {
        g_csConfigs[csNum] = config;
        iprintf("Configuration stored for CS%lu\r\n", csNum);
    }
}

void ApplyStoredConfiguration(uint32_t csNum) {
    if (csNum < 4) {
        ConfigureEBI_CS(csNum, g_csConfigs[csNum]);
        iprintf("Applied stored configuration to CS%lu\r\n", csNum);
    }
}

void UserMain(void *pd) {
    init();

    // Create and store configurations
    EBI_CS_cfg_t sramConfig = {
        .ncs_rd_setup = 2, .nrd_setup = 2,
        .ncs_rd_pulse = 10, .nrd_pulse = 10,
        .nrd_cycles = 15,

```

```

        .ncs_wr_setup = 2, .nwe_setup = 2,
        .ncs_wr_pulse = 10, .nwe_pulse = 10,
        .nwe_cycles = 15,
        .tdf_cycles = 2,
        .busWidth = EBI_BUS_WIDTH_16,
        .byteAccess = EBI_BYTE_ACCESS_WRITE,
        .nWait = EBI_NWAIT_DISABLED,
        .wrMode = EBI_WRITE_MODE_NWE,
        .rdMode = EBI_READ_MODE_NRD
    };

    StoreConfiguration(0, sramConfig);

    // Later, apply the stored configuration
    ConfigureEBI_CSPin(0);
    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();
    ApplyStoredConfiguration(0);

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Example 6: Dynamic configuration based on clock frequency

```

EBI_CS_cfg_t CreateTimingConfig(uint32_t ebiClockMHz,
                                uint32_t setupNs,
                                uint32_t pulseNs,
                                uint32_t holdNs)
{
    float nsPerCycle = 1000.0f / ebiClockMHz;

    uint8_t setupCycles = (uint8_t)((setupNs + nsPerCycle - 1) / nsPerCycle);
    uint8_t pulseCycles = (uint8_t)((pulseNs + nsPerCycle - 1) / nsPerCycle);
    uint8_t holdCycles = (uint8_t)((holdNs + nsPerCycle - 1) / nsPerCycle);
    uint16_t totalCycles = setupCycles + pulseCycles + holdCycles;

    return EBI_CS_cfg_t{
        .ncs_rd_setup = setupCycles, .nrd_setup = setupCycles,
        .ncs_rd_pulse = pulseCycles, .nrd_pulse = pulseCycles,
        .nrd_cycles = totalCycles,
        .ncs_wr_setup = setupCycles, .nwe_setup = setupCycles,
        .ncs_wr_pulse = pulseCycles, .nwe_pulse = pulseCycles,
        .nwe_cycles = totalCycles,
        .tdf_cycles = 2,
        .busWidth = EBI_BUS_WIDTH_16,
        .byteAccess = EBI_BYTE_ACCESS_WRITE,
        .nWait = EBI_NWAIT_DISABLED,
        .wrMode = EBI_WRITE_MODE_NWE,
        .rdMode = EBI_READ_MODE_NRD
    };
}

void InitializeDeviceWithTiming() {
    ConfigureEBI_CSPin(0);
    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();

    // Create config for 150MHz clock, 20ns setup, 70ns pulse, 10ns hold
    EBI_CS_cfg_t timing = CreateTimingConfig(150, 20, 70, 10);

    // Apply configuration
    ConfigureEBI_CS(0, timing);

    iprintf("Device configured with calculated timing\r\n");
}

```

14.40.3.2.3.3 ConfigureEBI_CSPin()

```

void ConfigureEBI_CSPin (
    int csNum)
#include <ebi.h>

```

Configure the I/O pin for a given Chip Select for the external data bus.

This function configures the GPIO pin associated with a specific chip select line on the External Bus Interface (EBI). The chip select pin is configured for proper timing and electrical characteristics to control external memory or peripheral devices connected to the data bus.

Parameters

<i>csNum</i>	The chip select number to configure (typically 0-3 depending on platform). Valid range depends on the specific NetBurner platform being used.

Note

This function should be called before attempting to access any external devices connected to the specified chip select line.

Warning

Ensure the chip select number is valid for your platform. Invalid values may result in undefined behavior.

See also

[ConfigureEBI_NWRPin\(\)](#)

[ConfigureEBI_NRDPin\(\)](#)

Expand for Example Usage

Examples

Example 1: Configuring a single chip select for external SRAM

```
void UserMain(void *pd) {
    init();

    // Configure chip select 0 for external SRAM
    ConfigureEBI_CSPin(0);
    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();

    iprintf("EBI CS0 configured for external memory\r\n");

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

Example 2: Configuring multiple chip selects for different peripherals

```
void InitializeExternalBus() {
    // Configure CS0 for external SRAM
    ConfigureEBI_CSPin(0);

    // Configure CS1 for external FPGA
    ConfigureEBI_CSPin(1);

    // Configure CS2 for external display controller
    ConfigureEBI_CSPin(2);

    // Configure control signals shared by all devices
    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();

    iprintf("External bus configured for 3 devices\r\n");
}

void UserMain(void *pd) {
    init();
    InitializeExternalBus();

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

14.40.3.2.3.4 ConfigureEBI_NRDPin()

```
void ConfigureEBI_NRDPin ()
#include <ebi.h>
```

Configure the I/O pin for the active low read (NRD) bus signal.

This function configures the GPIO pin for the NRD (Not Read) signal on the External Bus Interface (EBI). The NRD signal is an active-low control signal that indicates when the bus is performing a read operation from an external device.

Note

This function must be called once during initialization before performing any read operations from external devices on the EBI bus.

The NRD pin is typically shared across all chip selects on the external bus.

See also

[ConfigureEBI_CSPin\(\)](#)

[ConfigureEBI_NWRPin\(\)](#)

Expand for Example Usage**Examples****Example 1: Basic EBI initialization with read capability**

```
void InitializeEBI() {
    // Configure chip select for external device
    ConfigureEBI_CSPin(0);

    // Configure write control signal
    ConfigureEBI_NWRPin();

    // Configure read control signal
    ConfigureEBI_NRDPin();

    iprintf("EBI configured for read/write operations\r\n");
}

void UserMain(void *pd) {
    init();
    InitializeEBI();

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

Example 2: Read-only external memory configuration

```
void InitializeReadOnlyMemory() {
    // Configure CS for ROM device
    ConfigureEBI_CSPin(2);

    // Only need read signal for ROM
    ConfigureEBI_NRDPin();

    iprintf("Read-only EBI device configured on CS2\r\n");
}

void UserMain(void *pd) {
    init();
    InitializeReadOnlyMemory();

    // Device ready for read operations
    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

Example 3: Complete external bus setup with error checking

```
bool InitializeExternalBus(int csNum) {
    if (csNum < 0 || csNum > 3) {
        iprintf("Error: Invalid chip select %d\r\n", csNum);
        return false;
    }

    // Configure all necessary signals
    ConfigureEBI_CSPin(csNum);
    ConfigureEBI_NWRPin();
    ConfigureEBI_NRDPin();

    iprintf("EBI CS%d fully configured\r\n", csNum);
    return true;
}

void UserMain(void *pd) {
    init();

    if (InitializeExternalBus(0)) {
        iprintf("External device ready for operations\r\n");
    }
}
```

```

    }

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

14.40.3.2.3.5 ConfigureEBI_NWRPin()

```

void ConfigureEBI_NWRPin ()
#include <ebi.h>

```

Configure the I/O pin for the active low write/read (NWR) bus signal.

This function configures the GPIO pin for the NWR (Not Write) signal on the External Bus Interface (EBI). The NWR signal is an active-low control signal that indicates when the bus is performing a write operation to an external device.

Note

This function must be called once during initialization before performing any write operations to external devices on the EBI bus.

The NWR pin is typically shared across all chip selects on the external bus.

See also

[ConfigureEBI_CSPin\(\)](#)

[ConfigureEBI_NRDPin\(\)](#)

Expand for Example Usage

Examples

Example 1: Basic EBI initialization with write capability

```

void InitializeEBI() {
    // Configure chip select for external device
    ConfigureEBI_CSPin(0);

    // Configure write control signal
    ConfigureEBI_NWRPin();

    // Configure read control signal
    ConfigureEBI_NRDPin();

    iprintf("EBI configured for read/write operations\r\n");
}

void UserMain(void *pd) {
    init();
    InitializeEBI();

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Example 2: Write-only external bus configuration

```

void InitializeWriteOnlyDevice() {
    // Configure CS for write-only display
    ConfigureEBI_CSPin(1);

    // Only need write signal for this device
    ConfigureEBI_NWRPin();

    iprintf("Write-only EBI device configured on CS1\r\n");
}

void UserMain(void *pd) {
    init();
    InitializeWriteOnlyDevice();

    // Device ready for write operations
    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

14.40.3.2.4 Variable Documentation

14.40.3.2.4.1 busWidth

`EBI_CS_BusWidth_t EBI_CS_cfg_t::busWidth`

Data bus width configuration.

Specifies whether the external device uses 8-bit or 16-bit data transfers. Must match the physical connection and device capabilities.

See also

[EBI_CS_BusWidth_t](#)

14.40.3.2.4.2 byteAccess

`EBI_CS_ByteAccess_t EBI_CS_cfg_t::byteAccess`

Byte access mode for 16-bit bus configurations.

Controls how 8-bit accesses are handled on a 16-bit bus. Only relevant when busWidth is set to `EBI_BUS_WIDTH_16`.

See also

[EBI_CS_ByteAccess_t](#)

Note

This setting is ignored for 8-bit bus configurations.

14.40.3.2.4.3 ncs_rd_pulse

`uint8_t EBI_CS_cfg_t::ncs_rd_pulse`

NCS (Chip Select) pulse width during read operations, in clock cycles. Defines how long the chip select remains asserted during a read.

Formula: NCS pulse length = $(256 \times \text{ncs\_rd\_pulse}[6] + \text{ncs\_rd\_pulse}[5:0])$ clock cycles

Range: 0-63 standard (0-16319 with extended bit)

Note

The pulse width determines the active time of the signal.

14.40.3.2.4.4 ncs_rd_setup

`uint8_t EBI_CS_cfg_t::ncs_rd_setup`

NCS (Chip Select) setup time for read operations, in clock cycles. Defines the time from CS assertion to read signal assertion.

Formula: NCS setup length = $(128 \times \text{ncs\_rd\_setup}[5] + \text{ncs\_rd\_setup}[4:0])$ clock cycles

Range: 0-63 standard (0-8063 with extended bit)

Note

This sets when the chip select activates before the read signal.

14.40.3.2.4.5 ncs_wr_pulse

`uint8_t EBI_CS_cfg_t::ncs_wr_pulse`

NCS (Chip Select) pulse width during write operations, in clock cycles. Defines how long the chip select remains asserted during a write.

Formula: NCS pulse length = $(256 \times \text{ncs\_wr\_pulse}[6] + \text{ncs\_wr\_pulse}[5:0])$ clock cycles

Range: 0-63 standard (0-16319 with extended bit)

14.40.3.2.4.6 ncs_wr_setup

`uint8_t EBI_CS_cfg_t::ncs_wr_setup`

NCS (Chip Select) setup time for write operations, in clock cycles. Defines the time from CS assertion to write signal assertion.

Formula: NCS setup length = $(128 \times \text{ncs\_wr\_setup}[5] + \text{ncs\_wr\_setup}[4:0])$ clock cycles

Range: 0-63 standard (0-8063 with extended bit)

14.40.3.2.4.7 nrd_cycles

`uint16_t EBI_CS_cfg_t::nrd_cycles`

Total read cycle length in clock cycles.

The total read cycle length is the complete duration of a read operation, equal to the sum of setup, pulse, and hold times for NRD and NCS signals.

Formula: Read cycle length = $(\text{nrd\_cycles}[8:7] \times 256 + \text{nrd\_cycles}[6:0])$ clock cycles

Range: 0-127 standard (0-32639 with extended bits)

Calculation: $\text{nrd\_cycles} = \text{ncs\_rd\_setup} + \text{ncs\_rd\_pulse} + \text{hold\_time}$

Note

The hold time is implicit: $\text{Hold} = \text{nrd\_cycles} - (\text{setup} + \text{pulse})$

14.40.3.2.4.8 nrd_pulse

`uint8_t EBI_CS_cfg_t::nrd_pulse`

NRD (Read Enable) pulse width, in clock cycles. Defines how long the read enable signal remains asserted.

Formula: NRD pulse length = $(256 \times \text{nrd\_pulse}[6] + \text{nrd\_pulse}[5:0])$ clock cycles

Range: 0-63 standard (0-16319 with extended bit)

Note

This is typically the minimum access time required by the device.

14.40.3.2.4.9 nrd_setup

`uint8_t EBI_CS_cfg_t::nrd_setup`

NRD (Read Enable) setup time, in clock cycles. Defines the delay from CS assertion to read enable assertion.

Formula: NRD setup length = $(128 \times \text{nrd\_setup}[5] + \text{nrd\_setup}[4:0])$ clock cycles

Range: 0-63 standard (0-8063 with extended bit)

14.40.3.2.4.10 nWait

`EBI_CS_NWait_t EBI_CS_cfg_t::nWait`

NWAIT signal mode for bus cycle extension.

Configures how the external NWAIT signal is used to extend read/write cycles for slow devices. Commonly used with flash memory or variable-speed peripherals.

See also

[EBI_CS_NWait_t](#)

Note

When disabled, all timing is strictly controlled by the configured values.

14.40.3.2.4.11 nwe_cycles

`uint16_t EBI_CS_cfg_t::nwe_cycles`

Total write cycle length in clock cycles.

The total write cycle length is the complete duration of a write operation, equal to the sum of setup, pulse, and hold times for NWE and NCS signals.

Formula: Write cycle length = (nwe_cycles[8:7] × 256 + nwe_cycles[6:0]) clock cycles

Range: 0-127 standard (0-32639 with extended bits)

Calculation: nwe_cycles = ncs_wr_setup + ncs_wr_pulse + hold_time

Note

The hold time is implicit: Hold = nwe_cycles - (setup + pulse)

14.40.3.2.4.12 nwe_pulse

uint8_t EBI_CS_cfg_t::nwe_pulse

NWE (Write Enable) pulse width, in clock cycles. Defines how long the write enable signal remains asserted.

Formula: NWE pulse length = (256 × nwe_pulse[6] + nwe_pulse[5:0]) clock cycles

Range: 0-63 standard (0-16319 with extended bit)

Note

This is typically the minimum write pulse time required by the device.

14.40.3.2.4.13 nwe_setup

uint8_t EBI_CS_cfg_t::nwe_setup

NWE (Write Enable) setup time, in clock cycles. Defines the delay from CS assertion to write enable assertion.

Formula: NWE setup length = (128 × nwe_setup[5] + nwe_setup[4:0]) clock cycles

Range: 0-63 standard (0-8063 with extended bit)

14.40.3.2.4.14 rdMode

EBI_CS_RdMode_t EBI_CS_cfg_t::rdMode

Read control signal selection.

Determines which signal (NCS or NRD) controls read operations from the external device.

See also

[EBI_CS_RdMode_t](#)

[ConfigureEBI_NRDPin\(\)](#)

14.40.3.2.4.15 tdf_cycles

uint8_t EBI_CS_cfg_t::tdf_cycles

Data Float time - number of clock cycles for the external device to release the data after the rising edge of the read controlling signal.

This represents the time the external device needs to tri-state its outputs after a read operation completes. The SMC always provides one full cycle of bus turnaround after this TDF period.

Valid range: 0-15 clock cycles

Note

This prevents bus contention when switching between read and write operations.

A value of 0-2 is typical for fast devices; 3-5 for slower devices.

14.40.3.2.4.16 wrMode

EBI_CS_WrMode_t EBI_CS_cfg_t::wrMode

Write control signal selection.

Determines which signal (NCS or NWE) controls write operations to the external device.

See also

[EBI_CS_WrMode_t](#)

[ConfigureEBI_NWRPin\(\)](#)

14.40.4 SOMRT1061

Topics

- [QTMR \(Quad Timer\)](#)
The QTMR is a peripheral module that contains a set of general-purpose, 16-bit timers. These timers are very flexible and can be used for a wide range of timing, counting, and control tasks, including:
- [SEMC](#)
Smart External Memory Controller functions (SOMRT1061)
- [SOMRT1061 Temperature Monitor](#)
i.MX RT1061 Processor Temperature Monitor

14.40.4.1 Detailed Description

APIs applicable to SOMRT1061

14.40.4.2 QTMR (Quad Timer)

The QTMR is a peripheral module that contains a set of general-purpose, 16-bit timers. These timers are very flexible and can be used for a wide range of timing, counting, and control tasks, including:

Classes

- [struct _qtmr_config](#)
Quad Timer config structure.

Typedefs

- typedef enum [_qtmr_primary_count_source](#) [qtmr_primary_count_source_t](#)
Quad Timer primary clock source selection.
- typedef enum [_qtmr_input_source](#) [qtmr_input_source_t](#)
Quad Timer input sources selection.
- typedef enum [_qtmr_counting_mode](#) [qtmr_counting_mode_t](#)
Quad Timer counting mode selection.
- typedef enum [_qtmr_pwm_out_state](#) [qtmr_pwm_out_state_t](#)
Quad Timer PWM output state.
- typedef enum [_qtmr_output_mode](#) [qtmr_output_mode_t](#)
Quad Timer output mode selection.
- typedef enum [_qtmr_input_capture_edge](#) [qtmr_input_capture_edge_t](#)
Quad Timer input capture edge mode, rising edge, or falling edge.
- typedef enum [_qtmr_preload_control](#) [qtmr_preload_control_t](#)
Quad Timer input capture edge mode, rising edge, or falling edge.
- typedef enum [_qtmr_debug_action](#) [qtmr_debug_action_t](#)
List of Quad Timer run options when in Debug mode.
- typedef enum [_qtmr_interrupt_enable](#) [qtmr_interrupt_enable_t](#)
List of Quad Timer interrupts.
- typedef enum [_qtmr_status_flags](#) [qtmr_status_flags_t](#)
List of Quad Timer flags.
- typedef enum [_qtmr_channel_selection](#) [qtmr_channel_selection_t](#)
List of channel selection.
- typedef enum [_qtmr_dma_enable](#) [qtmr_dma_enable_t](#)
List of Quad Timer DMA enable.
- typedef struct [_qtmr_config](#) [qtmr_config_t](#)
Quad Timer config structure.

Enumerations

- enum `_qtmr_primary_count_source` {
`kQTMR_ClockCounter0InputPin` = 0 , `kQTMR_ClockCounter1InputPin` , `kQTMR_ClockCounter2InputPin` ,
`kQTMR_ClockCounter3InputPin` ,
`kQTMR_ClockCounter0Output` , `kQTMR_ClockCounter1Output` , `kQTMR_ClockCounter2Output` ,
`kQTMR_ClockCounter3Output` ,
`kQTMR_ClockDivide_1` , `kQTMR_ClockDivide_2` , `kQTMR_ClockDivide_4` , `kQTMR_ClockDivide_8` ,
`kQTMR_ClockDivide_16` , `kQTMR_ClockDivide_32` , `kQTMR_ClockDivide_64` , `kQTMR_ClockDivide_128` }
Quad Timer primary clock source selection.
- enum `_qtmr_input_source` { `kQTMR_Counter0InputPin` = 0 , `kQTMR_Counter1InputPin` , `kQTMR_Counter2InputPin` ,
`kQTMR_Counter3InputPin` }
Quad Timer input sources selection.
- enum `_qtmr_counting_mode` {
`kQTMR_NoOperation` = 0 , `kQTMR_PriSrcRiseEdge` , `kQTMR_PriSrcRiseAndFallEdge` , `kQTMR_PriSrcRiseEdgeSecInpHigh` ,
`kQTMR_QuadCountMode` , `kQTMR_PriSrcRiseEdgeSecDir` , `kQTMR_SecSrcTrigPriCnt` , `kQTMR_CascadeCount` }
Quad Timer counting mode selection.
- enum `_qtmr_pwm_out_state` { `kQTMR_PwmLow` = 0 , `kQTMR_PwmHigh` }
Quad Timer PWM output state.
- enum `_qtmr_output_mode` {
`kQTMR_AssertWhenCountActive` = 0 , `kQTMR_ClearOnCompare` , `kQTMR_SetOnCompare` , `kQTMR_ToggleOnCompare` ,
`kQTMR_ToggleOnAltCompareReg` , `kQTMR_SetOnCompareClearOnSecSrcInp` , `kQTMR_SetOnCompareClearOnCountRoll` ,
`kQTMR_EnableGateClock` }
Quad Timer output mode selection.
- enum `_qtmr_input_capture_edge` { `kQTMR_NoCapture` = 0 , `kQTMR_RisingEdge` , `kQTMR_FallingEdge` ,
`kQTMR_RisingAndFallingEdge` }
Quad Timer input capture edge mode, rising edge, or falling edge.
- enum `_qtmr_preload_control` { `kQTMR_NoPreload` = 0 , `kQTMR_LoadOnComp1` , `kQTMR_LoadOnComp2` }
Quad Timer input capture edge mode, rising edge, or falling edge.
- enum `_qtmr_debug_action` { `kQTMR_RunNormalInDebug` = 0U , `kQTMR_HaltCounter` , `kQTMR_ForceOutToZero` ,
`kQTMR_HaltCountForceOutZero` }
List of Quad Timer run options when in Debug mode.
- enum `_qtmr_interrupt_enable` {
`kQTMR_CompareInterruptEnable` = (1U << 0) , `kQTMR_Compare1InterruptEnable` = (1U << 1) ,
`kQTMR_Compare2InterruptEnable` = (1U << 2) , `kQTMR_OverflowInterruptEnable` = (1U << 3) ,
`kQTMR_EdgeInterruptEnable` = (1U << 4) }
List of Quad Timer interrupts.
- enum `_qtmr_status_flags` {
`kQTMR_CompareFlag` = (1U << 0) , `kQTMR_Compare1Flag` = (1U << 1) , `kQTMR_Compare2Flag` = (1U << 2) ,
`kQTMR_OverflowFlag` = (1U << 3) ,
`kQTMR_EdgeFlag` = (1U << 4) }
List of Quad Timer flags.
- enum `_qtmr_channel_selection` { `kQTMR_Channel_0` = 0U , `kQTMR_Channel_1` , `kQTMR_Channel_2` ,
`kQTMR_Channel_3` }
List of channel selection.
- enum `_qtmr_dma_enable` { `kQTMR_InputEdgeFlagDmaEnable` = (1U << 0) , `kQTMR_ComparatorPreload1DmaEnable` =
(1U << 1) , `kQTMR_ComparatorPreload2DmaEnable` = (1U << 2) }
List of Quad Timer DMA enable.

Functions

- status_t [QTMR_SetupPwm](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel, uint32_t pwmFreqHz, uint8_t dutyCyclePercent, bool outputPolarity, uint32_t srcClock_Hz)
Sets up Quad timer module for PWM signal output.
- void [QTMR_SetupInputCapture](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel, [qtmr_input_source_t](#) capturePin, bool inputPolarity, bool reloadOnCapture, [qtmr_input_capture_edge_t](#) captureMode)
Allows the user to count the source clock cycles until a capture event arrives.

Initialization and deinitialization

- void [QTMR_Init](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel, const [qtmr_config_t](#) *config)
Ungates the Quad Timer clock and configures the peripheral for basic operation.
- void [QTMR_Deinit](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel)
Stops the counter and gates the Quad Timer clock.
- void [QTMR_GetDefaultConfig](#) ([qtmr_config_t](#) *config)
Fill in the Quad Timer config struct with the default settings.

Interrupt Interface

- void [QTMR_EnableInterrupts](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel, uint32_t mask)
Enables the selected Quad Timer interrupts.
- void [QTMR_DisableInterrupts](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel, uint32_t mask)
Disables the selected Quad Timer interrupts.
- uint32_t [QTMR_GetEnabledInterrupts](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel)
Gets the enabled Quad Timer interrupts.

Status Interface

- uint32_t [QTMR_GetStatus](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel)
Gets the Quad Timer status flags.
- void [QTMR_ClearStatusFlags](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel, uint32_t mask)
Clears the Quad Timer status flags.

Read and Write the timer period

- void [QTMR_SetTimerPeriod](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel, uint16_t ticks)
Sets the timer period in ticks.
- void [QTMR_SetCompareValue](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel, uint16_t ticks)
Set compare value.

Enable and Disable the Quad Timer DMA

- void [QTMR_EnableDma](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel, uint32_t mask)
Enable the Quad Timer DMA.
- void [QTMR_DisableDma](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel, uint32_t mask)
Disable the Quad Timer DMA.
- void [QTMR_SetPwmOutputToggle](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel, bool idleStatus)
Set PWM output in idle status (high or low).
- uint8_t [QTMR_GetPwmChannelStatus](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel)
Get the PWM channel dutycycle value.
- void [QTMR_SetPwmClockMode](#) (TMR_Type *base, [qtmr_channel_selection_t](#) channel, [qtmr_primary_count_source_t](#) prescaler)
This function set the value of the prescaler on QTimer channels.

Driver version

- #define `FSL_QTMR_DRIVER_VERSION` (MAKE_VERSION(2, 2, 2))

14.40.4.2.1 Detailed Description

The QTMR is a peripheral module that contains a set of general-purpose, 16-bit timers. These timers are very flexible and can be used for a wide range of timing, counting, and control tasks, including:

- Input capture
- Output compare
- PWM (Pulse-Width Modulation) signal generation
- Quadrature decoding

14.40.4.2.2 Macro Definition Documentation**14.40.4.2.2.1 FSL_QTMR_DRIVER_VERSION**

```
#define FSL_QTMR_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))
#include <fsl_qtmr.h>
Version
```

14.40.4.2.3 Typedef Documentation**14.40.4.2.3.1 qtmr_config_t**

```
typedef struct _qtmr_config qtmr_config_t
#include <fsl_qtmr.h>
```

Quad Timer config structure.

This structure holds the configuration settings for the Quad Timer peripheral. To initialize this structure to reasonable defaults, call the [QTMR_GetDefaultConfig\(\)](#) function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

14.40.4.2.4 Enumeration Type Documentation**14.40.4.2.4.1 _qtmr_channel_selection**

```
enum _qtmr_channel_selection
#include <fsl_qtmr.h>
```

List of channel selection.

Enumerator

kQTMR_Channel↵ _0	TMR Channel 0
kQTMR_Channel↵ _1	TMR Channel 1
kQTMR_Channel↵ _2	TMR Channel 2
kQTMR_Channel↵ _3	TMR Channel 3

14.40.4.2.4.2 _qtmr_counting_mode

```
enum _qtmr_counting_mode
#include <fsl_qtmr.h>
```

Quad Timer counting mode selection.

Enumerator

kQTMR_NoOperation	No operation
kQTMR_PriSrcRiseEdge	Count rising edges of primary source
kQTMR_PriSrcRiseAndFallEdge	Count rising and falling edges of primary source
kQTMR_PriSrcRiseEdgeSecInpHigh	Count rise edges of pri SRC while sec inp high active
kQTMR_QuadCountMode	Quadrature count mode, uses pri and sec sources
kQTMR_PriSrcRiseEdgeSecDir	Count rising edges of pri SRC; sec SRC specifies dir
kQTMR_SecSrcTrigPriCnt	Edge of sec SRC trigger primary count until compare
kQTMR_CascadeCount	Cascaded count mode (up/down)

14.40.4.2.4.3 _qtmr_debug_action

```
enum _qtmr_debug_action
```

```
#include <fsl_qtmr.h>
```

List of Quad Timer run options when in Debug mode.

Enumerator

kQTMR_RunNormalInDebug	Continue with normal operation
kQTMR_HaltCounter	Halt counter
kQTMR_ForceOutToZero	Force output to logic 0
kQTMR_HaltCountForceOutZero	Halt counter and force output to logic 0

14.40.4.2.4.4 _qtmr_dma_enable

```
enum _qtmr_dma_enable
```

```
#include <fsl_qtmr.h>
```

List of Quad Timer DMA enable.

Enumerator

kQTMR_InputEdgeFlagDmaEnable	Input Edge Flag DMA Enable.
kQTMR_ComparatorPreload1DmaEnable	Comparator Preload Register 1 DMA Enable.
kQTMR_ComparatorPreload2DmaEnable	Comparator Preload Register 2 DMA Enable.

14.40.4.2.4.5 _qtmr_input_capture_edge

```
enum _qtmr_input_capture_edge
```

```
#include <fsl_qtmr.h>
```

Quad Timer input capture edge mode, rising edge, or falling edge.

Enumerator

kQTMR_NoCapture	Capture is disabled
kQTMR_RisingEdge	Capture on rising edge (IPS=0) or falling edge (IPS=1)
kQTMR_FallingEdge	Capture on falling edge (IPS=0) or rising edge (IPS=1)
kQTMR_RisingAndFallingEdge	Capture on both edges

14.40.4.2.4.6 _qtmr_input_source

```
enum _qtmr_input_source
#include <fsl_qtmr.h>
Quad Timer input sources selection.
```

Enumerator

kQTMR_Counter0InputPin	Use counter 0 input pin
kQTMR_Counter1InputPin	Use counter 1 input pin
kQTMR_Counter2InputPin	Use counter 2 input pin
kQTMR_Counter3InputPin	Use counter 3 input pin

14.40.4.2.4.7 _qtmr_interrupt_enable

```
enum _qtmr_interrupt_enable
#include <fsl_qtmr.h>
List of Quad Timer interrupts.
```

Enumerator

kQTMR_CompareInterruptEnable	Compare interrupt.
kQTMR_Compare1InterruptEnable	Compare 1 interrupt.
kQTMR_Compare2InterruptEnable	Compare 2 interrupt.
kQTMR_OverflowInterruptEnable	Timer overflow interrupt.
kQTMR_EdgeInterruptEnable	Input edge interrupt.

14.40.4.2.4.8 _qtmr_output_mode

```
enum _qtmr_output_mode
#include <fsl_qtmr.h>
Quad Timer output mode selection.
```

Enumerator

kQTMR_AssertWhenCountActive	Assert OFLAG while counter is active
kQTMR_ClearOnCompare	Clear OFLAG on successful compare
kQTMR_SetOnCompare	Set OFLAG on successful compare
kQTMR_ToggleOnCompare	Toggle OFLAG on successful compare
kQTMR_ToggleOnAltCompareReg	Toggle OFLAG using alternating compare registers
kQTMR_SetOnCompareClearOnSecSrcInp	Set OFLAG on compare, clear on sec SRC input edge
kQTMR_SetOnCompareClearOnCountRoll	Set OFLAG on compare, clear on counter rollover
kQTMR_EnableGateClock	Enable gated clock output while count is active

14.40.4.2.4.9 _qtmr_preload_control

```
enum _qtmr_preload_control
#include <fsl_qtmr.h>
Quad Timer input capture edge mode, rising edge, or falling edge.
```

Enumerator

kQTMR_NoPreload	Never preload

Enumerator

kQTMR_LoadOnComp1	Load upon successful compare with value in COMP1
kQTMR_LoadOnComp2	Load upon successful compare with value in COMP2

14.40.4.2.4.10 _qtmr_primary_count_source

```
enum _qtmr_primary_count_source
```

```
#include <fsl_qtmr.h>
```

Quad Timer primary clock source selection.

Enumerator

kQTMR_ClockCounter0InputPin	Use counter 0 input pin
kQTMR_ClockCounter1InputPin	Use counter 1 input pin
kQTMR_ClockCounter2InputPin	Use counter 2 input pin
kQTMR_ClockCounter3InputPin	Use counter 3 input pin
kQTMR_ClockCounter0Output	Use counter 0 output
kQTMR_ClockCounter1Output	Use counter 1 output
kQTMR_ClockCounter2Output	Use counter 2 output
kQTMR_ClockCounter3Output	Use counter 3 output
kQTMR_ClockDivide_1	IP bus clock divide by 1 prescaler
kQTMR_ClockDivide_2	IP bus clock divide by 2 prescaler
kQTMR_ClockDivide_4	IP bus clock divide by 4 prescaler
kQTMR_ClockDivide_8	IP bus clock divide by 8 prescaler
kQTMR_ClockDivide_16	IP bus clock divide by 16 prescaler
kQTMR_ClockDivide_32	IP bus clock divide by 32 prescaler
kQTMR_ClockDivide_64	IP bus clock divide by 64 prescaler
kQTMR_ClockDivide_128	IP bus clock divide by 128 prescaler

14.40.4.2.4.11 _qtmr_pwm_out_state

```
enum _qtmr_pwm_out_state
```

```
#include <fsl_qtmr.h>
```

Quad Timer PWM output state.

Enumerator

kQTMR_PwmLow	The output state of PWM channel is low
kQTMR_PwmHigh	The output state of PWM channel is low

14.40.4.2.4.12 _qtmr_status_flags

```
enum _qtmr_status_flags
```

```
#include <fsl_qtmr.h>
```

List of Quad Timer flags.

Enumerator

kQTMR_CompareFlag	Compare flag

Enumerator

kQTMR_Compare1Flag	Compare 1 flag
kQTMR_Compare2Flag	Compare 2 flag
kQTMR_OverflowFlag	Timer overflow flag
kQTMR_EdgeFlag	Input edge flag

14.40.4.2.5 Function Documentation

14.40.4.2.5.1 QTMR_ClearStatusFlags()

```
void QTMR_ClearStatusFlags (
    TMR_Type * base,
    qtmr_channel_selection_t channel,
    uint32_t mask)
#include <fsl_qtmr.h>
Clears the Quad Timer status flags.
```

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number
<i>mask</i>	The status flags to clear. This is a logical OR of members of the enumeration qtmr_status_flags_t

brief Clears the Quad Timer status flags.

param base Quad Timer peripheral base address param channel Quad Timer channel number param mask The status flags to clear. This is a logical OR of members of the enumeration [qtmr_status_flags_t](#)

14.40.4.2.5.2 QTMR_Deinit()

```
void QTMR_Deinit (
    TMR_Type * base,
    qtmr_channel_selection_t channel)
#include <fsl_qtmr.h>
Stops the counter and gates the Quad Timer clock.
```

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number

brief Stops the counter and gates the Quad Timer clock

param base Quad Timer peripheral base address param channel Quad Timer channel number

14.40.4.2.5.3 QTMR_DisableDma()

```
void QTMR_DisableDma (
    TMR_Type * base,
    qtmr_channel_selection_t channel,
    uint32_t mask)
#include <fsl_qtmr.h>
Disable the Quad Timer DMA.
```

Parameters

<i>base</i>	Quad Timer peripheral base address

<i>channel</i>	Quad Timer channel number
<i>mask</i>	The DMA to enable. This is a logical OR of members of the enumeration qtmr_dma_enable_t

brief Disable the Quad Timer DMA.

param base Quad Timer peripheral base address param channel Quad Timer channel number param mask The DMA to enable. This is a logical OR of members of the enumeration [qtmr_dma_enable_t](#)

14.40.4.2.5.4 QTMR_DisableInterrupts()

```
void QTMR_DisableInterrupts (
    TMR_Type * base,
    qtmr_channel_selection_t channel,
    uint32_t mask)
```

```
#include <fsl_qtmr.h>
```

Disables the selected Quad Timer interrupts.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration qtmr_interrupt_enable_t

brief Disables the selected Quad Timer interrupts

param base Quad Timer peripheral base address param channel Quad Timer channel number param mask The interrupts to enable. This is a logical OR of members of the enumeration [qtmr_interrupt_enable_t](#)

14.40.4.2.5.5 QTMR_EnableDma()

```
void QTMR_EnableDma (
    TMR_Type * base,
    qtmr_channel_selection_t channel,
    uint32_t mask)
```

```
#include <fsl_qtmr.h>
```

Enable the Quad Timer DMA.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number
<i>mask</i>	The DMA to enable. This is a logical OR of members of the enumeration qtmr_dma_enable_t

brief Enable the Quad Timer DMA.

param base Quad Timer peripheral base address param channel Quad Timer channel number param mask The DMA to enable. This is a logical OR of members of the enumeration [qtmr_dma_enable_t](#)

14.40.4.2.5.6 QTMR_EnableInterrupts()

```
void QTMR_EnableInterrupts (
    TMR_Type * base,
    qtmr_channel_selection_t channel,
    uint32_t mask)
```

```
#include <fsl_qtmr.h>
```

Enables the selected Quad Timer interrupts.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration qtmr_interrupt_enable_t

brief Enables the selected Quad Timer interrupts

param base Quad Timer peripheral base address param channel Quad Timer channel number param mask The interrupts to enable. This is a logical OR of members of the enumeration [qtmr_interrupt_enable_t](#)

14.40.4.2.5.7 QTMR_GetDefaultConfig()

```
void QTMR_GetDefaultConfig (
    qtmr_config_t * config)
#include <fsl_qtmr.h>
```

Fill in the Quad Timer config struct with the default settings.

The default values are:

```
config->debugMode = kQTMR_RunNormalInDebug;
config->enableExternalForce = false;
config->enableMasterMode = false;
config->faultFilterCount = 0;
config->faultFilterPeriod = 0;
config->primarySource = kQTMR_ClockDivide_2;
config->secondarySource = kQTMR_Counter0InputPin;
```

Parameters

<i>config</i>	Pointer to user's Quad Timer config structure.

brief Fill in the Quad Timer config struct with the default settings

The default values are: code config->debugMode = kQTMR_RunNormalInDebug; config->enableExternalForce = false; config->enableMasterMode = false; config->faultFilterCount = 0; config->faultFilterPeriod = 0; config->primarySource = kQTMR_ClockDivide_2; config->secondarySource = kQTMR_Counter0InputPin; end-code param config Pointer to user's Quad Timer config structure.

14.40.4.2.5.8 QTMR_GetEnabledInterrupts()

```
uint32_t QTMR_GetEnabledInterrupts (
    TMR_Type * base,
    qtmr_channel_selection_t channel)
#include <fsl_qtmr.h>
```

Gets the enabled Quad Timer interrupts.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [qtmr_interrupt_enable_t](#)

brief Gets the enabled Quad Timer interrupts

param base Quad Timer peripheral base address param channel Quad Timer channel number

return The enabled interrupts. This is the logical OR of members of the enumeration [qtmr_interrupt_enable_t](#)

14.40.4.2.5.9 QTMR_GetPwmChannelStatus()

```
uint8_t QTMR_GetPwmChannelStatus (
    TMR_Type * base,
    qtmr_channel_selection_t channel)
```

```
#include <fsl_qtmr.h>
```

Get the PWM channel dutycycle value.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number

Returns

Current channel dutycycle value.

brief Get the PWM channel dutycycle value.

param base Quad Timer peripheral base address param channel Quad Timer channel number

return Current channel dutycycle value.

14.40.4.2.5.10 QTMR_GetStatus()

```
uint32_t QTMR_GetStatus (
    TMR_Type * base,
    qtmr_channel_selection_t channel)
```

```
#include <fsl_qtmr.h>
```

Gets the Quad Timer status flags.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number

Returns

The status flags. This is the logical OR of members of the enumeration [qtmr_status_flags_t](#)

brief Gets the Quad Timer status flags

param base Quad Timer peripheral base address param channel Quad Timer channel number

return The status flags. This is the logical OR of members of the enumeration [qtmr_status_flags_t](#)

14.40.4.2.5.11 QTMR_Init()

```
void QTMR_Init (
    TMR_Type * base,
    qtmr_channel_selection_t channel,
    const qtmr_config_t * config)
```

```
#include <fsl_qtmr.h>
```

Ungates the Quad Timer clock and configures the peripheral for basic operation.

Note

This API should be called at the beginning of the application using the Quad Timer driver.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number
<i>config</i>	Pointer to user's Quad Timer config structure

brief Ungates the Quad Timer clock and configures the peripheral for basic operation.

note This API should be called at the beginning of the application using the Quad Timer driver.

param base Quad Timer peripheral base address param channel Quad Timer channel number param config Pointer to user's Quad Timer config structure

14.40.4.2.5.12 QTMR_SetCompareValue()

```
void QTMR_SetCompareValue (
    TMR_Type * base,
    qtmr_channel_selection_t channel,
    uint16_t ticks)
```

```
#include <fsl_qtmr.h>
```

Set compare value.

This function sets the value used for comparison with the counter value.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number
<i>ticks</i>	Timer period in units of ticks.

brief Set compare value.

This function sets the value used for comparison with the counter value.

param base Quad Timer peripheral base address param channel Quad Timer channel number param ticks Timer period in units of ticks.

14.40.4.2.5.13 QTMR_SetPwmClockMode()

```
void QTMR_SetPwmClockMode (
    TMR_Type * base,
    qtmr_channel_selection_t channel,
    qtmr_primary_count_source_t prescaler)
```

```
#include <fsl_qtmr.h>
```

This function set the value of the prescaler on QTimer channels.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number
<i>prescaler</i>	Set prescaler value

brief This function set the value of the prescaler on QTimer channels.

param base Quad Timer peripheral base address param channel Quad Timer channel number param prescaler Set prescaler value

14.40.4.2.5.14 QTMR_SetPwmOutputToIdle()

```
void QTMR_SetPwmOutputToIdle (
    TMR_Type * base,
    qtmr_channel_selection_t channel,
    bool idleStatus)
```

```
#include <fsl_qtmr.h>
```

Set PWM output in idle status (high or low).

Note

When the PWM is set again, the counting needs to be restarted.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number
<i>idleStatus</i>	True: PWM output is high in idle status; false: PWM output is low in idle status.

brief Set PWM output in idle status (high or low).

Note: When the PWM is set again, the counting needs to be restarted.

param base Quad Timer peripheral base address param channel Quad Timer channel number param idleStatus True: PWM output is high in idle status; false: PWM output is low in idle status.

14.40.4.2.5.15 QTMR_SetTimerPeriod()

```
void QTMR_SetTimerPeriod (
    TMR_Type * base,
    qtmr_channel_selection_t channel,
    uint16_t ticks)
```

```
#include <fsl_qtmr.h>
```

Sets the timer period in ticks.

Timers counts from initial value till it equals the count value set here. The counter will then reinitialize to the value specified in the Load register.

Note

1. This function will write the time period in ticks to COMP1 or COMP2 register depending on the count direction
2. User can call the utility macros provided in [fsl_common.h](#) to convert to ticks
3. This function supports cases, providing only primary source clock without secondary source clock.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number
<i>ticks</i>	Timer period in units of ticks

brief Sets the timer period in ticks.

Timers counts from initial value till it equals the count value set here. The counter will then reinitialize to the value specified in the Load register.

note

1. This function will write the time period in ticks to COMP1 or COMP2 register depending on the count direction
2. User can call the utility macros provided in [fsl_common.h](#) to convert to ticks
3. This function supports cases, providing only primary source clock without secondary source clock.
4. The load register is reset before the counter is reinitialized to the value specified in the load register.

param base Quad Timer peripheral base address param channel Quad Timer channel number param ticks Timer period in units of ticks

14.40.4.2.5.16 QTMR_SetupInputCapture()

```
void QTMR_SetupInputCapture (
    TMR_Type * base,
    qtmr_channel_selection_t channel,
    qtmr_input_source_t capturePin,
    bool inputPolarity,
    bool reloadOnCapture,
    qtmr_input_capture_edge_t captureMode)
```

```
#include <fsl_qtmr.h>
```

Allows the user to count the source clock cycles until a capture event arrives.

The count is stored in the capture register.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number
<i>capturePin</i>	Pin through which we receive the input signal to trigger the capture
<i>inputPolarity</i>	true: invert polarity of the input signal, false: no inversion
<i>reloadOnCapture</i>	true: reload the counter when an input capture occurs, false: no reload
<i>captureMode</i>	Specifies which edge of the input signal triggers a capture

brief Allows the user to count the source clock cycles until a capture event arrives.

The count is stored in the capture register.

param base Quad Timer peripheral base address param channel Quad Timer channel number param capturePin Pin through which we receive the input signal to trigger the capture param inputPolarity true: invert polarity of the input signal, false: no inversion param reloadOnCapture true: reload the counter when an input capture occurs, false: no reload param captureMode Specifies which edge of the input signal triggers a capture

14.40.4.2.5.17 QTMR_SetupPwm()

```
status_t QTMR_SetupPwm (
    TMR_Type * base,
    qtmr_channel_selection_t channel,
    uint32_t pwmFreqHz,
    uint8_t dutyCyclePercent,
    bool outputPolarity,
    uint32_t srcClock_Hz)
```

```
#include <fsl_qtmr.h>
```

Sets up Quad timer module for PWM signal output.

The function initializes the timer module according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements.

Parameters

<i>base</i>	Quad Timer peripheral base address
<i>channel</i>	Quad Timer channel number
<i>pwmFreqHz</i>	PWM signal frequency in Hz
<i>dutyCyclePercent</i>	PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)
<i>outputPolarity</i>	true: invert polarity of the output signal, false: no inversion
<i>srcClock_Hz</i>	Main counter clock in Hz.

Returns

Returns an error if there was error setting up the signal.

brief Sets up Quad timer module for PWM signal output.

The function initializes the timer module according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements.

param base Quad Timer peripheral base address param channel Quad Timer channel number param pwmFreqHz PWM signal frequency in Hz param dutyCyclePercent PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) param outputPolarity true: invert polarity of the output signal, false: no inversion param srcClock_Hz Main counter clock in Hz.

return Returns an error if there was error setting up the signal.

14.40.4.3 SEMC

Smart External Memory Controller functions (SOMRT1061)

Classes

- struct [SEMC_cfg_t](#)

Configuration structure for SEMC (Smart External Memory Controller) interface.

Enumerations

- enum class [SEMC_BusWidth_t](#) : uint8_t { [SEMC_BusWidth_t::Width_8](#) = 0 , [SEMC_BusWidth_t::Width_16](#) = 1 }
- Data bus width configuration for SEMC interface.*
- enum class [SEMC_AddrMux_t](#) : uint8_t { [SEMC_AddrMux_t::Mux](#) = 0 , [SEMC_AddrMux_t::AdvMux](#) = 1 , [SEMC_AddrMux_t::Bus](#) = 2 }
- Address multiplexing mode configuration.*
- enum class [SEMC_ColAddrWidth_t](#) : uint8_t { [SEMC_ColAddrWidth_t::Width_12](#) = 0 , [SEMC_ColAddrWidth_t::Width_11](#) = 1 , [SEMC_ColAddrWidth_t::Width_10](#) = 2 , [SEMC_ColAddrWidth_t::Width_9](#) = 3 , [SEMC_ColAddrWidth_t::Width_8](#) = 4 , [SEMC_ColAddrWidth_t::Width_7](#) = 5 , [SEMC_ColAddrWidth_t::Width_6](#) = 6 , [SEMC_ColAddrWidth_t::Width_5](#) = 7 , [SEMC_ColAddrWidth_t::Width_4](#) = 8 , [SEMC_ColAddrWidth_t::Width_3](#) = 9 , [SEMC_ColAddrWidth_t::Width_2](#) = 10 , [SEMC_ColAddrWidth_t::Width_0](#) = 12 }
- Column address width configuration for memory devices.*
- enum class [SEMC_ClkMode_t](#) : uint8_t { [SEMC_ClkMode_t::Async](#) = 0 , [SEMC_ClkMode_t::Sync](#) = 1 }
- Clock mode configuration for SEMC interface.*
- enum class [SEMC_Burst_t](#) : uint8_t { [SEMC_Burst_t::Burst_1](#) = 0 , [SEMC_Burst_t::Burst_2](#) = 1 , [SEMC_Burst_t::Burst_4](#) = 2 , [SEMC_Burst_t::Burst_8](#) = 3 , [SEMC_Burst_t::Burst_16](#) = 4 , [SEMC_Burst_t::Burst_32](#) = 5 , [SEMC_Burst_t::Burst_64](#) = 6 }
- Burst transfer length configuration.*

Functions

- void [ConfigureSEMC](#) (uint32_t baseAddr, uint32_t siz, [SEMC_cfg_t](#) &&cfg)
Configure a Chip Select for the external data bus with rvalue reference configuration.
- void [ConfigureSEMC](#) (uint32_t baseAddr, uint32_t siz, [SEMC_cfg_t](#) &cfg)
Configure a Chip Select for the external data bus with lvalue reference configuration.

14.40.4.3.1 Detailed Description

Smart External Memory Controller functions (SOMRT1061)

```
#include< semc.h>
```

14.40.4.3.2 SEMC Overview

The Smart External Memory Controller (SEMC) provides a flexible interface for connecting external parallel memory devices to the i.MX RT1061 processor. The SEMC supports various memory types including SRAM, PSRAM, NOR Flash, and other parallel bus devices.

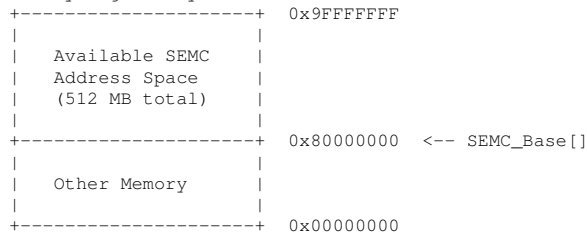
14.40.4.3.2.1 Key Features

- Multiple chip select support (CS0-CS3)
- Configurable data bus widths (8-bit, 16-bit)
- Asynchronous and synchronous clock modes
- Burst transfer support (1, 2, 4, 8, 16, 32, 64 beats)
- Multiplexed and non-multiplexed addressing modes

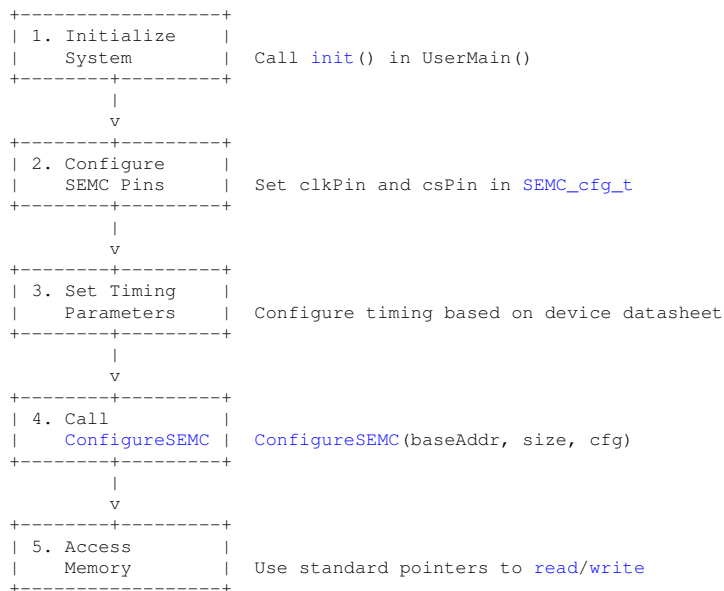
- Independent timing configuration per chip select
- Memory-mapped access via standard pointers
- Address range: 0x80000000 - 0x9FFFFFFF

14.40.4.3.2.2 Memory Address Map

Memory Region Layout:



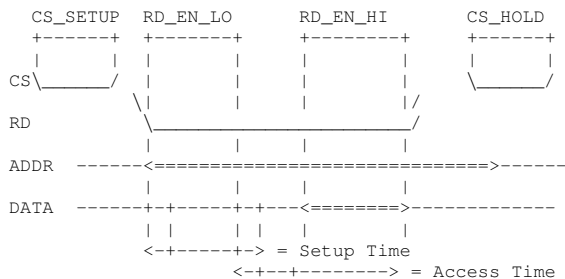
14.40.4.3.2.3 Basic Usage Workflow



14.40.4.3.2.4 Operating Modes

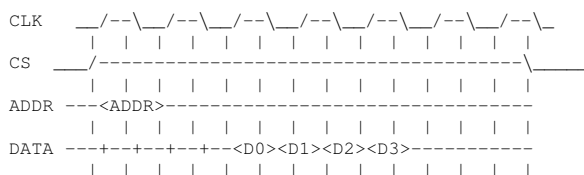
Asynchronous Mode (`SEMC_ClkMode_t::Async`)

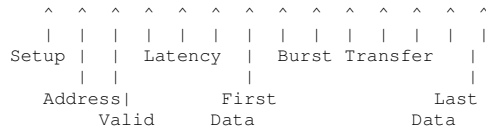
Timing Diagram (`Async` Read):



Synchronous Mode (`SEMC_ClkMode_t::Sync`)

Timing Diagram (`Sync` Burst Read):





14.40.4.3.2.5 Addressing Modes

Multiplexed Mode ([SEMC_AddrMux_t::Mux](#))

Address and Data share the same bus pins:

```
AD_BUS[15:0]  <ADDRESS> <===== DATA =====>
CS            \-----/
ADV           \-----/
```

- Reduces pin count
- Address latched by ADV signal
- Common **for** SRAM and PSRAM devices

Non-Multiplexed Mode ([SEMC_AddrMux_t::Bus](#))

Separate Address and Data buses:

```
ADDR[n:0]     <===== ADDRESS =====>
DATA[15:0]    <===== DATA =====>
CS            \-----/
```

- Faster access times
- More pins required
- Better **for** high-performance applications

14.40.4.3.2.6 Timing Calculation

All timing values are specified in SEMC clock cycles. The actual time depends on the SEMC clock frequency configured in your system.

Example: SEMC Clock = 100 MHz (10 ns per cycle)

If device datasheet specifies:

- Address Setup Time: 15 ns minimum
- Read Enable Low: 30 ns minimum
- Read Enable High: 50 ns minimum

Calculate cycles needed:

```
addr_setup = ceil(15 ns / 10 ns) - 1 = 2 - 1 = 1
rd_en_lo   = ceil(30 ns / 10 ns) - 1 = 3 - 1 = 2
rd_en_hi   = ceil(50 ns / 10 ns) - 1 = 5 - 1 = 4
```

Note: Many timing parameters are (n+1) cycles, so subtract 1 from the calculated value.

14.40.4.3.2.7 Configuration Structure

The [SEMC_cfg_t](#) structure uses unions to provide both hardware register access and structured field access:

Memory Layout:

```
+-----+ CR0 (Control Register 0)
| busWidth      :1| Bus width selection
| clkMode       :1| Clock mode (Async/Sync)
| RESERVED      :2|
| burstLen      :3| Burst length
| RESERVED      :1|
| muxMode       :2| Address multiplexing mode
| advPol        :1| ADV signal polarity
| advHoldLvl    :1| ADV hold level
| colAddrWidth  :4| Column address width
+-----+

+-----+ CR1 (Control Register 1)
| cs_setup      :4| Chip select setup time (n+1)
| cs_hold_min   :4| Chip select hold time (n+1)
| addr_setup    :4| Address setup time (n+1)
| addr_hold     :4| Address hold time
| wr_en_lo      :4| Write enable low time (n+1)
| wr_en_hi      :4| Write enable high time (n+1)
| rd_en_lo      :4| Read enable low time (n+1)
| rd_en_hi      :4| Read enable high time (n+1)
+-----+

+-----+ CR2 (Control Register 2)
| wr_dat_setup  :4| Write data setup (n+1)
| wr_dat_hold   :4| Write data hold (n)
| addr_wr_hold  :4| Address write hold (n+1)
```

```

| turnaround      :4| Turnaround time (n)
| latency         :4| Read latency (n)
| rd_cycle        :4| Read cycle time (n+1)
| cs_min_intv     :4| CS minimum interval (n+1)
| rd_hold         :4| Read hold time (n)
+-----+

+-----+
| clkPin (PinIO) | Clock pin assignment
+-----+

+-----+
| csPin (PinIO)  | Chip select pin assignment
+-----+

```

14.40.4.3.2.8 Multiple Chip Select Configuration

Memory Map Example with Multiple CS:

```

+-----+ 0x9FFFFFFF
| Unused Space |
+-----+ 0x80180000
| CS2: 16-bit   | ConfigureSEMC(0x80100000, 0x80000, cfg2)
| Flash (512 KB)|
+-----+ 0x80100000
| CS1: 8-bit    | ConfigureSEMC(0x80080000, 0x80000, cfg1)
| SRAM (512 KB) |
+-----+ 0x80080000
| CS0: 16-bit   | ConfigureSEMC(0x80000000, 0x80000, cfg0)
| PSRAM (512 KB)|
+-----+ 0x80000000 <-- SEMC_Base[]

```

Requirements:

- Base addresses must be aligned to size boundary
- Memory regions must not overlap
- Each CS can have independent timing configuration
- Different CS pins must be used **for each** region

14.40.4.3.2.9 Common Configuration Examples

Typical Async SRAM Configuration:

Device Type: IS61WV51216BLL (512K x 16-bit SRAM)
Access Time: 10ns
SEMC Clock: 100 MHz

```

SEMC_cfg_t cfg {
    .busWidth = SEMC_BusWidth_t::Width_16,
    .clkMode = SEMC_ClkMode_t::Async,
    .burstLen = SEMC_Burst_t::Burst_1,
    .muxMode = SEMC_AddrMux_t::Mux,
    .colAddrWidth = SEMC_ColAddrWidth_t::Width_0,
    .cs_setup = 1,           // 2 cycles = 20ns
    .cs_hold_min = 1,       // 2 cycles = 20ns
    .addr_setup = 1,        // 2 cycles = 20ns
    .addr_hold = 1,         // 2 cycles = 20ns
    .wr_en_lo = 2,          // 3 cycles = 30ns
    .wr_en_hi = 2,          // 3 cycles = 30ns
    .rd_en_lo = 2,          // 3 cycles = 30ns
    .rd_en_hi = 4,          // 5 cycles = 50ns
};

```

Typical Sync Burst PSRAM Configuration:

Device Type: IS66/67WVE4M16EBLL (Synchronous PSRAM)
Clock: 100 MHz
Latency: 3 cycles

```

SEMC_cfg_t cfg {
    .busWidth = SEMC_BusWidth_t::Width_16,
    .clkMode = SEMC_ClkMode_t::Sync,
    .burstLen = SEMC_Burst_t::Burst_8,
    .muxMode = SEMC_AddrMux_t::Mux,
    .colAddrWidth = SEMC_ColAddrWidth_t::Width_8,
    .cs_setup = 2,           // CS setup before clock
    .cs_hold_min = 2,       // CS hold after clock
    .addr_setup = 1,        // Address setup
    .addr_hold = 1,         // Address hold
    .wr_dat_setup = 1,      // Write data setup
    .wr_dat_hold = 0,       // Write data hold
    .latency = 3,           // Read latency cycles
};

```

```

        .rd_cycle = 5,          // Total read cycle time
        .cs_min_intv = 1,      // Minimum CS interval
        .rd_hold = 1,         // Read data hold
};

```

14.40.4.3.2.10 Memory Access Patterns

Direct Memory Access:

```

// Configure SEMC for external SRAM
ConfigureSEMC(0x80000000, 0x100000, cfg);

// Access as standard memory
uint8_t *bytePtr = (uint8_t *)0x80000000;
uint16_t *wordPtr = (uint16_t *)0x80000000;
uint32_t *dwordPtr = (uint32_t *)0x80000000;

// Byte access
bytePtr[0] = 0x42;
uint8_t value = bytePtr[0];

// Word access (16-bit)
wordPtr[0] = 0x1234;
uint16_t word = wordPtr[0];

// Double-word access (32-bit)
dwordPtr[0] = 0xDEADBEEF;
uint32_t dword = dwordPtr[0];

```

Bulk Data Transfer:

```

uint8_t *extMem = (uint8_t *)0x80000000;
uint8_t buffer[1024];

// Write block of data
memcpy(extMem, buffer, sizeof(buffer));

// Read block of data
memcpy(buffer, extMem, sizeof(buffer));

// Pattern fill
memset(extMem, 0xFF, 1024);

```

Structure Access:

```

struct DataRecord {
    uint32_t id;
    uint16_t value;
    uint8_t status;
    uint8_t flags;
};

// Place structure array in external memory
DataRecord *records = (DataRecord *)0x80000000;

// Access structure members
records[0].id = 1234;
records[0].value = 0x5678;
records[0].status = 0x01;

// Iterate through records
for (int i = 0; i < 100; i++) {
    records[i].id = i;
    records[i].value = i * 2;
}

```

14.40.4.3.2.11 Troubleshooting Guide

Common Issues and Solutions:

```

Issue: Data corruption or incorrect reads
+-----+
| Check: |
| 1. Timing parameters match datasheet |
| 2. SEMC clock frequency is correct |
| 3. Setup/hold times meet minimum specs |
| 4. Signal integrity (scope waveforms) |
| 5. Power supply stability |
+-----+

Issue: System crashes when accessing memory
+-----+
| Check: |
| 1. Base address within valid range |
| 2. Size is power of 2 and aligned |
| 3. No address overlap with other CS |
| 4. ConfigureSEMC called before access |
| 5. Pins configured correctly |
+-----+

```

```

+-----+
Issue: Intermittent failures
+-----+
| Check:                                     |
| 1. Add margin to timing parameters         |
| 2. Check turnaround time settings          |
| 3. Verify CS hold time adequate           |
| 4. Temperature effects on device          |
| 5. Bus contention with other devices      |
+-----+

```

14.40.4.3.2.12 Best Practices

- Always add timing margin (10-20%) beyond minimum datasheet specifications
- Initialize SEMC during system startup, before any memory access
- Verify timing calculations against datasheet at your SEMC clock frequency
- Use const for read-only data stored in external Flash memory
- Consider cache implications for frequently accessed data
- Test memory thoroughly after configuration changes
- Document the specific memory device part number and timing calculations
- Use volatile keyword when hardware registers might change externally
- Align data structures to natural boundaries for best performance
- Implement error detection/correction if data integrity is critical

14.40.4.3.2.13 Performance Considerations

Access Performance Comparison (approximate):

Memory Type	Typical Access	Throughput
Internal SRAM	1 cycle	Maximum
External SRAM (Async)	5-10 cycles	Medium
External SRAM (Sync)	3-6 cycles	High
External Flash (Async)	10-20 cycles	Low

Burst Mode Benefits:

- First access: Full latency
- Subsequent accesses: 1 cycle each
- Best for sequential data access
- Up to 8x performance for burst-8 mode

14.40.4.3.2.14 Memory Region Symbols

The linker provides symbols for SEMC memory regions:

```

extern uint8_t SEMC_Base[];      // Start of SEMC address space
extern uint8_t _s_data_bus[];    // Start of data bus region
extern uint8_t _e_data_bus[];    // End of data bus region

```

Usage example:

```

void CheckSEMCRegion(void *ptr) {
    uintptr_t addr = (uintptr_t)ptr;
    uintptr_t base = (uintptr_t)SEMC_Base;

    if (addr >= base && addr < (base + 0x20000000)) {
        iprintf("Address is in SEMC region\n");
    }
}

```

Note

For detailed timing parameter descriptions, see [SEMC_cfg_t](#) structure documentation.

Consult your external memory device datasheet for specific timing requirements.

SEMC configuration should be done early in UserMain() before accessing external memory.

See also

[ConfigureSEMC\(\)](#)
[SEMC_cfg_t](#)

14.40.4.3.3 Enumeration Type Documentation

14.40.4.3.3.1 SEMC_AddrMux_t

```
enum class SEMC_AddrMux_t : uint8_t [strong]
#include <semc.h>
```

Address multiplexing mode configuration.

Determines how address signals are presented on the bus interface. The mode affects pin usage, timing characteristics, and compatibility with different memory devices.

Note

Multiplexed modes reduce pin count by sharing address and data pins.

Non-multiplexed mode provides fastest access but requires separate address pins.

Consult your memory device datasheet for supported addressing modes.

Warning

Bus mode (non-multiplexed) performs reads as AD-Mux due to hardware limitation.

See also

[SEMC_cfg_t::muxMode](#)

Enumerator

Mux	Multiplexed address/data mode. Address and data share the same pins. Requires ADV signal to latch address.
AdvMux	Advanced multiplexed mode. Enhanced multiplexing with additional control signals for complex timing.
Bus	Non-multiplexed bus mode (write only). Separate address and data buses. Reads are performed as AD-Mux. Best for write-heavy applications.

14.40.4.3.3.2 SEMC_Burst_t

```
enum class SEMC_Burst_t : uint8_t [strong]
#include <semc.h>
```

Burst transfer length configuration.

Specifies the number of data beats (words) transferred in a single burst operation. Burst transfers improve throughput for sequential memory accesses by eliminating address overhead after the first access.

Note

Burst mode is most effective in synchronous clock mode ([SEMC_ClkMode_t::Sync](#)).

The external memory device must support the selected burst length.

Larger burst sizes provide better performance for sequential access patterns.

Single-beat (Burst_1) effectively disables burst mode.

Burst Performance Example ([Sync](#) Mode, 100 MHz SEMC Clock):

```
Single Access (Burst_1):
+-----+-----+-----+-----+
| Addr | Data | Addr | Data | = 4 cycles for 2 words
+-----+-----+-----+-----+
```



```

Throughput: 50 million words/sec

Burst-4 Access:
+-----+-----+-----+-----+-----+
| Addr  | Data0  | Data1  | Data2  | Data3  | = 5 cycles for 4 words
+-----+-----+-----+-----+-----+
Throughput: 80 million words/sec (1.6x faster)

Burst-8 Access:
+-----+-----+-----+-----+
| Addr  | Data0  | ...   | Data7  |      = 9 cycles for 8 words
+-----+-----+-----+-----+
Throughput: 88.9 million words/sec (1.78x faster)

Performance Improvement:
- Burst_2: ~1.3x faster than single
- Burst_4: ~1.6x faster than single
- Burst_8: ~1.8x faster than single
- Burst_16: ~1.9x faster than single
- Burst_32: ~1.94x faster than single

```

Warning

Verify your memory device supports the selected burst length before use.
Some devices may have alignment requirements for burst addresses.

See also

[SEMC_cfg_t::burstLen](#)
[SEMC_ClkMode_t::Sync](#)

Enumerator

Burst_1	Single-beat transfer. No burst mode. Each access requires full address cycle. Use for random access or non-burst devices.
Burst_2	2-beat burst transfer. Address once, transfer 2 consecutive data words. Minimal burst mode.
Burst_4	4-beat burst transfer. Address once, transfer 4 consecutive data words. Common burst length for many devices.
Burst_8	8-beat burst transfer. Address once, transfer 8 consecutive data words. Optimal for cache line fills (32 bytes with 16-bit bus).
Burst_16	16-beat burst transfer. Address once, transfer 16 consecutive data words. High-performance sequential access.
Burst_32	32-beat burst transfer. Address once, transfer 32 consecutive data words. Maximum sequential performance.
Burst_64	64-beat burst transfer. Address once, transfer 64 consecutive data words. Extreme burst length for specialized applications.

14.40.4.3.3.3 SEMC_BusWidth_t

```

enum class SEMC_BusWidth_t : uint8_t [strong]
#include <semc.h>

```

Data bus width configuration for SEMC interface.

Specifies the width of the data bus connecting the processor to the external memory device. The bus width must match the physical memory device's data interface width.

Note

16-bit bus width provides higher throughput but requires more pins.
8-bit bus width uses fewer pins but requires more bus cycles for wide data transfers.
Verify your hardware PCB design supports the selected bus width.

See also

[SEMC_cfg_t::busWidth](#)

Enumerator

Width_8	8-bit data bus (D[7:0]). Suitable for 8-bit memory devices or low pin-count designs.
Width_16	16-bit data bus (D[15:0]). Suitable for 16-bit memory devices. Provides 2x throughput compared to 8-bit.

14.40.4.3.3.4 SEMC_ClkMode_t

```
enum class SEMC_ClkMode_t : uint8_t [strong]
```

```
#include <semc.h>
```

Clock mode configuration for SEMC interface.

Determines whether the memory interface operates in asynchronous or synchronous mode. This affects timing requirements, performance characteristics, and which timing parameters in [SEMC_cfg_t](#) are used.

Note

Asynchronous mode is simpler and works with most SRAM and NOR Flash devices.

Synchronous mode provides better performance with SDRAM and synchronous PSRAM.

Different timing parameters apply depending on the selected mode.

Mode Comparison:

Characteristic	Async Mode	Sync Mode
Timing	Fixed delays	Clock-synchronized
Performance	Moderate	Higher
Burst Support	Limited	Full support
Device Types	SRAM, NOR Flash	SDRAM, Sync PSRAM
Complexity	Simple	More complex

Timing Parameters Used:

Async Mode:

- cs_setup, cs_hold_min
- addr_setup, addr_hold
- wr_en_lo, wr_en_hi
- rd_en_lo, rd_en_hi
- addr_wr_hold
- turnaround

Sync Mode:

- cs_setup, cs_hold_min
- addr_setup, addr_hold
- wr_dat_setup, wr_dat_hold
- latency
- rd_cycle
- rd_hold

See also

[SEMC_cfg_t::clkMode](#)

[SEMC_cfg_t](#) (for timing parameter details)

Enumerator

Async	Asynchronous mode. Timing based on fixed delays. No clock synchronization required. Best for SRAM and asynchronous NOR Flash devices.
Sync	Synchronous mode. All signals synchronized to SEMC clock. Supports burst transfers. Best for SDRAM and synchronous PSRAM devices.

14.40.4.3.3.5 SEMC_ColAddrWidth_t

```
enum class SEMC_ColAddrWidth_t : uint8_t [strong]
```

```
#include <semc.h>
```

Column address width configuration for memory devices.

Specifies the number of column address bits used by the external memory device. This setting determines how many address pins are configured and affects the memory organization and access patterns.

Note

Width_0 disables address pin configuration - use for simple SRAM/Flash without column addressing.

For SDRAM and burst-capable devices, match this to the device's column address width.

Incorrect width settings can cause address aliasing or inaccessible memory regions.

Example Device Specifications:

Memory Device	Column Addresses	Use Setting
Simple SRAM/Flash	N/A	Width_0
SDRAM (8-bit columns)	A[7:0]	Width_8
SDRAM (10-bit columns)	A[9:0]	Width_10
SDRAM (12-bit columns)	A[11:0]	Width_12

See also

[SEMC_cfg_t::colAddrWidth](#)

Enumerator

Width_12	12-bit column address (A[11:0]). Maximum column address width, suitable for large SDRAM devices.
Width_11	11-bit column address (A[10:0]). Common for medium-density SDRAM.
Width_10	10-bit column address (A[9:0]). Common for standard SDRAM configurations.
Width_9	9-bit column address (A[8:0]). Used in some SDRAM and synchronous memory devices.
Width_8	8-bit column address (A[7:0]). Common for smaller SDRAM and burst-capable PSRAM.
Width_7	7-bit column address (A[6:0]). Used in specialized or smaller memory configurations.
Width_6	6-bit column address (A[5:0]). Minimal column addressing for compact devices.
Width_5	5-bit column address (A[4:0]). Very limited column addressing.
Width_4	4-bit column address (A[3:0]). Minimal configuration for small burst regions.
Width_3	3-bit column address (A[2:0]). Extremely limited addressing, rarely used.
Width_2	2-bit column address (A[1:0]). Minimal burst addressing capability.
Width_0	No column addressing. Do not configure address pins for column mode. Use for simple SRAM/NOR Flash without column addressing requirements.

14.40.4.3.4 Function Documentation**14.40.4.3.4.1 ConfigureSEMC() [1/2]**

```
void ConfigureSEMC (
    uint32_t baseAddr,
    uint32_t siz,
    SEMC_cfg_t && cfg)
```

```
#include <semc.h>
```

Configure a Chip Select for the external data bus with rvalue reference configuration.

This function configures the Smart External Memory Controller (SEMC) for a specific chip select with the provided memory region and timing parameters. This overload accepts an rvalue reference, making it suitable for temporary configuration objects.

The SEMC controller manages external memory devices connected to the RT1061's parallel bus interface. Configuration includes timing parameters, bus width, addressing modes, and signal characteristics required by the external memory device.

Parameters

<i>baseAddr</i>	The base address to assign to the configured chip select. This address must be within the valid SEMC memory region and properly aligned based on the size parameter. Typical range: 0x80000000 - 0x9FFFFFFF for SEMC CS0-CS3.

Parameters

<i>size</i>	The size (in bytes) of the memory region assigned to this chip select. Must be a power of 2 and properly aligned with the base address. Common values include 64KB (0x10000), 128KB (0x20000), 256KB (0x40000), etc.
<i>cfg</i>	Rvalue reference to the SEMC_cfg_t structure containing the configuration parameters. This allows passing temporary configuration objects efficiently.

Note

The base address and size must not overlap with other configured chip selects.

Ensure all timing parameters in the *cfg* structure match your external device's datasheet requirements.

The *clkPin* and *csPin* in the configuration must be properly initialized before calling this function.

Warning

Incorrect timing parameters can cause data corruption or system instability.

This function should be called during system initialization before accessing the external memory region.

Expand for Example Usage

Examples

Basic SRAM Configuration (Async Mode, 8-bit)

```
#include <semc.h>

void UserMain(void *pd) {
    init();

    // Configure SEMC for an 8-bit async SRAM at 0x80000000
    ConfigureSEMC(
        0x80000000,    // Base address
        0x40000,      // 256KB size
        SEMC_cfg_t{
            .busWidth = SEMC_BusWidth_t::Width_8,
            .clkMode = SEMC_ClkMode_t::Async,
            .burstLen = SEMC_Burst_t::Burst_1,
            .muxMode = SEMC_AddrMux_t::Mux,
            .colAddrWidth = SEMC_ColAddrWidth_t::Width_0,
            .cs_setup = 1,
            .cs_hold_min = 1,
            .addr_setup = 1,
            .addr_hold = 1,
            .wr_en_lo = 2,
            .wr_en_hi = 2,
            .rd_en_lo = 2,
            .rd_en_hi = 4,
            .clkPin = Pins[10], // Example clock pin
            .csPin = Pins[11]  // Example CS pin
        }
    );

    // Access external memory
    uint8_t *extMem = (uint8_t *)0x80000000;
    extMem[0] = 0x42;
    iprintf("Value written: 0x%02X\n", extMem[0]);
}
```

16-bit Synchronous Burst Memory

```
#include <semc.h>

void ConfigureSyncMemory() {
    // Configure SEMC for 16-bit synchronous burst memory
    ConfigureSEMC(
        0x80000000,    // Base address
        0x100000,      // 1MB size
        SEMC_cfg_t{
            .busWidth = SEMC_BusWidth_t::Width_16,
            .clkMode = SEMC_ClkMode_t::Sync,
            .burstLen = SEMC_Burst_t::Burst_8,
            .muxMode = SEMC_AddrMux_t::Mux,
            .colAddrWidth = SEMC_ColAddrWidth_t::Width_8,
            .cs_setup = 2,
            .cs_hold_min = 2,
            .addr_setup = 1,
        }
    );
}
```

```

        .addr_hold = 1,
        .wr_dat_setup = 1,
        .wr_dat_hold = 0,
        .latency = 3,
        .rd_cycle = 5,
        .cs_min_intv = 1,
        .rd_hold = 1,
        .clkPin = Pins[10],
        .csPin = Pins[11]
    }
};

```

Multiple Chip Selects with Non-Multiplexed Addressing

```

#include <semc.h>

void ConfigureMultipleMemories() {
    // First memory region - 8-bit SRAM
    ConfigureSEMC(
        0x80000000,
        0x80000, // 512KB
        SEMC_cfg_t{
            .busWidth = SEMC_BusWidth_t::Width_8,
            .clkMode = SEMC_ClkMode_t::Async,
            .muxMode = SEMC_AddrMux_t::Bus,
            .colAddrWidth = SEMC_ColAddrWidth_t::Width_0,
            .cs_setup = 1,
            .cs_hold_min = 1,
            .addr_setup = 2,
            .addr_hold = 2,
            .wr_en_lo = 3,
            .wr_en_hi = 3,
            .rd_en_lo = 3,
            .rd_en_hi = 5,
            .turnaround = 2,
            .clkPin = Pins[10],
            .csPin = Pins[11]
        }
    );

    // Second memory region - 16-bit Flash
    ConfigureSEMC(
        0x80080000, // Non-overlapping base address
        0x100000, // 1MB
        SEMC_cfg_t{
            .busWidth = SEMC_BusWidth_t::Width_16,
            .clkMode = SEMC_ClkMode_t::Async,
            .muxMode = SEMC_AddrMux_t::Mux,
            .colAddrWidth = SEMC_ColAddrWidth_t::Width_0,
            .cs_setup = 2,
            .cs_hold_min = 2,
            .addr_setup = 3,
            .addr_hold = 3,
            .wr_en_lo = 5,
            .wr_en_hi = 5,
            .rd_en_lo = 6,
            .rd_en_hi = 8,
            .clkPin = Pins[10],
            .csPin = Pins[12] // Different CS pin
        }
    );
}

```

See also

[ConfigureSEMC\(uint32_t, uint32_t, SEMC_cfg_t&\)](#)
[SEMC_cfg_t](#)

14.40.4.3.4.2 ConfigureSEMC() [2/2]

```

void ConfigureSEMC (
    uint32_t baseAddr,
    uint32_t siz,
    SEMC_cfg_t & cfg)

```

```
#include <semc.h>
```

Configure a Chip Select for the external data bus with lvalue reference configuration.

This function configures the Smart External Memory Controller (SEMC) for a specific chip select with the provided memory region and timing parameters. This overload accepts an lvalue reference, making it suitable for reusing existing configuration objects.

The SEMC controller manages external memory devices connected to the RT1061's parallel bus interface. Configuration includes timing parameters, bus width, addressing modes, and signal characteristics required by the external memory device.

Parameters

<i>baseAddr</i>	The base address to assign to the configured chip select. This address must be within the valid SEMC memory region and properly aligned based on the size parameter. Typical range: 0x80000000 - 0x9FFFFFFF for SEMC CS0-CS3.
<i>sz</i>	The size (in bytes) of the memory region assigned to this chip select. Must be a power of 2 and properly aligned with the base address. Common values include 64KB (0x10000), 128KB (0x20000), 256KB (0x40000), etc.
<i>cfg</i>	Lvalue reference to the SEMC_cfg_t structure containing the configuration parameters. This allows reusing configuration objects across multiple calls.

Note

The base address and size must not overlap with other configured chip selects.

Ensure all timing parameters in the *cfg* structure match your external device's datasheet requirements.

The *clkPin* and *csPin* in the configuration must be properly initialized before calling this function.

Warning

Incorrect timing parameters can cause data corruption or system instability.

This function should be called during system initialization before accessing the external memory region.

Expand for Example Usage

Examples

Reusing Configuration for Multiple Memory Banks

```
#include <semc.h>

void ConfigureMemoryBanks() {
    // Create a reusable configuration for identical memory chips
    SEMC_cfg_t sramConfig{
        .busWidth = SEMC_BusWidth_t::Width_8,
        .clkMode = SEMC_ClkMode_t::Async,
        .burstLen = SEMC_Burst_t::Burst_1,
        .muxMode = SEMC_AddrMux_t::Mux,
        .colAddrWidth = SEMC_ColAddrWidth_t::Width_0,
        .cs_setup = 1,
        .cs_hold_min = 1,
        .addr_setup = 1,
        .addr_hold = 1,
        .wr_en_lo = 2,
        .wr_en_hi = 2,
        .rd_en_lo = 2,
        .rd_en_hi = 4,
        .clkPin = Pins[10]
    };

    // Configure first bank with CS0
    sramConfig.csPin = Pins[11];
    ConfigureSEMC(0x80000000, 0x40000, sramConfig);

    // Configure second bank with CS1 (reusing same timing)
    sramConfig.csPin = Pins[12];
    ConfigureSEMC(0x80040000, 0x40000, sramConfig);

    iprintf("Two memory banks configured\n");
}
```

Conditional Configuration Based on Hardware Detection

```

#include <semc.h>

SEMC_cfg_t GetMemoryConfig(bool isFastMemory) {
    SEMC_cfg_t cfg{
        .busWidth = SEMC_BusWidth_t::Width_16,
        .clkMode = SEMC_ClkMode_t::Async,
        .muxMode = SEMC_AddrMux_t::Mux,
        .colAddrWidth = SEMC_ColAddrWidth_t::Width_0,
        .clkPin = Pins[10],
        .csPin = Pins[11]
    };

    if (isFastMemory) {
        // Fast memory timing
        cfg.cs_setup = 1;
        cfg.rd_en_hi = 3;
        cfg.wr_en_hi = 3;
    } else {
        // Slow memory timing
        cfg.cs_setup = 2;
        cfg.rd_en_hi = 6;
        cfg.wr_en_hi = 6;
    }

    return cfg;
}

void UserMain(void *pd) {
    init();

    bool fastHardware = DetectMemorySpeed();
    SEMC_cfg_t config = GetMemoryConfig(fastHardware);

    // Use lvalue reference overload
    ConfigureSEMC(0x80000000, 0x80000, config);

    iprintf("Memory configured for %s device\n",
        fastHardware ? "fast" : "slow");
}

```

Dynamic Configuration with User Input

```

#include <semc.h>

void ConfigureUserMemory() {
    // Base configuration template
    SEMC_cfg_t memConfig{
        .busWidth = SEMC_BusWidth_t::Width_8,
        .clkMode = SEMC_ClkMode_t::Async,
        .muxMode = SEMC_AddrMux_t::Mux,
        .colAddrWidth = SEMC_ColAddrWidth_t::Width_0,
        .cs_setup = 1,
        .cs_hold_min = 1,
        .addr_setup = 1,
        .addr_hold = 1,
        .clkPin = Pins[10],
        .csPin = Pins[11]
    };

    // Adjust timing based on configuration file or user settings
    int readSpeed = ReadConfigValue("memory.read_speed");
    memConfig.rd_en_lo = readSpeed;
    memConfig.rd_en_hi = readSpeed + 2;

    int writeSpeed = ReadConfigValue("memory.write_speed");
    memConfig.wr_en_lo = writeSpeed;
    memConfig.wr_en_hi = writeSpeed + 2;

    uint32_t baseAddr = ReadConfigValue("memory.base_addr");
    uint32_t size = ReadConfigValue("memory.size");

    // Configure with modified settings
    ConfigureSEMC(baseAddr, size, memConfig);

    iprintf("Memory configured from user settings\n");
}

```

See also

[ConfigureSEMC\(uint32_t, uint32_t, SEMC_cfg_t&&\)](#)

[SEMC_cfg_t](#)

14.40.4.4 SOMRT1061 Temperature Monitor

i.MX RT1061 Processor Temperature Monitor

Files

- file [fsl_tempmon.h](#)

Classes

- struct [_tempmon_config](#)
TEMPMON temperature structure.

Typedefs

- typedef struct [_tempmon_config](#) [tempmon_config_t](#)
TEMPMON temperature structure.
- typedef enum [_tempmon_alarm_mode](#) [tempmon_alarm_mode](#)
TEMPMON alarm mode.

Enumerations

- enum [_tempmon_alarm_mode](#) { [kTEMPMON_HighAlarmMode](#) = 0U , [kTEMPMON_PanicAlarmMode](#) = 1U ,
[kTEMPMON_LowAlarmMode](#) = 2U }
TEMPMON alarm mode.

Functions

- void [TEMPMON_Init](#) (TEMPMON_Type *base, const [tempmon_config_t](#) *config)
Initializes the TEMPMON module.
- void [TEMPMON_Deinit](#) (TEMPMON_Type *base)
Deinitializes the TEMPMON module.
- void [TEMPMON_GetDefaultConfig](#) ([tempmon_config_t](#) *config)
Gets the default configuration structure.
- float [TEMPMON_GetCurrentTemperature](#) (TEMPMON_Type *base)
Get current temperature with the fused temperature calibration data.
- void [TEMPMON_SetTempAlarm](#) (TEMPMON_Type *base, int8_t tempVal, [tempmon_alarm_mode](#) alarm↔
Mode)
Set the temperature count (raw sensor output) that will generate an alarm interrupt.

Driver version

- #define [FSL_TEMPMON_DRIVER_VERSION](#) (MAKE_VERSION(2, 1, 1))
TEMPMON driver version.

14.40.4.4.1 Detailed Description

i.MX RT1061 Processor Temperature Monitor

14.40.4.4.2 Enumeration Type Documentation

14.40.4.4.2.1 _tempmon_alarm_mode

```
enum \_tempmon\_alarm\_mode
#include <fsl\_tempmon.h>
TEMPMON alarm mode.
```


Enumerator

kTEMPMON_HighAlarmMode	The high alarm temperature interrupt mode.
kTEMPMON_PanicAlarmMode	The panic alarm temperature interrupt mode.
kTEMPMON_LowAlarmMode	The low alarm temperature interrupt mode.

14.40.4.4.3 Function Documentation**14.40.4.4.3.1 TEMPMON_Deinit()**

```
void TEMPMON_Deinit (
    TEMPMON_Type * base)
#include <fsl_tempmon.h>
Deinitializes the TEMPMON module.
```

Parameters

<i>base</i>	TEMPMON base pointer

14.40.4.4.3.2 TEMPMON_GetCurrentTemperature()

```
float TEMPMON_GetCurrentTemperature (
    TEMPMON_Type * base)
#include <fsl_tempmon.h>
Get current temperature with the fused temperature calibration data.
```

Parameters

<i>base</i>	TEMPMON base pointer

Returns

current temperature with degrees Celsius.

14.40.4.4.3.3 TEMPMON_GetDefaultConfig()

```
void TEMPMON_GetDefaultConfig (
    tempmon_config_t * config)
#include <fsl_tempmon.h>
Gets the default configuration structure.
```

This function initializes the TEMPMON configuration structure to a default value. The default values are: tempmonConfig->frequency = 0x02U; tempmonConfig->highAlarmTemp = 44U; tempmonConfig->panicAlarmTemp = 90U; tempmonConfig->lowAlarmTemp = 39U;

Parameters

<i>config</i>	Pointer to a configuration structure.

14.40.4.4.3.4 TEMPMON_Init()

```
void TEMPMON_Init (
    TEMPMON_Type * base,
    const tempmon_config_t * config)
#include <fsl_tempmon.h>
Initializes the TEMPMON module.
```

Parameters

<i>base</i>	TEMPMON base pointer
<i>config</i>	Pointer to configuration structure.

14.40.4.4.3.5 TEMPMON_SetTempAlarm()

```
void TEMPMON_SetTempAlarm (
    TEMPMON_Type * base,
    int8_t tempVal,
    tempmon_alarm_mode alarmMode)
```

```
#include <fsl_tempmon.h>
```

Set the temperature count (raw sensor output) that will generate an alarm interrupt.

Parameters

<i>base</i>	TEMPMON base pointer
<i>tempVal</i>	The alarm temperature with degrees Celsius
<i>alarmMode</i>	The alarm mode.

14.41 SMTP - Send Email

Topics

- [MIME Content Types](#)
- [SMTP Error Codes](#)
- [SSL SMTP Functions](#)

Functions

- int [SendMail](#) (IPADDR smtp_server, PCSTR userid, PCSTR from_addr, PCSTR to_addr, PCSTR subject, PCSTR textbox)

Send an email message. The function will open a TCP connection to the specified SMTP server, create a message based on the parameters, and send the message.
- int [SendMailAuth](#) (IPADDR smtp_server, PCSTR userid, PCSTR pass, PCSTR from_addr, PCSTR to_addr, PCSTR subject, PCSTR textbox)

Send an email message with plain text authentication. The function will open a TCP connection to the specified SMTP server, create a message based on the parameters, and send the message.
- int [SendMailEx](#) (IPADDR smtp_server, PCSTR userid, PCSTR from_addr_rev_path, PCSTR from_addr, PCSTR to_addr, PCSTR subject, PCSTR textbox)

Send an email message function, extended version.
- int [IsMailError](#) ()

Returns the error status of the last send mail transaction.
- void [PrintNError](#) (int fd=0)

If an error occurred, prints the error information received from the SMTP server.
- void [PrintServerLog](#) (int fd=0)

Prints the server log of the last send mail transaction.
- int [SendMailAuthStartMIME](#) (IPADDR smtp_server, PCSTR userid, PCSTR pass, PCSTR from_addr, PCSTR to_addr, PCSTR subject, int &fd)

Start a Multi-purpose Internet Mail Extension (MIME) session.
- int [SendMailAuthAddMIME](#) (int fd, int ContentType, const char *pContent, const char *FileName)

Add a MIME part or attachment to an open MIME Session.
- int [SendMailAuthEndMIME](#) (int fd, PCSTR userid)

Send a MIME email message and close the SMTP session.

- enum [CONTENT_TYPE_ENUM](#) {
[CONTENT_TYPE_PLAIN_TEXT](#) , [CONTENT_TYPE_PLAIN_TEXT_ATTACH](#) , [CONTENT_TYPE_BINARY_ATTACH](#)
, [CONTENT_TYPE_HTML_DECOMP](#) ,
[CONTENT_TYPE_END](#) }

SMTP MIME Conetnet Types.

14.41.1 Detailed Description

```
#include< mailto.h>
```

Send emails using the SMTP specification. The Send Mail functions provide a variety of sending options for simple sending an email, or sending with authentication. The IsMailError function is used to check for errors after an email is sent.

14.41.2 Enumeration Type Documentation

14.41.2.1 CONTENT_TYPE_ENUM

```
enum CONTENT\_TYPE\_ENUM
#include <mailto.h>
SMTP MIME Conetnet Types.
```

Enumerator

CONTENT_TYPE_PLAIN_TEXT	Plain text.
CONTENT_TYPE_PLAIN_TEXT_ATTACH	Plain text attachment.
CONTENT_TYPE_BINARY_ATTACH	Binary attachment.
CONTENT_TYPE_HTML_DECOMP	HTML.
CONTENT_TYPE_END	Additional content types can be added above this line.

14.41.3 Function Documentation

14.41.3.1 IsMailError()

```
int IsMailError ()
#include <mailto.h>
```

Returns the error status of the last send mail transaction.

Returns

0 if no error occurred, [SMTP Error Codes](#).

See also

[PrintNError\(\)](#), [PrintServerLog\(\)](#)

14.41.3.2 PrintNError()

```
void PrintNError (
    int fd = 0)
#include <mailto.h>
```

If an error occurred, prints the error information received from the SMTP server.

If an error occurs when sending an email, the system will record the error information received from the SMTP server. The information is only valid for the last transaction.

Parameters

<i>fd</i>	If no parameter is specified information will be sent to stdout. Otherwise it will be sent to the specified file descriptor.

See also

[IsMailError\(\)](#), [PrintServerLog\(\)](#)

14.41.3.3 PrintServerLog()

```
void PrintServerLog (
    int fd = 0)
#include <mailto.h>
```

Prints the server log of the last send mail transaction.

The results of each send mail transaction are logged to a buffer. This function will send the results of the last transaction to stdout, or the specified file descriptor. The information is only valid for the last transaction.

Parameters

<i>fd</i>	If no parameter is specified information will be sent to stdout. Otherwise it will be sent to the specified file descriptor.

See also

[IsMailError\(\)](#), [PrintNError\(\)](#)

14.41.3.4 SendMail()

```
int SendMail (
    IPADDR smtp_server,
    PCSTR userid,
    PCSTR from_addr,
    PCSTR to_addr,
    PCSTR subject,
    PCSTR textbody)
```

```
#include <mailto.h>
```

Send an email message. The function will open a TCP connection to the specified SMTP server, create a message based on the parameters, and send the message.

Parameters

<i>smtp_server</i>	IP address of the destination SMTP server.
<i>userid</i>	ASCII string representing the RFC 931 identification.
<i>from_addr</i>	"From" email address
<i>to_addr</i>	Destination email address
<i>subject</i>	Email subject
<i>textbody</i>	Body of email message

Returns

0 on failure, 1 on success.

See also

[SendMailAuth\(\)](#)

14.41.3.5 SendMailAuth()

```
int SendMailAuth (
    IPADDR smtp_server,
    PCSTR userid,
    PCSTR pass,
    PCSTR from_addr,
    PCSTR to_addr,
    PCSTR subject,
    PCSTR textbody)
```

```
#include <mailto.h>
```

Send an email message with plain text authentication. The function will open a TCP connection to the specified SMTP server, create a message based on the parameters, and send the message.

Send an email message with SMTP plain text authentication per RFC 931. The function will open a TCP connection to the specified SMTP server, authenticate the connection with the specified username and password, create a message based on the parameters, and send the message.

Parameters

<i>smtp_server</i>	IP address of the destination SMTP server.
<i>userid</i>	ASCII string representing the RFC 931 identification.
<i>pass</i>	ASCII string to provide for AUTH identification.
<i>from_addr</i>	"From" email address
<i>to_addr</i>	Destination email address
<i>subject</i>	Email subject
<i>textbody</i>	Body of email message

Returns

0 on failure, 1 on success.

See also

[SendMail\(\)](#)

14.41.3.6 SendMailAuthAddMIME()

```
int SendMailAuthAddMIME (
    int fd,
    int ContentType,
    const char * pContent,
    const char * FileName)
```

```
#include <mailto.h>
```

Add a MIME part or attachment to an open MIME Session.

Must only be called for an open session started with [SendMailAuthStartMIME\(\)](#). It can be called as many times as needed to add each MIME part or attachment to the message.

Parameters

<i>fd</i>	The file descriptor of the open MIME session.
<i>ContentType</i>	The MIME content type, such as: CONTENT_TYPE_PLAIN_TEXT, CONTENT_TYPE_PLAIN_TEXT_ATTACH, CONTENT_TYPE_BINARY_ATTACH.
<i>*pContent</i>	Pointer to the content to add.
<i>*FileName</i>	File name for the attachment.

Returns

0 on failure, 1 on success.

See also

[SendMailAuthStartMIME\(\)](#), [SendMailAuthEndMIME\(\)](#)

14.41.3.7 SendMailAuthEndMIME()

```
int SendMailAuthEndMIME (
    int fd,
    PCSTR userid)
```

```
#include <mailto.h>
```

Send a MIME email message and close the SMTP session.

Must only be called for an open session started with [SendMailAuthStartMIME\(\)](#).

Parameters

<i>fd</i>	The file descriptor of the open MIME session.
<i>*userid</i>	The user ID that was used by SendMailAuthStartMIME() to open the session.

Returns

0 on failure, 1 on success.

See also

[SendMailAuthStartMIME\(\)](#), [SendMailAuthAddMIME\(\)](#)

14.41.3.8 SendMailAuthStartMIME()

```
int SendMailAuthStartMIME (
    IPADDR smtp_server,
    PCSTR userid,
    PCSTR pass,
    PCSTR from_addr,
    PCSTR to_addr,
    PCSTR subject,
    int & fd)
```

```
#include <mailto.h>
```

Start a Multi-purpose Internet Mail Extension (MIME) session.

Start a MIME session with plain text password authentication. This function will open a TCP connection to the SMTP server, authenticate, then return while leaving the TCP connection open. The Session file descriptor is returned in the "int &fd" reference variable. This function must be called before any other SendMail MIME function.

Sending an E-Mail message with Multipurpose Internet Mail Extensions (MIME) is a multi step process:

- Call [SendMailAuthStartMIME\(\)](#) to begin a Session with the SMTP server, connection will remain open.
- Call [SendMailAuthAddMIME\(\)](#) for each MIME attachment you want to add.
- Call [SendMailAuthEndMIME\(\)](#) to send the email message and close the Session.

The following MIME types are supported by default. The binary attachment should cover most types such as jpg, gif, etc. The types are defined in \nburn\nbrtos\include\mailto.h, and the functions are implemented in \nburn\system\mailto.cpp.

MIME Content Types defined by [MIME Content Types](#).

Parameters

<i>smtp_server</i>	IP address of the destination SMTP server.
<i>userid</i>	ASCII string representing the RFC 931 identification.
<i>pass</i>	Password.
<i>from_addr</i>	Reverse path per RFC 821, reverse source route.
<i>to_addr</i>	Destination email address.
<i>subject</i>	Email subject.
<i>fd</i>	A reference (or pointer) to an integer value of the calling function. The value of fd is modified by this function and will contain the file descriptor for the open SMTP Session. This value must always be checked to be greater than 0 for a valid Session. The fd is passed to other MIME functions to identify the open Session.

Returns

0 on failure, 1 on success.

See also

[SendMailAuthAddMIME\(\)](#), [SendMailAuthEndMIME\(\)](#)

14.41.3.9 SendMailEx()

```
int SendMailEx (
    IPADDR smtp_server,
    PCSTR userid,
    PCSTR from_addr_rev_path,
    PCSTR from_addr_memo_hdr,
    PCSTR to_addr,
    PCSTR subject,
    PCSTR textbody)
```

#include <mailto.h>

Send an email message function, extended version.

Send an email message with extended parameters. The function will open a TCP connection to the specified SMTP server, create a message based on the parameters, and send the message. This function is identical to [SendMail\(\)](#), with the addition of the from_addr_rev_path and from_addr_memo_hdr parameters.

Parameters

<i>smtp_server</i>	IP address of the destination SMTP server.
<i>userid</i>	ASCII string representing the RFC 931 identification.
<i>from_addr_rev_path</i>	Reverse path per RFC 821, reverse source route.
<i>from_addr_memo_hdr</i>	From address memo header per RFC 821. Contains items such as date, subject, to:, CC: and from:.
<i>to_addr</i>	Destination email address
<i>subject</i>	Email subject
<i>textbody</i>	Body of email message

Returns

0 on failure, 1 on success.

See also

[SendMail\(\)](#)

14.41.4 MIME Content Types

```
#include< mailto.h>
```

14.41.5 SMTP Error Codes

Macros

- **#define STATUS_OK (0)**
OK, no errors.
- **#define CONNECT_TO_SMTP_SERVER_FAILED (-1)**
Could not connect to SMTP server.
- **#define INITIAL_SERVER_REPLY_FAILED (-2)**
Initial server reply failed.
- **#define HELO_SERVER_REPLY_FAILED (-3)**
Server HELO reply failed.
- **#define MAIL_FROM_SERVER_REPLY_FAILED (-4)**
Mail From server reply failed.
- **#define RCPT_TO_SERVER_REPLY_FAILED (-5)**
Receipt To server reply failed.
- **#define DATA_SERVER_REPLY_FAILED (-6)**
Data server reply failed.
- **#define DATA_END_SERVER_REPLY_FAILED (-7)**
Date end server reply failed.
- **#define AUTH_LOGIN_SERVER_REPLY_FAILED (-8)**
AUTH login server reply failed.
- **#define USER_ID_SERVER_REPLY_FAILED (-9)**
User ID server reply failed.
- **#define PASSWORD_SERVER_REPLY_FAILED (-10)**
Password server reply failed.
- **#define CONNECT931_SMTP_SERVER_FAILED (-11)**
SMTP connection failed.

14.41.5.1 Detailed Description

```
#include< mailto.h>
```

14.41.6 SSL SMTP Functions

Functions

- int [SSL_SendMail](#) ([IPADDR](#) smtp_server, PCSTR userid, PCSTR pass, PCSTR from_addr, PCSTR to_↵
addr, PCSTR subject, PCSTR textbody, BOOL STARTTLS=false, int remotePort=-1, const char *server↵
Name=nullptr, bool verifyPeer=false, const unsigned char *certBuff=nullptr, uint32_t certBuffSize=0, uint16_↵
_t certBuffFormat=SSL_FILETYPE_PEM)
Send an email message using SSL encryption.
- int [SSL_SendMailHTML](#) ([IPADDR](#) smtp_server, PCSTR userid, PCSTR pass, PCSTR from_addr, PCSTR
to_addr, PCSTR subject, const char *filename, BOOL STARTTLS=false, int remotePort=-1, const char
*serverName=nullptr, const unsigned char *certBuff=nullptr, uint32_t certBuffSize=0, uint16_t certBuff↵
Format=SSL_FILETYPE_PEM)

Send an HTML email message from a file using SSL encryption.

- int [SSL_SendMailStartMIME](#) (IPADDR smtp_server, PCSTR userid, PCSTR pass, PCSTR from_addr, PCSTR to_addr, PCSTR subject, BOOL STARTTLS=false, int remotePort=-1, const char *serverName=nullptr, bool verifyPeer=false, const unsigned char *certBuff=nullptr, uint32_t certBuffSize=0, uint16_t certBuffFormat=SSL_FILETYPE_PEM)

Start a Multi-purpose Internet Mail Extension (MIME) session using SSL encryption.

14.41.6.1 Detailed Description

The following functions is very similar to the unencrypted [SendMail\(\)](#) function. The initial connection of the SMTP server can happen in one of two ways:

1) An SSL connection is negotiated immediately. 2) A TCP connection is opened first, and a SSL connection is negotiated when the SMTP server sends a STARTTLS command to the NetBurner device.

Setting STARTTLS = TRUE enables the second mode of operation.

14.41.6.2 Function Documentation

14.41.6.2.1 SSL_SendMail()

```
int SSL_SendMail (
    IPADDR smtp_server,
    PCSTR userid,
    PCSTR pass,
    PCSTR from_addr,
    PCSTR to_addr,
    PCSTR subject,
    PCSTR textbody,
    BOOL STARTTLS = false,
    int remotePort = -1,
    const char * serverName = nullptr,
    bool verifyPeer = false,
    const unsigned char * certBuff = nullptr,
    uint32_t certBuffSize = 0,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)
```

```
#include <ssl_mailto.h>
```

Send an email message using SSL encryption.

Parameters

<i>smtp_server</i>	IP address of the SMTP server.
<i>userid</i>	Ascii String to provide for RFC931 IDentification.
<i>pass</i>	Ascii String to provide for AUTH IDentification.
<i>from_addr</i>	From email address.
<i>to_addr</i>	To email address.
<i>subject</i>	The email subject.
<i>textbody</i>	The email body.
<i>STARTTLS</i>	Enable/disable STARTTLS functionality. Default is disabled.
<i>remotePort</i>	Port number to connect to on the SMTP server.
<i>serverName</i>	Server common name to use for certificate name verification and when sending SNI extensions.
<i>verifyPeer</i>	Set true to verify the server's certificate against a Certificate Authority list. Default if false.
<i>certBuff</i>	Buffer containing the certificate data.Default is null.
<i>certBuffSize</i>	Size of the buffer containing the certificate data.
<i>certBuffFormat</i>	The format of the certificate buffer. Default is SSL_FILETYPE_PEM.

Return values

0	Send failed
1	Send was successful

See also

[SSL_SendMailHTML\(\)](#)

[SSL_SendMailStartMIME\(\)](#)

[SendMail\(\)](#)

14.41.6.2.2 SSL_SendMailHTML()

```
int SSL_SendMailHTML (
    IPADDR smtp_server,
    PCSTR userid,
    PCSTR pass,
    PCSTR from_addr,
    PCSTR to_addr,
    PCSTR subject,
    const char * filename,
    BOOL STARTTLS = false,
    int remotePort = -1,
    const char * serverName = nullptr,
    const unsigned char * certBuff = nullptr,
    uint32_t certBuffSize = 0,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)
#include <ssl_mailto.h>
```

Send an HTML email message from a file using SSL encryption.

Parameters

<i>smtp_server</i>	IP address of the SMTP server.
<i>userid</i>	Ascii string to provide for RFC931 IDentification.
<i>pass</i>	Ascii string to provide for AUTH IDentification.
<i>from_addr</i>	From email address.
<i>to_addr</i>	To email address.
<i>subject</i>	The email subject.
<i>filename</i>	HTTP server filename to send.
<i>STARTTLS</i>	Enable/disable STARTTLS functionality. Default is disabled.
<i>remotePort</i>	Port number to connect to on the SMTP server.
<i>serverName</i>	Server name to use for certificate name verification and when sending SNI extensions.
<i>certBuff</i>	Buffer containing the certificate data. Default is null.
<i>certBuffSize</i>	Size of the buffer containing the certificate data.
<i>certBuffFormat</i>	The format of the certificate buffer. Default is SSL_FILETYPE_PEM.

Return values

0	Send failed
1	Send was successful

See also

[SSL_SendMail\(\)](#)

[SSL_SendMailStartMIME\(\)](#)

[SendMail\(\)](#)

14.41.6.2.3 SSL_SendMailStartMIME()

```
int SSL_SendMailStartMIME (
    IPADDR smtp_server,
    PCSTR userid,
    PCSTR pass,
    PCSTR from_addr,
    PCSTR to_addr,
    PCSTR subject,
    BOOL STARTTLS = false,
    int remotePort = -1,
    const char * serverName = nullptr,
    bool verifyPeer = false,
    const unsigned char * certBuff = nullptr,
    uint32_t certBuffSize = 0,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)

#include <ssl_mailto.h>
```

Start a Multi-purpose Internet Mail Extension (MIME) session using SSL encryption.

Sending an E-Mail message with Multipurpose Internet Mail Extensions (MIME) is a multi step process:

- Call [SSL_SendMailStartMIME\(\)](#) to begin a Session with the SMTP server, connection will remain open.
- Call [SendMailAuthAddMIME\(\)](#) for each MIME attachment you want to add.
- Call [SendMailAuthEndMIME\(\)](#) to send the email message and close the Session.

The following MIME types are supported by default. The binary attachment should cover most types such as jpg, gif, etc. The types are defined by [MIME Content Types](#).

Parameters

<i>smtp_server</i>	IP address of the SMTP server.
<i>userid</i>	Ascii string to provide for RFC931 IDentification.
<i>pass</i>	Ascii string to provide for AUTH IDentification.
<i>from_addr</i>	From email address.
<i>to_addr</i>	To email address.
<i>subject</i>	The email subject.
<i>STARTTLS</i>	Enable/disable STARTTLS functionality. Default is disabled.
<i>remotePort</i>	Port number to connect to on the SMTP server.
<i>serverName</i>	Server name to use for certificate name verification and when sending SNI extensions.
<i>verifyPeer</i>	Set true to verify the server's certificate against a Certificate Authority list. Default if false.
<i>certBuff</i>	Buffer containing the certificate data. Default is null.
<i>certBuffSize</i>	Size of the buffer containing the certificate data.
<i>certBuffFormat</i>	The format of the certificate buffer.

Return values

0	Send failed
1	Send was successful

See also

[SendMailAuthStartMIME\(\)](#)
[SendMailAuthAddMIME\(\)](#)
[SendMailAuthEndMIME\(\)](#)

14.42 SOCKS

Topics

- [SOCKS Error Codes](#)

Macros

- `#define SOCKS_MAX_UNAME_SIZE 255`
Max character length for usernames. Defined in RFC 1929.
- `#define SOCKS_MAX_PASSWD_SIZE 255`
Max character length for passwords. Defined in RFC 1929.

Enumerations

- enum [SocksAuthType](#) : unsigned char { [eSocksAuthTypeGssApi](#) = 0x01 , [eSocksAuthTypeUnPw](#) = 0x02 , [eSocksAuthTypeNoAuth](#) = 0x04 }
- SOCKS Authorization Types.*
- enum [SocksClientCmd](#) : unsigned char { [eSocksClientCmdConnect](#) = 1 , [eSocksClientCmdBind](#) = 2 , [eSocksClientCmdUdpAssoc](#) = 3 }
- SOCKS Client Commands.*
- enum [SocksAdrType](#) : unsigned char { [eSocksAdrTypeNone](#) = 0x00 , [eSocksAdrTypeIpv4](#) = 0x01 , [eSocksAdrTypeDomain](#) = 0x03 , [eSocksAdrTypeIpv6](#) = 0x04 }
- SOCKS Address Types.*

Functions

- bool [AuthWithGssApi](#) ()
A weak function that should be overridden by the developer in order to support GSSAPI authorization.
- void [SetSocksProxySettings](#) (SocksProxy *socksProxy)
Set the system level SOCKS proxy settings object to the one that is passed in. If this object is set and is marked as enabled, calls to [CoreConnect\(\)](#) (made from [connect\(\)](#), [DoGet\(\)](#), etc), will all initially connect to the proxy server using the SOCKS5 protocol.
- SocksProxy * [GetSocksProxySettings](#) ()
Get a pointer to the currnetly set Socks proxy settings object.

14.42.1 Detailed Description

```
#include< Socks.h>
```

The NetBurner SOCKS Library

14.42.2 Enumeration Type Documentation

14.42.2.1 SocksAdrType

```
enum SocksAdrType : unsigned char
#include <Socks.h>
SOCKS Address Types.
```

Enumerator

eSocksAdrTypeNone	None.
eSocksAdrTypeIpv4	IPv4.
eSocksAdrTypeDomain	Domain.
eSocksAdrTypeIpv6	IPv6.

14.42.2.2 SocksAuthType

```
enum SocksAuthType : unsigned char
#include <Socks.h>
SOCKS Authorization Types.
```

Enumerator

eSocksAuthTypeGssApi	GSS API.
eSocksAuthTypeUnPw	User name and password.
eSocksAuthTypeNoAuth	Not as defined in RFC so that it could be properly tested against when being set by user.

14.42.2.3 SocksClientCmd

```
enum SocksClientCmd : unsigned char
#include <Socks.h>
SOCKS Client Commands.
```

Enumerator

eSocksClientCmdConnect	Connect.
eSocksClientCmdBind	Not currently supported, see RFC 1928.
eSocksClientCmdUdpAssoc	Not currently supported, see RFC 1928.

14.42.3 Function Documentation**14.42.3.1 AuthWithGssApi()**

```
bool AuthWithGssApi ()
#include <Socks.h>
```

A weak function that should be overridden by the developer in order to support GSSAPI authorization.

Return values

<i>true</i>	The GSSAPI authorization succeeded.
<i>false</i>	The GSSAPI authorization failed.

14.42.3.2 GetSocksProxySettings()

```
SocksProxy * GetSocksProxySettings ()
#include <Socks.h>
```

Get a pointer to the currently set Socks proxy settings object.

Returns

SocksProxy*

14.42.3.3 SetSocksProxySettings()

```
void SetSocksProxySettings (
    SocksProxy * socksProxy)
#include <Socks.h>
```

Set the system level SOCKS proxy settings object to the one that is passed in. If this object is set and is marked as enabled, calls to CoreConnect() (made from [connect\(\)](#), [DoGet\(\)](#), etc), will all initially connect to the proxy server using the SOCKS5 protocol.

Parameters

<code>socksProxy</code>	The proxy object that the system should reference.

14.42.4 SOCKS Error Codes

Macros

- **#define SOCKS_SUCCESS** (0)
The connection was successful.
- **#define SOCKS_ERR_TIMEOUT** (-500)
The connection timed out.
- **#define SOCKS_BAD_ADR_TYPE** (-501)
Address type specified does not match the address provided.
- **#define SOCKS_CONN_ABORTED** (-502)
The other side aborted the connection.
- **#define SOCKS_BAD_AUTH_TYPE** (-503)
The server doesn't support the requested authorization types.
- **#define SOCKS_BAD_UN_PW** (-504)
The server did not approve the username and password provided.
- **#define SOCKS_UN_TOO_LONG** (-505)
The specified username is too long.
- **#define SOCKS_PW_TOO_LONG** (-506)
The specified password is too long.
- **#define SOCKS_BAD_REPLY** (-507)
The server reply was not successful.
- **#define SOCKS_BAD_DOMAIN** (-508)
Unable to resolve domain name passed.
- **#define SOCKS_CMD_NOT_SUPPORTED** (-509)
The requested command aren't supported.
- **#define SOCKS_BAD_PARAM** (-510)
The passed in parameters are not valid.

14.42.4.1 Detailed Description

```
#include< Socks.h>
```

Error codes that can be returned by SOCKS related functions.

14.43 SPI

Topics

- [MCF5441x \(DSPI\)](#)
DSPI for MCF5441x (MOD5441x, NANO, SB800EX)
- [MCF5441x \(Queued SPI 2.x compatibility layer\)](#)
QSPI for MCF5441x (MOD5441x, NANO, SB800EX)
- [MIMX10xx \(DSPI\)](#)
DSPI for ARM MIMX10xx (SOMRT1061)
- [SAME70 \(DSPI\)](#)
DSPI for ARM SAME70 (MODM7AE70, SBE70LC)
- [SAME70 \(QuadSPI\)](#)
QSPI for SAME70 (MODM7AE70, SBE70LC)
- [SAME70 \(USART\)](#)
USART for SAME70 (MODM7AE70, SBE70LC)

14.43.1 Detailed Description

Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals over a short distance, such as port expanders, FPGAs, CPLDs, shift registers, sensors, and flash SD cards. Unlike an asynchronous UART that does not use a separate clock, the SPI operates in a synchronous mode using a single clock for both sender and receiver that determines the data rate as well as when each bit of data is valid. This API uses the "native" SPI mode with the following signals:

- Clock (CLK). Synchronous clock signal.
- Master out, Slave in (MOSI). Data signal from master to slave device.
- Master in, Slave out (MISO). Data signal from slave to master.
- Chip select (CS). Chip selects can be part of the SPI peripheral, or can be general purpose I/O.

An advantage of SPI is that the receiving hardware can be a simple shift register. This is a much simpler and less expensive than a UART implementation.

14.43.2 MCF5441x (DSPI)

DSPI for MCF5441x (MOD5441x, NANO, SB800EX)

Classes

- struct [dspiDMAStruct](#)
DSPI DMA structure.
- struct [dspiDriverStruct](#)
Major variables/configurations used for a DSPI transfer.
- class [DSPIModule](#)
DSPIModule is a SPI communications driver. It is an object based driver, which allows for low overhead multiplexing between peripherals with different bus configurations.

Macros

- #define [DEFAULT_DSPI_MODULE](#) 1
- #define [DSPI_MODULE_COUNT](#) 4

Enumerations

- enum `csReturnType` { `DEASSERT_NEVER` = 0 , `DEASSERT_AFTER_LAST` = 1 , `DEASSERT_EVERY_TRANSFER` = 2 }
- Chip select return types.*
- enum `spiChipSelect` { `CHIP_SELECT_0` = 0xFE , `CHIP_SELECT_1` = 0xFD , `CHIP_SELECT_2` = 0xFB , `CHIP_SELECT_3` = 0xF7 , `CHIP_SELECT_DISABLED` = 0xFF }
- Chip select number.*
- enum `spiChipSelectPolarity` { `CS_ASSERT_LOW` = 0x00 , `CS_ASSERT_HIGH` = 0xFF }
- Chip select polarity.*

Functions

- uint8_t `DSPIInit` (uint8_t `SPIModule`=`DEFAULT_DSPI_MODULE`, uint32_t `Baudrate`=2000000, uint8_t `QueueBitSize`=8, uint8_t `CS`=0x00, uint8_t `CSPol`=0x0F, uint8_t `ClkPolarity`=0, uint8_t `ClkPhase`=1, BOOL `DoutHiz`=TRUE, uint8_t `QCD`=0, uint8_t `DTL`=0)
- Initialize a DSPI module.*
- uint8_t `DSPIStart` (uint8_t `SPIModule`, uint8_t `transmitBufferPtr`, volatile uint8_t *`receiveBufferPtr`, uint32_t `byteCount`, OS_SEM *`finishedSem`=NULL, uint8_t `enableDMA`=TRUE, int `csReturnToInactive`=`DEASSERT_AFTER_LAST`)
- Start DSPI Data Transfer.*
- BOOL `DSPIDone` (uint8_t `SPIModule`=`DEFAULT_DSPI_MODULE`)
- Check current DSPI Data Transfer.*
- uint8_t `QSPIInit` (uint32_t `baudRateInBps`=2000000, uint8_t `transferSizeInBits`=8, uint8_t `peripheralChipSelects`=0x0F, uint8_t `chipSelectPolarity`=1, uint8_t `clockPolarity`=0, uint8_t `clockPhase`=1, BOOL `doutHiz`=TRUE, uint8_t `csToClockDelay`=0, uint8_t `delayAfterTransfer`=0)
- Initialize Queued Serial Peripheral Interface (QSPI)*
- uint8_t `QSPIStart` (uint8_t `transmitBufferPtr`, volatile uint8_t *`receiveBufferPtr`, uint32_t `byteCount`, OS_SEM *`finishedSem`=NULL)
- Start QSPI Data Transfer.*
- BOOL `QSPIDone` ()
- Can be called after `QSPIStart()`. Returns TRUE when transfer is complete. This is an alternative to using a semaphore.*

DSPI state

- #define `DSPI_OK` (0)
- #define `DSPI_BUSY` (1)
- #define `DSPI_ERROR` (2)

MCR Register masks

- #define `MCR_MASTER_INIT` 0x80000C01
- #define `MCR_HALT_BIT` 0x00000001
- #define `MCR_DIS_TXF` 0x00002000
- #define `MCR_DIS_RXF` 0x00001000
- #define `MCR_CLR_FIFOS` 0x00000C00

CTAR Register masks

- #define `CTAR_CLOCK_POLARITY` 0x04000000
- #define `CTAR_CLOCK_PHASE` 0x02000000
- #define `CTAR_FRAME_16BIT` 0x78000000
- #define `CTAR_FRAME_4BIT` 0x18000000

SR Register masks

- #define `SR_EOQF_MASK` 0x10000000
- #define `SR_CLR_FLAGS` 0x8A0A0000

RSER (Interrupt/DMA enable) Register masks

- #define `RSER_EOQF_IRQ_ONLY` ~0x8B0B0000
- #define `RSER_DMA_IRQ_ONLY` 0x03030000

PUSHR (command) Register masks

- #define `PUSHR_CONT_BIT` 0x80000000
- #define `PUSHR_EOQ_BIT` 0x08000000

DMA CHANNEL NUMBERS

- #define `DMA_CH_DSPI_0_RX` 0x0C
(12)
- #define `DMA_CH_DSPI_0_TX` 0x0D
(13)
- #define `DMA_CH_DSPI_1_RX` 0x0E
(14)
- #define `DMA_CH_DSPI_1_TX` 0x0F
(15)
- #define `DMA_CH_DSPI_2_RX` 0x1C
(28)
- #define `DMA_CH_DSPI_2_TX` 0x1D
(29)
- #define `DMA_CH_DSPI_3_RX` 0x2C
(44)
- #define `DMA_CH_DSPI_3_TX` 0x2D
(45)

EDMA CR Register masks

- #define `CR_CLEAR_NON_GRP_PRIO` ~0x000300FF
- #define `CR_SET_RR_CH_ARB` 0x00000004

EDMA TCD masks

- #define `TCD_ATTR_8BIT_TRANS` 0x0000
- #define `TCD_ATTR_16BIT_TRANS` 0x0101
- #define `TCD_XOFF_0uint8_t` 0x0000
- #define `TCD_XOFF_1uint8_t` 0x0001
- #define `TCD_XOFF_2uint8_t` 0x0002
- #define `TCD_XITER_CNT_MASK` 0x7FFF
- #define `TCD_CSR_DONE_BIT` 0x0080
- #define `TCD_CSR_DISABLE_REQ` 0xC008
- #define `TCD_CSR_DREQ_INT_MAJOR` 0xC00A

14.43.2.1 Detailed Description

DSPI for MCF5441x (MOD5441x, NANO, SB800EX)

Supported platforms:

- MOD54415
- MOD54417
- NANO54415
- SB800EX

The DMA serial peripheral interface (DSPI) interfaces (up to 3 depending on your platform) provide a synchronous serial bus for communication between the processor and an external peripheral device. DSPI interface modules are numbered from 0 to 3. DSPI 0 is reserved for system use. The DSPI supports up to 32 queued SPI transfers (16 receive and 16 transmit) in the DSPI resident FIFOs eliminating CPU intervention between transfers.

The DSPI specification allows for sharing the SPI bus with multiple devices. However, when designing your system, you must consider what devices are connected and the desired throughput. For example, if you are using the Embedded Flash File System (EPPS) with external Flash cards, or the WiFi module, they will require exclusive use of the DSPI interface.

The DSPI uses the following hardware signals:

- SCLK Serial clock output from master.
- MOSI Master data output (transmit), slave data input (receive).
- MISO Master data input (receive), slave data output (transmit).
- QSPI_CS_n Optional QSPI chip selects. Note that you can also use GPIO signals and control them manually for the chip select functionality. In some cases, there are advantages to doing it this way.

DSPI Configuration and Initialization

Use of the DSPI requires:

1. Identify which pins on your NetBurner device provide primary or secondary QSPI functionality. Use the NetBurner Pins class functions to configure these pins for QSPI operation.
2. Single task applications: you can choose to call the [DSPIDone\(\)](#) function to determine when the DSPI transfer is complete, or use a semaphore and pend for completion.
3. Multi-task applications: If your application has more than one task that will access the same DSPI, you must create a semaphore to control access to the DSPI resource, just as you would with any shared resource.
4. Initialize the DSPI with the [DSPIInit\(\)](#) function.
5. When you have data to send, or want to read data from the DSPI slave device, call the [QSPIStart\(\)](#) function to initiate the data transfer.
6. The semaphore will post when the data transfer is complete.

14.43.2.2 Macro Definition Documentation

14.43.2.2.1 CR_CLEAR_NON_GRP_PRIO

```
#define CR_CLEAR_NON_GRP_PRIO ~0x000300FF
#include <dspi.h>
```

14.43.2.2.2 CR_SET_RR_CH_ARB

```
#define CR_SET_RR_CH_ARB 0x00000004
#include <dspi.h>
```

14.43.2.2.3 CTAR_CLOCK_PHASE

```
#define CTAR_CLOCK_PHASE 0x02000000
#include <dspi.h>
```

14.43.2.2.4 CTAR_CLOCK_POLARITY

```
#define CTAR_CLOCK_POLARITY 0x04000000
#include <dspi.h>
```

14.43.2.2.5 CTAR_FRAME_16BIT

```
#define CTAR_FRAME_16BIT 0x78000000
#include <dspi.h>
```

14.43.2.2.6 CTAR_FRAME_4BIT

```
#define CTAR_FRAME_4BIT 0x18000000
#include <dspi.h>
```

14.43.2.2.7 DEFAULT_DSPI_MODULE

```
#define DEFAULT_DSPI_MODULE 1
#include <dspi.h>
```

14.43.2.2.8 DSPI_BUSY

```
#define DSPI_BUSY (1)
#include <dspi.h>
```

14.43.2.2.9 DSPI_ERROR

```
#define DSPI_ERROR (2)
#include <dspi.h>
```

14.43.2.2.10 DSPI_MODULE_COUNT

```
#define DSPI_MODULE_COUNT 4
#include <dspi.h>
```

14.43.2.2.11 DSPI_OK

```
#define DSPI_OK (0)
#include <dspi.h>
```

14.43.2.2.12 MCR_CLR_FIFOS

```
#define MCR_CLR_FIFOS 0x00000C00
#include <dspi.h>
```

14.43.2.2.13 MCR_DIS_RXF

```
#define MCR_DIS_RXF 0x00001000
#include <dspi.h>
```

14.43.2.2.14 MCR_DIS_TXF

```
#define MCR_DIS_TXF 0x00002000
#include <dspi.h>
```

14.43.2.2.15 MCR_HALT_BIT

```
#define MCR_HALT_BIT 0x00000001
#include <dspi.h>
```

14.43.2.2.16 MCR_MASTER_INIT

```
#define MCR_MASTER_INIT 0x80000C01
#include <dspi.h>
```

14.43.2.2.17 PUSHR_CONT_BIT

```
#define PUSHR_CONT_BIT 0x80000000
#include <dspi.h>
```

14.43.2.2.18 PUSHR_EOQ_BIT

```
#define PUSHR_EOQ_BIT 0x08000000
#include <dspi.h>
```

14.43.2.2.19 RSER_DMA_IRQ_ONLY

```
#define RSER_DMA_IRQ_ONLY 0x03030000
#include <dspi.h>
```

14.43.2.2.20 RSER_EOQF_IRQ_ONLY

```
#define RSER_EOQF_IRQ_ONLY ~0x8B0B0000
#include <dspi.h>
```

14.43.2.2.21 SR_CLR_FLAGS

```
#define SR_CLR_FLAGS 0x8A0A0000
#include <dspi.h>
```

14.43.2.2.22 SR_EOQF_MASK

```
#define SR_EOQF_MASK 0x10000000
#include <dspi.h>
```

14.43.2.2.23 TCD_ATTR_16BIT_TRANS

```
#define TCD_ATTR_16BIT_TRANS 0x0101
#include <dspi.h>
```

14.43.2.2.24 TCD_ATTR_8BIT_TRANS

```
#define TCD_ATTR_8BIT_TRANS 0x0000
#include <dspi.h>
```

14.43.2.2.25 TCD_CSR_DISABLE_REQ

```
#define TCD_CSR_DISABLE_REQ 0xC008
#include <dspi.h>
```

14.43.2.2.26 TCD_CSR_DONE_BIT

```
#define TCD_CSR_DONE_BIT 0x0080
#include <dspi.h>
```

14.43.2.2.27 TCD_CSR_DREQ_INT_MAJOR

```
#define TCD_CSR_DREQ_INT_MAJOR 0xC00A
#include <dspi.h>
```

14.43.2.2.28 TCD_XITER_CNT_MASK

```
#define TCD_XITER_CNT_MASK 0x7FFF
#include <dspi.h>
```

14.43.2.2.29 TCD_XOFF_0uint8_t

```
#define TCD_XOFF_0uint8_t 0x0000
#include <dspi.h>
```

14.43.2.2.30 TCD_XOFF_1uint8_t

```
#define TCD_XOFF_1uint8_t 0x0001
#include <dspi.h>
```

14.43.2.2.31 TCD_XOFF_2uint8_t

```
#define TCD_XOFF_2uint8_t 0x0002
#include <dspi.h>
```

14.43.2.3 Enumeration Type Documentation**14.43.2.3.1 csReturnType**

```
enum csReturnType
#include <dspi.h>
```

Chip select return types.

Chip select deassertion modes. Used to determine when the driver should deassert chip selects during SPI transfer.

Enumerator

DEASSERT_NEVER	The chip select used for the transaction should remain asserted, even after the transaction is complete.
DEASSERT_AFTER_LAST	The chip select should remain asserted for the full duration of the transaction, and only be deasserted after the final transfer.
DEASSERT_EVERY_TRANSFER	The chip select should be deasserted between every transfer within the transaction.

14.43.2.3.2 spiChipSelect

```
enum spiChipSelect
#include <dspi.h>
```

Chip select number.

If supported by the SPI peripheral, these values can be used to configure the chip select to be used during a SPI transfer.

Enumerator

CHIP_SELECT_0	Configure bit mask for SPI chip select 0.
CHIP_SELECT_1	Configure bit mask for SPI chip select 1.
CHIP_SELECT_2	Configure bit mask for SPI chip select 2.
CHIP_SELECT_3	Configure bit mask for SPI chip select 3.
CHIP_SELECT_DISABLED	Configure bit mask to disable chip select.

14.43.2.3.3 spiChipSelectPolarity

```
enum spiChipSelectPolarity
```

```
#include <dspi.h>
```

Chip select polarity.

These values can be used to configure the chip select's asserted/deasserted logic level. These values will configure all chip selects to the same configuration. If you intend on using an SPI peripheral with multiple chip selects with different polarities, these values cannot not be used.

Enumerator

CS_ASSERT_LOW	Bit mask configures all active chip selects as active low, inactive high.
CS_ASSERT_HIGH	Bit mask configures all active chip selects as inactive low, active high.

14.43.2.4 Function Documentation

14.43.2.4.1 DSPIDone()

```
BOOL DSPIDone (
    uint8_t SPIModule = DEFAULT_DSPI_MODULE)
```

```
#include <dspi.h>
```

Check current DSPI Data Transfer.

Parameters: None

Return: Progress of transfer - TRUE is finished, FALSE is active

Notes: None

14.43.2.4.2 DSPIInit()

```
uint8_t DSPIInit (
    uint8_t SPIModule = DEFAULT_DSPI_MODULE,
    uint32_t Baudrate = 2000000,
    uint8_t QueueBitSize = 8,
    uint8_t CS = 0x00,
    uint8_t CSPol = 0x0F,
    uint8_t ClkPolarity = 0,
    uint8_t ClkPhase = 1,
    BOOL DoutHiz = TRUE,
    uint8_t QCD = 0,
    uint8_t DTL = 0)
```

```
#include <dspi.h>
```

Initialize a DSPI module.

Notes:

- The maximum baud rate is CPU_CLOCK / 3
- Will initialize to the highest available baud rate that does not exceed the maximum
- If configured for 8 bits per transfer then the data must be uint8_t aligned
- If configured for greater than 8 bits per transfer then the data must be uint16_t aligned
- If configured for greater than 16 bits per transfer then the data must be uint32_t aligned
- Chip select is based on chipSelectPolarity
- 0 data captured leading edge of DSPI_CLK, changed following edge.
- 1 data changed leading edge of DSPI_CLK, captured following edge.
- 0 default is as close to 1/2 DSPI_CLK without going under, keeping with the interface to SPI
- 0 default is 17/(system clock / 2), in keeping with interface to QSPI

Parameters

<i>SPIModule</i>	SPI module number, 0 - 1
<i>Baudrate</i>	Maximum baud rate requested
<i>QueueBitSize</i>	Number of bits per transfer: 8, 16 or 32
<i>CS</i>	SPI chip selects to use for transfer
<i>CSPol</i>	0 = inactive logic level low, 1 = high
<i>ClkPolarity</i>	0 = inactive logic level low, 1 = high
<i>ClkPhase</i>	0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge.
<i>DoutHiz</i>	Data output high impedance between transfers
<i>QCD</i>	Delay from chip select to valid clock (default is 0). QCD is a value in the QDLYR register and will change the delay between the assertion of the chip select and the start of the DSPI clock. Default setting of one half DSPI clk will be used if parameter is specified as 0x0 or not included.
<i>DTL</i>	DTL is a value in the QDLYR register and will change the delay following a transfer of a single uint16_t in the DSPI queue. Default reset value of 17/(fsys/2) will be used if parameter is specified 0 or not included.

Returns

Current stat of DSPI bus [MOD5441xDspiStates](#)

14.43.2.4.3 DSPIStart()

```
uint8_t DSPIStart (
    uint8_t SPIModule,
    puint8_t transmitBufferPtr,
    volatile uint8_t * receiveBufferPtr,
    uint32_t byteCount,
    OS_SEM * finishedSem = NULL,
    uint8_t enableDMA = TRUE,
    int csReturnToInactive = DEASSERT_AFTER_LAST)
```

#include <dspi.h>

Start DSPI Data Transfer.

Parameters:

transmitBufferPtr	- Buffer with data to send, NULL for receiving
receiveBufferPtr	- Buffer to receive data, NULL for sending
byteCount	- Count of bytes to send or receive
SPIModule	- Selects the dspi module to use
finishedSem	- Optional semaphore to set when completed

Return:

Current DSPI state - DSPI_OK success or DSPI_BUSY busy

Notes:

None

14.43.2.4.4 QSPIdone()

```
BOOL QSPIdone (
    void ) [inline]
```

#include <dspi.h>

Can be called after [QSPIStart\(\)](#). Returns TRUE when transfer is complete. This is an alternative to using a semaphore.

Return values

<i>TRUE</i>	When transfer is complete
<i>FALSE</i>	On error

See also

[QSPInit\(\)](#)[QSPStart\(\)](#)

14.43.2.4.5 QSPInit()

```
uint8_t QSPInit (
    uint32_t baudRateInBps = 2000000,
    uint8_t transferSizeInBits = 8,
    uint8_t peripheralChipSelects = 0x0F,
    uint8_t chipSelectPolarity = 1,
    uint8_t clockPolarity = 0,
    uint8_t clockPhase = 1,
    BOOL doutHiz = TRUE,
    uint8_t csToClockDelay = 0,
    uint8_t delayAfterTransfer = 0) [inline]
#include <dsapi.h>
```

Initialize Queued Serial Peripheral Interface (QSPI)

Notes: (3) Transfers > 8 must be word aligned (4) Select based on chipSelectPolarity (5) 0 data captured leading edge of QSPI_CLK, changed following edge. (6) 1 data changed leading edge of QSPI_CLK, captured following edge. (7) 0 default is 1/2 QSPI_CLK or see MFCXXXX reference manual (8) 0 default is 8192/(system clock / 2) see MFCXXXX reference manual

Parameters

<i>baudRateInBps</i>	Maximum requested baud rate in bits per second. The maximum possible baud rate is determined by the system clock. The function will select the highest possible baud rate if the specified value cannot be achieved.
<i>transferSizeInBits</i>	Size of data values to be transferred. Values can be 8, 16 or 32 bits. Note that 16 and 32 bit size values must be word (16-bit) aligned (16-bit).
<i>peripheralChipSelects</i>	Peripheral chip select drive level. Used to select an external device for serial data transfer. More than one chip select may be active at once, and more than one device can be connected to each chip select. Bits 3-0 map directly to QSPI_CS[3:0], respectively. For each bit: • 0 = Chip select is 0 during a transfer. • 1 = Chip select is 1 during a transfer.
<i>chipSelectPolarity</i>	Peripheral chip select inactive level (no transfer in progress). Applies to all chip selects. • 0 = 0 when inactive • 1 = 1 when inactive
<i>clockPolarity</i>	Clock polarity: • 0 = The inactive state value of QSPI_CLK is logic level 0. • 1 = The inactive state value of QSPI_CLK is logic level 1.

Parameters

<i>clockPhase</i>	Clock phase: • 0 = Data captured on the leading edge of QSPI_CLK and changed on the following edge of QSPI_CLK. • 1 = Data changed on the leading edge of QSPI_CLK and captured on the following edge of QSPI_CLK.
<i>doutHiz</i>	Data output high impedance enable. Selects QSPI_DOUT mode of operation. • 0 = Default value after reset. QSPI_DOUT is actively driven between transfers. • 1 = QSPI_DOUT is high impedance between transfers.
<i>csToClockDelay</i>	QCD, QSPICLK delay. When the DSCK bit in the command RAM is set this field determines the length of the delay from assertion of the chip selects to valid QSPI_CLK transition.
<i>delayAfterTransfer</i>	Delay after transfer. When the DT bit in the command RAM is set this field determines the length of delay after the serial transfer.

Returns

groupQspiState

See also

[QSPiInit\(\)](#)

[QSPiStart\(\)](#)

14.43.2.4.6 QSPiStart()

```
uint8_t QSPiStart (
    uint8_t transmitBufferPtr,
    volatile uint8_t * receiveBufferPtr,
    uint32_t byteCount,
    OS_SEM * finishedSem = NULL) [inline]
#include <dsapi.h>
Start QSPI Data Transfer.
```

Parameters

<i>transmitBufferPtr</i>	Pointer to the buffer to use in the transfer. Specify NULL for receive only.
<i>receiveBufferPtr</i>	Pointer to buffer to store received data. NULL for transmit only.
<i>byteCount</i>	Number of bytes to send or receive. If the data size is greater than 8-bit, you must provide the total number of bytes.
<i>finishedSem</i>	Pointer to preinitialized semaphore to post to when transfer is complete. A value of NULL disables the semaphore function.

Returns

groupQspiState

See also

[QSPiInit\(\)](#)

[QSPiDone\(\)](#)

14.43.3 MCF5441x (Queued SPI 2.x compatibility layer)

QSPI for MCF5441x (MOD5441x, NANO, SB800EX)

Topics

- [QSPI state](#)

Classes

- struct [QSPI_Record](#)

This struct contains the major variables/configurations used for a QSPI transfer.

Functions

- `uint8_t QSPIInit (uint32_t baudRateInBps=2000000, uint8_t transferSizeInBits=8, uint8_t peripheralChipSelects=0x0F, uint8_t chipSelectPolarity=1, uint8_t clockPolarity=0, uint8_t clockPhase=1, BOOL doutHiz=TRUE, uint8_t csToClockDelay=0, uint8_t delayAfterTransfer=0)`

Initialize Queued Serial Peripheral Interface (QSPI)

- `uint8_t QSPIStart (puint8_t transmitBufferPtr, volatile uint8_t *receiveBufferPtr, uint32_t byteCount, OS_SEM *finishedSem=NULL)`

Start QSPI Data Transfer.

- `BOOL QSPIdone (void)`

Can be called after [QSPIStart\(\)](#). Returns TRUE when transfer is complete. This is an alternative to using a semaphore.

14.43.3.1 Detailed Description

QSPI for MCF5441x (MOD5441x, NANO, SB800EX)

```
#include< qspi.h>
```

Supported platforms:

- MOD54415
- MOD54417
- NANO54415
- SB800EX

Note

The SPI interface modules (up to 4 depending on the platform) on the MCF5441x processor are termed DMA SPI (DSPI). Earlier ColdFire processors had a single SPI with a queue, and were termed QSPI. This header file is here as a guide for customers using earlier ColdFire processors to make porting code easier. However, we highly recommend using the DSPI driver instead. The calls are nearly identical.

Introduction

The Queued Serial Peripheral Interface (QSPI) is a full duplex synchronous serial data link that operates in master/slave mode. The QSPI is similar to SPI, but also provided a hardware queue for transmit data, receive data, and commands. Up to 16 transfers can be queued at one time, eliminating CPU intervention between transfers. The NetBurner QSPI driver and API provide interrupt driven QSPI communication.

The QSPI specification allows for sharing the QSPI bus with multiple devices. However, when designing your system you must consider what devices are connected and the desired throughput. For example, if you are using the Embedded Flash File System (EFlash) with external SD flash cards, or the WiFi module, they will require exclusive use of the QSPI.

The QSPI uses the following hardware signals:

- SCLK Serial clock output from master
- MOSI Master data output (transmit), slave data input (receive)
- MISO Master data input (receive), slave data output (transmit)
- QSPI_CS Optional QSPI chip selects

Programming tip: If you are new to QSPI and require a chip select for your device, we recommend using a GPIO signal during initial development to simplify your application code. Once proper operation has been achieved, modify the code to use the QSPI chip selects.

QSPI Configuration and Initialization

Use of the QSPI requires:

1. Identify which pins on your NetBurner device provide primary or secondary QSPI functionality. Use the NetBurner Pins class functions to configure these pins for QSPI operation.
2. Single task applications: you can choose to call the [QSPIdone\(\)](#) function to determine when the QSPI transfer is complete, or use a semaphore and pend for completion.
3. Multi-task applications: If your application has more than one task that will access the QSPI, you must create a semaphore to control access to the QSPI resource.
4. Initialize the QSPI with the [QSPInit\(\)](#) function.
5. When you have data to send, or want to read data from the QSPI slave device, call the [QSPIStart\(\)](#) function to initiate the data transfer.
6. The semaphore will post when the data transfer is complete.

14.43.3.2 Function Documentation

14.43.3.2.1 QSPIdone()

```

BOOL QSPIdone (
    void ) [inline]
#include <qspi.h>

```

Can be called after [QSPIStart\(\)](#). Returns TRUE when transfer is complete. This is an alternative to using a semaphore.

Return values

<i>TRUE</i>	When transfer is complete
<i>FALSE</i>	On error

See also

[QSPInit\(\)](#)

[QSPIStart\(\)](#)

Can be called after [QSPIStart\(\)](#). Returns TRUE when transfer is complete. This is an alternative to using a semaphore.

Returns

true if DSPI is finished, false if active

14.43.3.2.2 QSPInit()

```
uint8_t QSPInit (
    uint32_t baudRateInBps = 2000000,
    uint8_t transferSizeInBits = 8,
    uint8_t peripheralChipSelects = 0x0F,
    uint8_t chipSelectPolarity = 1,
    uint8_t clockPolarity = 0,
    uint8_t clockPhase = 1,
    BOOL doutHiz = TRUE,
    uint8_t csToClockDelay = 0,
    uint8_t delayAfterTransfer = 0) [inline]
```

```
#include <qspi.h>
```

Initialize Queued Serial Peripheral Interface (QSPI)

Notes: (3) Transfers > 8 must be word aligned (4) Select based on chipSelectPolarity (5) 0 data captured leading edge of QSPI_CLK, changed following edge. (6) 1 data changed leading edge of QSPI_CLK, captured following edge. (7) 0 default is 1/2 QSPI_CLK or see MFCXXXX reference manual (8) 0 default is 8192/(system clock / 2) see MFCXXXX reference manual

Parameters

<i>baudRateInBps</i>	Maximum requested baud rate in bits per second. The maximum possible baud rate is determined by the system clock. The function will select the highest possible baud rate if the specified value cannot be achieved.
<i>transferSizeInBits</i>	Size of data values to be transferred. Values can be 8, 16 or 32 bits. Note that 16 and 32 bit size values must be word (16-bit) aligned (16-bit).
<i>peripheralChipSelects</i>	Peripheral chip select drive level. Used to select an external device for serial data transfer. More than one chip select may be active at once, and more than one device can be connected to each chip select. Bits 3-0 map directly to QSPI_CS[3:0], respectively. For each bit: • 0 = Chip select is 0 during a transfer. • 1 = Chip select is 1 during a transfer.
<i>chipSelectPolarity</i>	Peripheral chip select inactive level (no transfer in progress). Applies to all chip selects. • 0 = 0 when inactive • 1 = 1 when inactive
<i>clockPolarity</i>	Clock polarity: • 0 = The inactive state value of QSPI_CLK is logic level 0. • 1 = The inactive state value of QSPI_CLK is logic level 1.
<i>clockPhase</i>	Clock phase: • 0 = Data captured on the leading edge of QSPI_CLK and changed on the following edge of QSPI_CLK. • 1 = Data changed on the leading edge of QSPI_CLK and captured on the following edge of QSPI_CLK.
<i>doutHiz</i>	Data output high impedance enable. Selects QSPI_DOUT mode of operation. • 0 = Default value after reset. QSPI_DOUT is actively driven between transfers. • 1 = QSPI_DOUT is high impedance between transfers.

Parameters

<i>csToClockDelay</i>	QCD, QSPICLK delay. When the DSCK bit in the command RAM is set this field determines the length of the delay from assertion of the chip selects to valid QSPI_CLK transition.
<i>delayAfterTransfer</i>	Delay after transfer. When the DT bit in the command RAM is set this field determines the length of delay after the serial transfer.

Returns

groupQspiState

See also

[QSPiInit\(\)](#)

[QSPiStart\(\)](#)

14.43.3.2.3 QSPiStart()

```
uint8_t QSPiStart (
    puint8_t transmitBufferPtr,
    volatile uint8_t * receiveBufferPtr,
    uint32_t byteCount,
    OS_SEM * finishedSem = NULL) [inline]
#include <qspi.h>
Start QSPI Data Transfer.
```

Parameters

<i>transmitBufferPtr</i>	Pointer to the buffer to use in the transfer. Specify NULL for receive only.
<i>receiveBufferPtr</i>	Pointer to buffer to store received data. NULL for transmit only.
<i>byteCount</i>	Number of bytes to send or receive. If the data size is greater than 8-bit, you must provide the total number of bytes.
<i>finishedSem</i>	Pointer to preinitialized semaphore to post to when transfer is complete. A value of NULL disables the semaphore function.

Returns

groupQspiState

See also

[QSPiInit\(\)](#)

[QSPiDone\(\)](#)

14.43.3.3 QSPI state

Macros

- #define **QSPI_OK** (0)
QSPI is busy.
- #define **QSPI_BUSY** (1)
QSPI is ready.

14.43.3.3.1 Detailed Description

```
#include< qspi.h>
```

14.43.4 MIMX10xx (DSPI)

DSPI for ARM MIMX10xx (SOMRT1061)

Topics

- [DspiModuleNumber](#)
DSPI Peripheral Module .
- [DspiState](#)
DSPI Bus State .

Classes

- struct [_lpspi_master_edma_private_handle](#)
Structure definition for dspi_master_edma_private_handle_t. The structure is private.
- struct [dspiDMAStruct](#)
DSPI DMA structure.
- struct [spiDriverStruct](#)
Variables/configurations for a DSPI transfer.
- class [SPIModule](#)
SPI Peripheral Module Class.

Typedefs

- typedef struct [_lpspi_master_edma_private_handle](#) [lpspi_master_edma_private_handle_t](#)
Structure definition for dspi_master_edma_private_handle_t. The structure is private.

Enumerations

- enum [csReturnType](#) { [DEASSERT_NEVER](#) = 0 , [DEASSERT_AFTER_LAST](#) = 1 , [DEASSERT_EVERY_TRANSFER](#) = 2 }
 - enum [spiChipSelect](#) { [CHIP_SELECT_0](#) = 0 , [CHIP_SELECT_1](#) = 1 , [CHIP_SELECT_2](#) = 2 , [CHIP_SELECT_3](#) = 3 , [CHIP_SELECT_DISABLED](#) = 0xFF }
 - enum [spiChipSelectPolarity](#) { [CS_ASSERT_LOW](#) = 0 , [CS_ASSERT_HIGH](#) = 1 }
- Chip select deassertion modes. Used to determine when the driver should deassert chip selects during SPI transfer.*
- Chip select number.*
- Chip select polarity.*

Functions

- uint8_t [DSPIInit](#) (uint8_t [SPIModule](#)=[DEFAULT_DSPI_MODULE](#), uint32_t Baudrate=2000000, uint8_t QueueBitSize=8, uint8_t CS=0x00, uint8_t CSPol=0x0F, uint8_t ClkPolarity=0, uint8_t ClkPhase=1, BOOL Douthiz=TRUE, uint8_t QCD=0, uint8_t DTL=0)
Initialize a DSPI module.
- uint8_t [DSPIInit](#) (uint8_t [SPIModule](#)=[DEFAULT_DSPI_MODULE](#), uint32_t Baudrate=2000000, uint8_t QueueBitSize=8, [PinIO](#) gpioChipSelect=PinDef_INVALID, uint8_t CSPol=0x0F, uint8_t ClkPolarity=0, uint8_t ClkPhase=1, BOOL Douthiz=TRUE, uint8_t QCD=0, uint8_t DTL=0)
Initialize a DSPI module.

- `uint8_t DSPIStart (uint8_t SPIModule, uint8_t transmitBufferPtr, volatile uint8_t *receiveBufferPtr, uint32_t byteCount, OS_SEM *finishedSem=NULL, uint8_t enableDMA=TRUE, int csReturnToInactive=DEASSERT_AFTER_LAST)`
Start a DSPI transfer.
- `BOOL DSPIDone (uint8_t SPIModule=DEFAULT_DSPI_MODULE)`
Check SPI status.

Variables

- `LPSPi_Type * lpspiBases []`
Pointers to LPSPi_Type for each instance.

DMA CHANNEL NUMBERS

- `#define DMA_CH_DSPI_1_RX 0x06`
DMA Channel allocated to LPSPi1 Rx (DMA Channel 6)
- `#define DMA_CH_DSPI_1_TX 0x07`
DMA Channel allocated to LPSPi1 Tx (DMA Channel 7)
- `#define DMA_CH_DSPI_2_RX 0x04`
DMA Channel allocated to LPSPi2 Rx (DMA Channel 4)
- `#define DMA_CH_DSPI_2_TX 0x05`
DMA Channel allocated to LPSPi2 Tx (DMA Channel 5)
- `#define DMA_CH_DSPI_4_RX 0x08`
DMA Channel allocated to LPSPi4 Rx (DMA Channel 8)
- `#define DMA_CH_DSPI_4_TX 0x09`
DMA Channel allocated to LPSPi4 Tx (DMA Channel 9)

14.43.4.1 Detailed Description

DSPI for ARM MIMX10xx (SOMRT1061)

Note that for the MIMXRT10xx there are 3 types of SPI peripherals:

- SPI using the SPI peripheral: Provides the most configuration options. Used in single data channel mode: SCLK, MISO, MOSI, Chip Selects.
- SPI using the QSPI peripheral: Single data channel as with SPI, but fewer configuration options.
- SPI using the USART peripheral: Single data channel as with SPI, but fewer configuration options.

14.43.4.2 Enumeration Type Documentation

14.43.4.2.1 csReturnType

```
enum csReturnType
```

```
#include <dspi.h>
```

Chip select deassertion modes. Used to determine when the driver should deassert chip selects during SPI transfer.

Enumerator

DEASSERT_NEVER	The chip select used for the transaction should remain asserted, even after the transaction is complete.
DEASSERT_AFTER_LAST	The chip select should remain asserted for the full duration of the transaction, and only be deasserted after the final transfer.
DEASSERT_EVERY_TRANSFER	The chip select should be deasserted between every transfer within the transaction. Deassert chip select After every transfer.

14.43.4.2.2 spiChipSelect

```
enum spiChipSelect
#include <dspi.h>
```

Chip select number.

If supported by the SPI peripheral, these values can be used to configure the chip select to be used during a SPI transfer.

Enumerator

CHIP_SELECT_0	Configure the SPI peripheral to use chip select 0.
CHIP_SELECT_1	Configure the SPI peripheral to use chip select 1.
CHIP_SELECT_2	Configure the SPI peripheral to use chip select 2.
CHIP_SELECT_3	Configure the SPI peripheral to use chip select 3.
CHIP_SELECT_DISABLED	Configure the SPI peripheral to disable chip select.

14.43.4.2.3 spiChipSelectPolarity

```
enum spiChipSelectPolarity
#include <dspi.h>
```

Chip select polarity.

This enum exists for API compatibility between platforms. The MIMXRT10xx's SPI peripheral does not support modifying chip select polarity. The SPI peripheral only supports asserting chip select to logic level LOW during a transaction. If you need to assert the chip select to logic level HIGH during a transaction, your alternative is to use a GPIO for chip select.

Enumerator

CS_ASSERT_LOW	Assert all chip selects to logic level LOW during a SPI transaction.
CS_ASSERT_HIGH	Assert all chip selects to logic level HIGH during a SPI transaction.

14.43.4.3 Function Documentation

14.43.4.3.1 DSPIDone()

```
BOOL DSPIDone (
    uint8_t SPIModule = DEFAULT_DSPI_MODULE)
#include <dspi.h>
```

Check SPI status.

Returns

true if DSPI is finished, false if active

14.43.4.3.2 DSPIInit() [1/2]

```
uint8_t DSPIInit (
    uint8_t SPIModule = DEFAULT_DSPI_MODULE,
    uint32_t Baudrate = 2000000,
    uint8_t QueueBitSize = 8,
    PinIO gpioChipSelect = PinDef_INVALID,
    uint8_t CSPol = 0x0F,
    uint8_t ClkPolarity = 0,
    uint8_t ClkPhase = 1,
    BOOL DoutHiz = TRUE,
    uint8_t QCD = 0,
    uint8_t DTL = 0)
```



```
#include <dspi.h>
```

Initialize a DSPI module.

Notes:

- The maximum baud rate is CPU_CLOCK / 3
- Will initialize to the highest available baud rate that does not exceed the maximum
- If configured for 8 bits per transfer then the data must be uint8_t aligned
- If configured for greater than 8 bits per transfer then the data must be uint16_t aligned
- If configured for greater than 16 bits per transfer then the data must be uint32_t aligned
- Chip select is based on chipSelectPolarity
- 0 data captured leading edge of DSPI_CLK, changed following edge.
- 1 data changed leading edge of DSPI_CLK, captured following edge.
- 0 default is as close to 1/2 DSPI_CLK without going under, keeping with the interface to SPI
- 0 default is 17/(system clock / 2), in keeping with interface to QSPI

Parameters

<i>SPIModule</i>	SPI module number, 0 - 1
<i>Baudrate</i>	Maximum baud rate requested
<i>QueueBitSize</i>	Number of bits per transfer: 8, 16 or 32
<i>gpioChipSelect</i>	PinIO to be configured as GPIO and used as chip select. Only supports spiDriverStruct::csReturnToInactive values of DEASSERT_AFTER_LAST and DEASSERT_NEVER
<i>CSPol</i>	0 = inactive logic level low, 1 = high
<i>ClkPolarity</i>	0 = inactive logic level low, 1 = high
<i>ClkPhase</i>	0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge.
<i>DoutHiz</i>	Data output high impedance between transfers
<i>QCD</i>	Delay from chip select to valid clock (default is 0)
<i>DTL</i>	Chip select mode dsppiChipSelectMode

Returns

Current stat of DSPI bus [dsppiState](#)

14.43.4.3.3 DSPIInit() [2/2]

```
uint8_t DSPIInit (
    uint8_t SPIModule = DEFAULT_DSPI_MODULE,
    uint32_t Baudrate = 2000000,
    uint8_t QueueBitSize = 8,
    uint8_t CS = 0x00,
    uint8_t CSPol = 0x0F,
    uint8_t ClkPolarity = 0,
    uint8_t ClkPhase = 1,
    BOOL DoutHiz = TRUE,
    uint8_t QCD = 0,
    uint8_t DTL = 0)
```

```
#include <dspi.h>
```

Initialize a DSPI module.

Notes:

- The maximum baud rate is $\text{CPU\_CLOCK} / 3$
- Will initialize to the highest available baud rate that does not exceed the maximum
- If configured for 8 bits per transfer then the data must be `uint8_t` aligned
- If configured for greater than 8 bits per transfer then the data must be `uint16_t` aligned
- If configured for greater than 16 bits per transfer then the data must be `uint32_t` aligned
- Chip select is based on `chipSelectPolarity`
- 0 data captured leading edge of `DSPI_CLK`, changed following edge.
- 1 data changed leading edge of `DSPI_CLK`, captured following edge.
- 0 default is as close to $1/2 \text{ DSPI\_CLK}$ without going under, keeping with the interface to SPI
- 0 default is $17/(\text{system clock} / 2)$, in keeping with interface to QSPI

Parameters

<i>SPIModule</i>	SPI module number, 0 - 1
<i>Baudrate</i>	Maximum baud rate requested
<i>QueueBitSize</i>	Number of bits per transfer: 8, 16 or 32
<i>CS</i>	SPI chip selects to use for transfer
<i>CSPol</i>	0 = inactive logic level low, 1 = high
<i>ClkPolarity</i>	0 = inactive logic level low, 1 = high
<i>ClkPhase</i>	0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge.
<i>DoutHiz</i>	Data output high impedance between transfers
<i>QCD</i>	Delay from chip select to valid clock (default is 0)
<i>DTL</i>	Chip select mode dspicChipSelectMode

Returns

Current stat of DSPI bus [dspistate](#)

14.43.4.3.4 DSPIStart()

```
uint8_t DSPIStart (
    uint8_t SPIModule,
    puint8_t transmitBufferPtr,
    volatile uint8_t * receiveBufferPtr,
    uint32_t byteCount,
    OS_SEM * finishedSem = NULL,
    uint8_t enableDMA = TRUE,
    int csReturnToInactive = DEASSERT_AFTER_LAST)
```

#include <dsapi.h>

Start a DSPI transfer.

- If configured for 8 bits per transfer then the data must be `uint8_t` aligned
- If configured for > than 8 bits per transfer then the data must be `uint16_t` aligned
- If configured for > than 16 bits per transfer then the data must be `uint32_t` aligned
- If either RX or TX pointer is assigned 'null' then that communication direction will not occur.
- If `DSPI_Finished` points to a semaphore, then the DSPI will POST to it when the transfer is complete.
- The semaphore is optional, but it can increase efficiency.

Parameters

<i>SPIModule</i>	DSPI module to use, 0 - 1
<i>transmitBufferPtr</i>	Pointer to the buffer containing the data to transmit
<i>receiveBufferPtr</i>	Pointer to the buffer to store the received data
<i>byteCount</i>	Number of bytes to transmit
<i>finishedSem</i>	Optional semaphore to post to when finished
<i>enableDMA</i>	Enable DMA transfers
<i>csReturnToInactive</i>	Chip select state

Returns

The current state of the SPI bus [dspiParam](#)

14.43.4.4 DspiModuleNumber

DSPI Peripheral Module .

Macros

- #define **DEFAULT_DSPI_MODULE** 2
Default DSPI module.
- #define **DSPI_MODULE_COUNT** 4
Number of modules: LPSPi modules 1, 2, 4 are available to the user. LPSPi 3 is dedicated to SPiFlash.

14.43.4.4.1 Detailed Description

DSPI Peripheral Module .

14.43.4.5 DspiState

DSPI Bus State .

Macros

- #define **DSPI_OK** (0)
DSPI OK.
- #define **DSPI_BUSY** (1)
DSPI Busy.
- #define **DSPI_ERROR** (2)
DSPI Error.
- #define **DSPI_HW_CFG_ERR** (3)
DSPI Hardware Config Error: Rx with DEASSERT_NEVER is only supported if a GPIO Chip Select is configured.

14.43.4.5.1 Detailed Description

DSPI Bus State .

14.43.5 SAME70 (DSPI)

DSPI for ARM SAME70 (MODM7AE70, SBE70LC)

Classes

- struct [spiDriverStruct](#)
Variables/configurations for a DSPI transfer.
- class [SPIModule](#)
SPI Peripheral Module Class.

Enumerations

- enum **csReturnType** { **DEASSERT_NEVER** = 0 , **DEASSERT_AFTER_LAST** = 1 , **DEASSERT EVERY_TRANSFER** = 2 }
Chip select deassertion modes. Used to determine when the driver should deassert chip selects during SPI transfer.
- enum **spiChipSelectPolarity** { **CS_ASSERT_LOW** = 0 , **CS_ASSERT_HIGH** = 1 }
Chip select polarity.

Functions

- uint8_t **DSPIInit** (uint8_t **SPIModule**=**DEFAULT_DSPI_MODULE**, uint32_t Baudrate=2000000, uint8_t QueueBitSize=8, uint8_t CS=0x00, uint8_t CSPol=0x0F, uint8_t ClkPolarity=0, uint8_t ClkPhase=1, BOOL DoutHiz=TRUE, uint8_t QCD=0, uint8_t DTL=0)
Initialize a DSPI module.
- uint8_t **DSPIStart** (uint8_t **SPIModule**, uint8_t transmitBufferPtr, volatile uint8_t *receiveBufferPtr, uint32_t byteCount, OS_SEM *finishedSem=NULL, uint8_t enableDMA=TRUE, int csReturnToInactive=**DEASSERT_AFTER_LAST**)
Start a DSPI transfer.
- BOOL **DSPIDone** (uint8_t **SPIModule**=**DEFAULT_DSPI_MODULE**)
Check SPI status.

Variables

- const uint8_t **CHIP_SELECT_0** = 0
Chip select number.
- const uint8_t **CHIP_SELECT_1** = 1
Configure the SPI peripheral to use chip select 1.
- const uint8_t **CHIP_SELECT_2** = 2
Configure the SPI peripheral to use chip select 2.
- const uint8_t **CHIP_SELECT_3** = 3
Configure the SPI peripheral to use chip select 3.
- const uint8_t **CHIP_SELECT_DISABLED** = 0xFF
Or flag to set multi bit chipselect.

QSPI to DSPI Translation macros

(Compatibility function for previous drivers, note that the 'Q' stands for Queued SPI)

- uint8_t **QSPIInit** (uint32_t baudRateInBps=2000000, uint8_t transferSizeInBits=8, uint8_t peripheralChipSelects=0x0F, uint8_t chipSelectPolarity=1, uint8_t clockPolarity=0, uint8_t clockPhase=1, BOOL doutHiz=TRUE, uint8_t csToClockDelay=0, uint8_t delayAfterTransfer=0)
Initialize Queued Serial Peripheral Interface (QSPI)
- uint8_t **QSPIStart** (uint8_t transmitBufferPtr, volatile uint8_t *receiveBufferPtr, uint32_t byteCount, OS_SEM *finishedSem=NULL)
Start QSPI Data Transfer.
- BOOL **QSPIDone** ()
Check SPI status. (Compatibility function for previous drivers, note that the 'Q' stands for Queued SPI)

DSPI Bus State

- #define **DSPI_OK** (0)
DSPI OK.
- #define **DSPI_BUSY** (1)
DSPI Busy.
- #define **DSPI_ERROR** (2)
DSPI Error.

DSPI Peripheral Module

- #define **DEFAULT_DSPI_MODULE** 0
Default DSPI module.
- #define **DSPI_MODULE_COUNT** 1
Number of modules: 0, 1.

14.43.5.1 Detailed Description

DSPI for ARM SAME70 (MODM7AE70, SBE70LC)

Supported Platforms:

- MODM7AE70
- SBE70LC

Note that for the SAME70 there are 3 types of SPI peripherals:

- SPI using the SPI peripheral: Provides the most configuration options. Used in single data channel mode: SCLK, MISO, MOSI, Chip Selects.
- SPI using the QSPI peripheral: Single data channel as with SPI, but fewer configuration options.
- SPI using the USART peripheral: Single data channel as with SPI, but fewer configuration options.

14.43.5.2 Enumeration Type Documentation

14.43.5.2.1 csReturnType

```
enum csReturnType
```

```
#include <dspi.h>
```

Chip select deassertion modes. Used to determine when the driver should deassert chip selects during SPI transfer.

Enumerator

DEASSERT_NEVER	The chip select used for the transaction should remain asserted, even after the transaction is complete.
DEASSERT_AFTER_LAST	The chip select should remain asserted for the full duration of the transaction, and only be deasserted after the final transfer.
DEASSERT_EVERY_TRANSFER	The chip select should be deasserted between every transfer within the transaction. Deassert chip select After every transfer.

14.43.5.2.2 spiChipSelectPolarity

```
enum spiChipSelectPolarity
```

```
#include <dspi.h>
```

Chip select polarity.

This enum exists for API compatibility between platforms. The SAME70's SPI peripheral does not support modifying chip select polarity. The SPI peripheral only supports asserting chip select to logic level LOW during a transaction. If you need to assert the chip select to logic level HIGH during a transaction, your alternative is to use a GPIO for chip select.

Enumerator

CS_ASSERT_LOW	Assert all chip selects to logic level LOW during a SPI transaction.
CS_ASSERT_HIGH	Assert all chip selects to logic level HIGH during a SPI transaction.

14.43.5.3 Function Documentation

14.43.5.3.1 DSPIdone()

```

BOOL DSPIdone (
    uint8_t SPIModule = DEFAULT_DSPI_MODULE)
#include <dspi.h>
Check SPI status.

```

Returns

true if DSPI is finished, false if active

14.43.5.3.2 DSPIInit()

```

uint8_t DSPIInit (
    uint8_t SPIModule = DEFAULT_DSPI_MODULE,
    uint32_t Baudrate = 2000000,
    uint8_t QueueBitSize = 8,
    uint8_t CS = 0x00,
    uint8_t CSPol = 0x0F,
    uint8_t ClkPolarity = 0,
    uint8_t ClkPhase = 1,
    BOOL DoutHiz = TRUE,
    uint8_t QCD = 0,
    uint8_t DTL = 0)
#include <dspi.h>
Initialize a DSPI module.

```

Initialize a DSPI module.

Notes:

- The maximum baud rate is CPU_CLOCK / 3
- Will initialize to the highest available baud rate that does not exceed the maximum
- If configured for 8 bits per transfer then the data must be uint8_t aligned
- If configured for greater than 8 bits per transfer then the data must be uint16_t aligned
- If configured for greater than 16 bits per transfer then the data must be uint32_t aligned
- Chip select is based on chipSelectPolarity
- 0 data captured leading edge of DSPI_CLK, changed following edge.
- 1 data changed leading edge of DSPI_CLK, captured following edge.
- 0 default is as close to 1/2 DSPI_CLK without going under, keeping with the interface to SPI
- 0 default is 17/(system clock / 2), in keeping with interface to QSPI

Parameters

<i>SPIModule</i>	SPI module number, 0 - 1
<i>Baudrate</i>	Maximum baud rate requested
<i>QueueBitSize</i>	Number of bits per transfer: 8, 16 or 32
<i>CS</i>	SPI chip selects to use for transfer
<i>CSPol</i>	0 = inactive logic level low, 1 = high
<i>ClkPolarity</i>	0 = inactive logic level low, 1 = high
<i>ClkPhase</i>	0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge.
<i>DoutHiz</i>	Data output high impedance between transfers

<i>QCD</i>	Delay from chip select to valid clock (default is 0)
<i>DTL</i>	Chip select mode dsPICChipSelectMode

Returns

Current stat of DSPI bus [dsPICState](#)

14.43.5.3.3 DSPIStart()

```
uint8_t DSPIStart (
    uint8_t SPIModule,
    uint8_t * transmitBufferPtr,
    volatile uint8_t * receiveBufferPtr,
    uint32_t byteCount,
    OS_SEM * finishedSem = NULL,
    uint8_t enableDMA = TRUE,
    int csReturnToInactive = DEASSERT_AFTER_LAST)
```

```
#include <dsPIC.h>
```

Start a DSPI transfer.

- If configured for 8 bits per transfer then the data must be uint8_t aligned
- If configured for > than 8 bits per transfer then the data must be uint16_t aligned
- If configured for > than 16 bits per transfer then the data must be uint32_t aligned
- If either RX or TX pointer is assigned 'null' then that communication direction will not occur.
- If DSPI_Finished points to a semaphore, then the DSPI will POST to it when the transfer is complete.
- The semaphore is optional, but it can increase efficiency.

Parameters

SPIModule	DSPI module to use, 0 - 1
<i>transmitBufferPtr</i>	Pointer to the buffer containing the data to transmit
<i>receiveBufferPtr</i>	Pointer to the buffer to store the received data
<i>byteCount</i>	Number of bytes to transmit
<i>finishedSem</i>	Optional semaphore to post to when finished
<i>enableDMA</i>	Enable DMA transfers
<i>csReturnToInactive</i>	Chip select state

Returns

The current state of the SPI bus [dsPICState](#)

14.43.5.3.4 QSPIDone()

```
BOOL QSPIDone (
    void ) [inline]
#include <dsPIC.h>
```

Check SPI status. (Compatibility function for previous drivers, note that the 'Q' stands for Queued SPI)

Can be called after [QSPIDone\(\)](#). Returns TRUE when transfer is complete. This is an alternative to using a semaphore.

Returns

true if DSPI is finished, false if active

14.43.5.3.5 QSPIInit()

```
uint8_t QSPIInit (
    uint32_t baudRateInBps = 2000000,
    uint8_t transferSizeInBits = 8,
    uint8_t peripheralChipSelects = 0x0F,
    uint8_t chipSelectPolarity = 1,
    uint8_t clockPolarity = 0,
    uint8_t clockPhase = 1,
    BOOL doutHiz = TRUE,
    uint8_t csToClockDelay = 0,
    uint8_t delayAfterTransfer = 0) [inline]
#include <dspi.h>
```

Initialize Queued Serial Peripheral Interface (QSPI)

Notes: (3) Transfers > 8 must be word aligned (4) Select based on chipSelectPolarity (5) 0 data captured leading edge of QSPI_CLK, changed following edge. (6) 1 data changed leading edge of QSPI_CLK, captured following edge. (7) 0 default is 1/2 QSPI_CLK or see MFCXXXX reference manual (8) 0 default is 8192/(system clock / 2) see MFCXXXX reference manual

Parameters

<i>baudRateInBps</i>	Maximum requested baud rate in bits per second. The maximum possible baud rate is determined by the system clock. The function will select the highest possible baud rate if the specified value cannot be achieved.
<i>transferSizeInBits</i>	Size of data values to be transferred. Values can be 8, 16 or 32 bits. Note that 16 and 32 bit size values must be word (16-bit) aligned (16-bit).
<i>peripheralChipSelects</i>	Peripheral chip select drive level. Used to select an external device for serial data transfer. More than one chip select may be active at once, and more than one device can be connected to each chip select. Bits 3-0 map directly to QSPI_CS[3:0], respectively. For each bit: • 0 = Chip select is 0 during a transfer. • 1 = Chip select is 1 during a transfer.
<i>chipSelectPolarity</i>	Peripheral chip select inactive level (no transfer in progress). Applies to all chip selects. • 0 = 0 when inactive • 1 = 1 when inactive
<i>clockPolarity</i>	Clock polarity: • 0 = The inactive state value of QSPI_CLK is logic level 0. • 1 = The inactive state value of QSPI_CLK is logic level 1.
<i>clockPhase</i>	Clock phase: • 0 = Data captured on the leading edge of QSPI_CLK and changed on the following edge of QSPI_CLK. • 1 = Data changed on the leading edge of QSPI_CLK and captured on the following edge of QSPI_CLK.
<i>doutHiz</i>	Data output high impedance enable. Selects QSPI_DOUT mode of operation. • 0 = Default value after reset. QSPI_DOUT is actively driven between transfers. • 1 = QSPI_DOUT is high impedance between transfers.

Parameters

<i>csToClockDelay</i>	QCD, QSPICLK delay. When the DSCK bit in the command RAM is set this field determines the length of the delay from assertion of the chip selects to valid QSPI_CLK transition.
<i>delayAfterTransfer</i>	Delay after transfer. When the DT bit in the command RAM is set this field determines the length of delay after the serial transfer.

Returns

groupQspiState

See also

[QSPInit\(\)](#)

[QSPiStart\(\)](#)

14.43.5.3.6 QSPiStart()

```
uint8_t QSPiStart (
    uint8_t transmitBufferPtr,
    volatile uint8_t * receiveBufferPtr,
    uint32_t byteCount,
    OS_SEM * finishedSem = NULL) [inline]
```

#include <dspi.h>

Start QSPI Data Transfer.

Parameters

<i>transmitBufferPtr</i>	Pointer to the buffer to use in the transfer. Specify NULL for receive only.
<i>receiveBufferPtr</i>	Pointer to buffer to store received data. NULL for transmit only.
<i>byteCount</i>	Number of bytes to send or receive. If the data size is greater than 8-bit, you must provide the total number of bytes.
<i>finishedSem</i>	Pointer to preinitialized semaphore to post to when transfer is complete. A value of NULL disables the semaphore function.

Returns

groupQspiState

See also

[QSPInit\(\)](#)

[QSPIdone\(\)](#)

14.43.5.4 Variable Documentation

14.43.5.4.1 CHIP_SELECT_0

```
const uint8_t CHIP_SELECT_0 = 0
```

```
#include <dspi.h>
```

Chip select number.

If supported by the SPI peripheral, these values can be used to configure the chip select to be used during a SPI transfer. Configure the SPI peripheral to use chip select 0.

14.43.5.4.2 CHIP_SELECT_DISABLED

```
const uint8_t CHIP_SELECT_DISABLED = 0xFF
#include <dspi.h>
```

Or flag to set multi bit chipselect.
Configure the SPI peripheral to disable chip select.

14.43.6 SAME70 (QuadSPI)

QSPI for SAME70 (MODM7AE70, SBE70LC)

Classes

- class [SPI_QSPI](#)
The Single-Bit SPI mode QSPI Peripheral Class.

QuadSPI Peripheral Module Definitions

- #define **DEFAULT_QUADSPI_MODULE** 0
Default QUADSPI module.
- #define **QUADSPI_MODULE_COUNT** 1
Number of modules: 0, 1.

14.43.6.1 Detailed Description

QSPI for SAME70 (MODM7AE70, SBE70LC)

Supported Platforms:

- MODM7AE70
- SBE70LC

The NetBurner ARM SAME70 Quad SPI (QSPI) API. Note that there are 3 types of SPI peripherals:

- SPI using the SPI peripheral: Provides the most configuration options. Used in single data channel mode: SCLK, MISO, MOSI, Chip Selects.
- SPI using the QuadSPI peripheral: Single data channel as with SPI, but fewer configuration options.
- SPI using the USART peripheral: Single data channel as with SPI, but fewer configuration options.

14.43.7 SAME70 (USART)

USART for SAME70 (MODM7AE70, SBE70LC)

Topics

- [UsartModuleNumber](#)
USART Peripheral Module .

Classes

- class [SPI_USART](#)
USART in SPI mode Peripheral Module Class.

14.43.7.1 Detailed Description

USART for SAME70 (MODM7AE70, SBE70LC)

Supported Platforms:

- MODM7AE70
- SBE70LC

Note that for the ARM SAME70 there are 3 types of SPI peripherals:

- SPI using the SPI peripheral: Provides the most configuration options. Used in single data channel mode: SCLK, MISO, MOSI, Chip Selects.
- SPI using the QSPI peripheral: Single data channel as with SPI, but fewer configuration options.
- SPI using the USART peripheral: Single data channel as with SPI, but fewer configuration options.

14.43.7.2 UsartModuleNumber

USART Peripheral Module .

Macros

- #define **DEFAULT_USART_SPI_MODULE** 0
Default QUADSPI module.
- #define **USART_SPI_MODULE_COUNT** 2
Number of modules: 0, 1.

14.43.7.2.1 Detailed Description

USART Peripheral Module .

14.44 SSH

Topics

- [SSH Error Codes](#)

Typedefs

- typedef int(* [sshUserAuthenticateFn](#)) (const char *usernamePtr, const char *passwordPtr)
[DEPRECATED] User provided SSH username and password authenticate routine for a server. Please consider sshUserAuthenticateWithTypeFn.
- typedef int(* [sshUserAuthenticateWithTypeFn](#)) (const char *usernamePtr, const char *authValPtr, [AuthType](#) authType)
User provided SSH user authenticate routine for a server.
- typedef int(* [sshGetUserPwFn](#)) (const [NBString](#) &usernamePtr, [NBString](#) &passwordPtr)
User provided SSH user password authentication routine for clients.
- typedef int(* [sshGetUserKeyFn](#)) (const [NBString](#) &usernamePtr, [NBString](#) &publicKey, [NBString](#) &privateKey, [NBString](#) &keyType)
User provided SSH user key authenticate routine for clients.
- typedef int(* [sshUserGetKeyFn](#)) (int keyRequested, const unsigned char **keyBufferPtr, int *keyLengthPtr)
The user defined callback to get the server key used during the initial SSH negotiation.

Functions

- void [SshSetUserAuthenticate](#) ([sshUserAuthenticateFn](#) sshUserAuthenticateFnPtr)
[DEPRECATED] Sets the user defined server authentication function. Please consider [sshUserAuthenticateWithTypeFn](#).
- [sshUserAuthenticateFn](#) [SshGetUserAuthenticate](#) (void)
[DEPRECATED] Gets the user defined server authentication function. Please consider [SshGetUserAuthenticateWithType](#).
- void [SshSetUserAuthenticateWithType](#) ([sshUserAuthenticateWithTypeFn](#) sshUserAuthenticateFnPtr)
Sets the user defined server authentication function.
- [sshUserAuthenticateWithTypeFn](#) [SshGetUserAuthenticateWithType](#) (void)
Gets the user defined server authentication function..
- void [SshClientSetGetUserPasswordFn](#) ([sshGetUserPwFn](#) sshGetUserPwFnPtr)
Sets the user defined client authentication function for getting user passwords during SSH authentication.
- [sshGetUserPwFn](#) [SshClientGetUserPasswordFn](#) (void)
Gets the user defined client authentication function for getting a user password during authentication.
- void [SshClientSetGetUserKeyFn](#) ([sshGetUserKeyFn](#) sshGetUserKeyFnPtr)
Sets the user defined client authentication function for getting user keys during SSH authentication.
- [sshGetUserKeyFn](#) [SshClientGetUserKeyFn](#) (void)
Gets the user defined client authentication function for getting a user key during authentication.
- void [SshSetUserGetKey](#) ([sshUserGetKeyFn](#) sshUserGetKeyFnPtr)
Sets the user defined callback method to provide the server key.
- [sshUserGetKeyFn](#) [SshGetUserGetKey](#) (void)
Gets the user defined callback method to provide the server key.
- bool [SshValidateKey](#) (const char *candidateKey, int candidateKeySize, int *keyTypePtr, int keyFormat=WOLFSSH_FORMAT_ASN1)
Takes a key and returns if it's valid or not.
- bool [SshWritePublicKey](#) (int publicKeyFd, unsigned char *candidateKey, int candidateKeySize)
Write public key to file descriptor. Takes both PEM and ANS1 formats.
- int [NbSshInit](#) ()
Initializes the underlying SSH framework. This will start a background task used to handle negotiations and SSH traffic. This is automatically called by [SshAccept](#) and [SshConnect](#), and doesn't need to be called directly except in special circumstances.
- int [SshAccept](#) (int listenFd, [IPADDR](#) *clientAddress, uint16_t *securePort, uint16_t timeout)
Accepts and negotiates SSH session. Automatically calls [NbSshInit\(\)](#) if required.
- int [SshConnect](#) ([IPADDR](#) clientAddress, uint16_t securePort, uint16_t localPort, uint16_t timeout, const char *username)
Issues a connect request to negotiates an SSH session. Automatically calls [NbSshInit\(\)](#) if required.
- SshSocket * [SshNegotiateSession](#) (int fd)
Negotiates an SSH server session on an open file descriptor.
- SshSocket * [SshNegotiateSessionClient](#) (int secureFd, const char *username)
Negotiates an SSH client session on an open file descriptor.
- void [SshPrintStatistics](#) (int secureFd)
Negotiates an SSH client session on an open file descriptor.
- int [SshGetKeySize](#) ()
Determines and returns SSH's installed key size.
- int [SshSetBannerText](#) (const char *banner)
Sets the banner text displayed by the SSH server on connection.
- int [SshSetSockOption](#) (int fd, int option)
Set SSH TCP socket options.
- int [SshClrSockOption](#) (int fd, int option)
Clear SSH TCP socket options.
- int [SshGetSockOption](#) (int fd)
Returns the options for the specified SSH TCP socket.

14.44.1 Detailed Description

The NetBurner SSH Library

14.44.2 Typedef Documentation

14.44.2.1 sshGetUserKeyFn

```
typedef int(* sshGetUserKeyFn) (const NBString &usernamePtr, NBString &publicKey, NBString
&privateKey, NBString &keyType)
#include <NbWolfSsh.h>
```

User provided SSH user key authenticate routine for clients.

Parameters

	<i>usernamePtr</i>	Username in plain text
out	<i>publicKey</i>	Used to return the corresponding public key for the user
out	<i>privateKey</i>	Used to return the corresponding private key for the user
out	<i>keyType</i>	Used to return the corresponding key type (ECC or RSA) for the user

Return values

>0	Password available for the user
<=0	Authentication failed

14.44.2.2 sshGetUserPwFn

```
typedef int(* sshGetUserPwFn) (const NBString &usernamePtr, NBString &passwordPtr)
#include <NbWolfSsh.h>
```

User provided SSH user password authentication routine for clients.

Parameters

	<i>usernamePtr</i>	Username in plain text
out	<i>passwordPtr</i>	Used to return the corresponding password for the user

Return values

>0	Password available for the user
<=0	Authentication failed

14.44.2.3 sshUserAuthenticateFn

```
typedef int(* sshUserAuthenticateFn) (const char *usernamePtr, const char *passwordPtr)
#include <NbWolfSsh.h>
```

[DEPRECATED] User provided SSH username and password authenticate routine for a server. Please consider `sshUserAuthenticateWithTypeFn`.

Parameters

<i>usernamePtr</i>	Username in plain text
<i>passwordPtr</i>	Password in plain text

Return values

1	Authentication passed
!1	Authentication failed

14.44.2.4 sshUserAuthenticateWithTypeFn

```
typedef int(* sshUserAuthenticateWithTypeFn) (const char *usernamePtr, const char *authValPtr,  
AuthType authType)
```

```
#include <NbWolfSsh.h>
```

User provided SSH user authenticate routine for a server.

Parameters

<i>usernamePtr</i>	Username in plain text
<i>authValPtr</i>	The value that is being passed in for authentication
<i>authType</i>	Either a password or a key

Return values

1	Authentication passed
!1	Authentication failed

14.44.2.5 sshUserGetKeyFn

```
typedef int(* sshUserGetKeyFn) (int keyRequested, const unsigned char **keyBufferPtr, int *key↵  
LengthPtr)
```

```
#include <NbWolfSsh.h>
```

The user defined callback to get the server key used during the initial SSH negotiation.

Parameters

<i>keyRequested</i>	Type key requested (ECC or RSA)
<i>keyBufferPtr</i>	Buffer containing the key (ASN1 or PEM format supported)
<i>keyLengthPtr</i>	Size of the key in 8 bit bytes

Return values

0	Key and length are valid
-1	Key requested is not available

See also

[SshSetUserGetKey\(\)](#)

[SshGetUserGetKey\(\)](#)

14.44.3 Function Documentation**14.44.3.1 NbSshInit()**

```
int NbSshInit ()
```

```
#include <NbWolfSsh.h>
```

Initializes the underlying SSH framework. This will start a background task used to handle negotiations and SSH traffic. This is automatically called by SshAccept and SshConnect, and doesn't need to be called directly except in special circumstances.

Return values

<i>SSH_SUCCESS</i>	if SSH system has been correctly initialized.
<i>Other</i>	SSH error code if initialization was unsuccessful.

See also

[SshAccept\(\)](#)[SshConnect\(\)](#)**14.44.3.2 SshAccept()**

```
int SshAccept (
    int listenFd,
    IPADDR * clientAddress,
    uint16_t * securePort,
    uint16_t timeout)
```

```
#include <NbWolfSsh.h>
```

Accepts and negotiates SSH session. Automatically calls [NbSshInit\(\)](#) if required.

Parameters

<i>listenFd</i>	File descriptor of listening socket
<i>clientAddress</i>	Address of client
<i>securePort</i>	Secure port of negotiated socket
<i>timeout</i>	Ticks to wait for connection, 0 is infinite

Return values

<i>>0</i>	The secure file descriptor if successful.
<i>Other</i>	A TCP or SSH error code, depending on the error..

See also

[NbSshInit\(\)](#)[SshConnect\(\)](#)**14.44.3.3 SshClientGetUserKeyFn()**

```
sshGetUserKeyFn SshClientGetUserKeyFn (
    void )
```

```
#include <NbWolfSsh.h>
```

Gets the user defined client authentication function for getting a user key during authentication.

Return values

<i>sshGetUserPwFnPtr</i>	Sets the user defined server authentication function.

See also

[SshClientSetGetUserPaswordFn\(\)](#)[SshClientGetUserPaswordFn\(\)](#)[SshClientSetGetUserKeyFn\(\)](#)

14.44.3.4 SshClientGetUserPaswordFn()

```
sshGetUserPwFn SshClientGetUserPaswordFn (
    void )
```

```
#include <NbWolfSsh.h>
```

Gets the user defined client authentication function for getting a user password during authentication.

Return values

A	pointer to the fuction that is currently set for passing in a user's password during authentication.

See also

[SshClientSetGetUserPaswordFn\(\)](#)

[SshClientSetGetUserKeyFn\(\)](#)

[SshClientGetUserKeyFn\(\)](#)

14.44.3.5 SshClientSetGetUserKeyFn()

```
void SshClientSetGetUserKeyFn (
    sshGetUserKeyFn sshGetUserKeyFnPtr)
```

```
#include <NbWolfSsh.h>
```

Sets the user defined client authentication function for getting user keys during SSH authentication.

Parameters

<i>sshGetUserKeyFnPtr</i>	Sets the user defined server authentication function.

See also

[SshClientSetGetUserPaswordFn\(\)](#)

[SshClientGetUserPaswordFn\(\)](#)

[SshClientGetUserKeyFn\(\)](#)

14.44.3.6 SshClientSetGetUserPaswordFn()

```
void SshClientSetGetUserPaswordFn (
    sshGetUserPwFn sshGetUserPwFnPtr)
```

```
#include <NbWolfSsh.h>
```

Sets the user defined client authentication function for getting user passwords during SSH authentication.

Parameters

<i>sshGetUserPwFnPtr</i>	Sets the user defined client authentication function.

See also

[SshClientGetUserPaswordFn\(\)](#)

[SshClientSetGetUserKeyFn\(\)](#)

[SshClientGetUserKeyFn\(\)](#)

14.44.3.7 SshClrSockOption()

```
int SshClrSockOption (
    int fd,
    int option)
```

```
#include <NbWolfSsh.h>
```

Clear SSH TCP socket options.

Parameters

<i>fd</i>	Socket file descriptor.
<i>option</i>	Socket option to clear: TCP Socket Options .

Returns

A bitmask of the options for the specified socket

See also

[SshSetSockOption\(\)](#)

[SshGetSockOption\(\)](#)

14.44.3.8 SshConnect()

```
int SshConnect (
    IPADDR clientAddress,
    uint16_t securePort,
    uint16_t localPort,
    uint16_t timeout,
    const char * username)
```

```
#include <NbWolfSsh.h>
```

Issues a connect request to negotiates an SSH session. Automatically calls [NbSshInit\(\)](#) if required.

Parameters

<i>clientAddress</i>	Address of client
<i>securePort</i>	Secure port of negotiated socket
<i>localPort</i>	Optional parameter to specify a Local port number. Recommend this always be set to a value of 0 so a random local port number is used.
<i>timeout</i>	Ticks to wait for connection, 0 is infinite
<i>username</i>	Username to use in negotiation process of secure connection

Return values

>0	The secure file descriptor if successful.
0	Request timed out.
<0	A TCP or SSH error code, depending on the error..

See also

[SshAccept\(\)](#)

[NbSshInit\(\)](#)

14.44.3.9 SshGetKeySize()

```
int SshGetKeySize ()
#include <NbWolfSsh.h>
```

Determines and returns SSH's installed key size.

Returns

The installed key's size

14.44.3.10 SshGetSockOption()

```
int SshGetSockOption (
    int fd)
#include <NbWolfSsh.h>
```

Returns the options for the specified SSH TCP socket.

Parameters

<i>fd</i>	Socket file descriptor.

Returns

A bitmask of the options for the specified socket

See also

[SshSetSockOption\(\)](#)
[SshClrSockOption\(\)](#)

14.44.3.11 SshGetUserAuthenticate()

```
sshUserAuthenticateFn SshGetUserAuthenticate (
    void )
#include <NbWolfSsh.h>
```

[DEPRECATED] Gets the user defined server authentication function. Please consider [SshGetUserAuthenticateWithType](#).

Return values

<i>Returns</i>	a pointer to the function that is currently set for server user authentication.

See also

[SshGetUserAuthenticateWithType](#)

14.44.3.12 SshGetUserAuthenticateWithType()

```
sshUserAuthenticateWithTypeFn SshGetUserAuthenticateWithType (
    void )
#include <NbWolfSsh.h>
```

Gets the user defined server authentication function..

Return values

<i>A</i>	pointer to the function that is currently set for server user authentication.

See also

[SshGetUserAuthenticateWithType\(\)](#)

14.44.3.13 SshGetUserGetKey()

```
sshUserGetKeyFn SshGetUserGetKey (
    void )
#include <NbWolfSsh.h>
```

Gets the user defined callback method to provide the server key.

Return values

<code>sshUserGetKeyFn</code>	The user defined callback

See also

[sshUserGetKeyFn\(\)](#)

[SshSetUserGetKey\(\)](#)

14.44.3.14 SshNegotiateSession()

```
SshSocket * SshNegotiateSession (
    int fd)
```

```
#include <NbWolfSsh.h>
```

Negotiates an SSH server session on an open file descriptor.

Parameters

<code>fd</code>	File descriptor to use in the SSN negotiation, usually obtained with a call to accept()

Returns

A pointer to the SSH socket created on a successfull connection, or a nullptr if not successful.

14.44.3.15 SshNegotiateSessionClient()

```
SshSocket * SshNegotiateSessionClient (
    int secureFd,
    const char * username)
```

```
#include <NbWolfSsh.h>
```

Negotiates an SSH client session on an open file descriptor.

Parameters

<code>secureFd</code>	File descriptor to use in the SSN negotiation
<code>username</code>	Username to use in negotiation process of secure connection

Returns

A pointer to the SSH socket created on a successfull connection, or a nullptr if not successful.

14.44.3.16 SshPrintStatistics()

```
void SshPrintStatistics (
    int secureFd)
```

```
#include <NbWolfSsh.h>
```

Negotiates an SSH client session on an open file descriptor.

Parameters

<code>secureFd</code>	Secure file descriptor

14.44.3.17 SshSetBannerText()

```
int SshSetBannerText (
    const char * banner)
```

```
#include <NbWolfSsh.h>
```

Sets the banner text displayed by the SSH server on connection.

Parameters

<i>banner</i>	The text to display to any connected clients

Return values

<i>SSH_SUCCESS</i>	If successful
<i><0</i>	Error code on failure

14.44.3.18 SshSetSockOption()

```
int SshSetSockOption (
    int fd,
    int option)
#include <NbWolfSsh.h>
Set SSH TCP socket options.
```

Parameters

<i>fd</i>	Socket file descriptor.
<i>option</i>	Socket option to set: TCP Socket Options .

Returns

A bitmask of the options for the specified socket

See also

[SshClrSockOption\(\)](#)

[SshGetSockOption\(\)](#)

14.44.3.19 SshSetUserAuthenticate()

```
void SshSetUserAuthenticate (
    sshUserAuthenticateFn sshUserAuthenticateFnPtr)
#include <NbWolfSsh.h>
[DEPRECATED] Sets the user defined server authentication function. Please consider sshUserAuthenticateWith↵
TypeFn.
```

Parameters

<i>sshUserAuthenticateFnPtr</i>	Fuction to set

See also

[SshSetUserAuthenticateWithType\(\)](#)

14.44.3.20 SshSetUserAuthenticateWithType()

```
void SshSetUserAuthenticateWithType (
    sshUserAuthenticateWithTypeFn sshUserAuthenticateFnPtr)
#include <NbWolfSsh.h>
Sets the user defined server authentication function.
```

Parameters

<code>sshUserAuthenticateFnPtr</code>	Fuction to set

See also

[SshSetUserAuthenticateWithType\(\)](#)

14.44.3.21 SshSetUserGetKey()

```
void SshSetUserGetKey (
    sshUserGetKeyFn sshUserGetKeyFnPtr)
#include <NbWolfSsh.h>
```

Sets the user defined callback method to provide the server key.

Parameters

<code>sshUserGetKeyFnPtr</code>	The user defined callback

See also

[sshUserGetKeyFn\(\)](#)

[SshGetUserGetKey\(\)](#)

14.44.3.22 SshValidateKey()

```
bool SshValidateKey (
    const char * candidateKey,
    int candidateKeySize,
    int * keyTypePtr,
    int keyFormat = WOLFSSH_FORMAT_ASN1)
#include <NbWolfSsh.h>
```

Takes a key and returns if it's valid or not.

Parameters

	<code>candidateKey</code>	A pointer to a buffer containing the key
	<code>candidateKeySize</code>	The size of the key in bytes
out	<code>keyTypePtr</code>	Is set to what kind of key is passed it (ECC and RSA are valid)
	<code>keyFormat</code>	The key format (ASN1 or PEM)

Return values

<code>true</code>	The key is valid
<code>false</code>	The key is not valid

14.44.3.23 SshWritePublicKey()

```
bool SshWritePublicKey (
    int publicKeyFd,
    unsigned char * candidateKey,
    int candidateKeySize)
#include <NbWolfSsh.h>
```

Write public key to file descriptor. Takes both PEM and ANS1 formats.

Parameters

<i>publicKeyFd</i>	Open target file descriptor
<i>candidateKey</i>	key in buffer
<i>candidateKeySize</i>	Size of key in bytes

Return values

<i>true</i>	A valid key was written to the file descriptor
<i>false</i>	An error occurred

14.44.4 SSH Error Codes

Macros

- **#define SSH_SUCCESS (0)**
The function completed successfully.
- **#define SSH_ERROR_FAILED_SESSION_FAILED (-300)**
Not currently used, kept for backward compatibility.
- **#define SSH_ERROR_FAILED_NEGOTIATION (-301)**
The connection failed the SSH negotiation.
- **#define SSH_ERROR_FAILED_INITIALIZATION (-302)**
The SSH system failed to initialize.
- **#define SSH_ERROR_FAILED_CONTEXT_INIT (-303)**
Unable to instantiate SSH client/server context.
- **#define SSH_ERROR_BAD_KEY (-304)**
The key provided was invalid.
- **#define SSH_ERROR_BAD_ARGUMENT (-305)**
A bad argument was provided to the function.
- **#define SSH_FAILED_KEY_CHECK (-306)**
There were no keys designated to use for the SSH library, .

14.44.4.1 Detailed Description

Error codes that can be returned by the NetBurner SSH functions.

14.45 SSL/TLS

The NetBurner SSL/TLS Library. See [SSL/TLS and Certificates](#) and examples in the examples/SSL folder.

Topics

- [ACME Certificate Generation](#)
Request valid public certificates from an ACME certificate authority for devices with a public IP and DNS.
- [Onboard Self-Signed Certificate Generation](#)
See [On-board Cert Generation - Simple](#) and other examples in the examples/SSL folder.
- [SSL Error Codes](#)

Enumerations

- enum class [ExpireRetCodes](#) : unsigned char { [ExpireRetCodes::E_CERT_VALID](#) = 0 , [ExpireRetCodes::E_CERT_EXPIRED](#) = 1 , [ExpireRetCodes::E_BAD_EXPIRATION_VALUE](#) = 2 , [ExpireRetCodes::E_NO_SYSTEM_TIME](#) = 3 }
- Return values for [SSL_IsCertExpired\(\)](#).*

Functions

- int **SSL_connect** (IPADDR ip, uint16_t remotePort, uint32_t timeout, const char *commonName, bool verifyPeer=false, bool waitForNegotiation=true, const unsigned char *certBuff=nullptr, uint32_t certBuffSize=0, uint16_t certBuffFormat=SSL_FILETYPE_PEM, int intf=-1)
SSL mirror of the TCP::connect() call.
- int **SSL_connect** (IPADDR ip, uint16_t localPort, uint16_t remotePort, uint32_t timeout, const char *commonName, bool verifyPeer=false, bool waitForNegotiation=true, const unsigned char *certBuff=nullptr, uint32_t certBuffSize=0, uint16_t certBuffFormat=SSL_FILETYPE_PEM, int intf=-1)
Old version of SSL_Connect with local port.
- int **SSL_makeSecure** (int tcpFd, const char *commonName, bool verifyPeer=false, bool waitForNegotiation=true, const unsigned char *certBuff=nullptr, uint32_t certBuffSize=0, uint16_t certBuffFormat=SSL_FILETYPE_PEM, bool blsClient=true, uint32_t timeout=WAIT_FOREVER)
SSL mirror of the TCP::connect() call, but uses an existing TCP file descriptor.
- int **SSL_negotiate** (int tcpFd, const char *common_name, bool reuseSessionPort=true, bool verifyPeer=false, const unsigned char *certBuff=nullptr, uint32_t certBuffSize=0, uint16_t certBuffFormat=SSL_FILETYPE_PEM, bool blsClient=true)
Executes the TLS negotiation over a given socket, to establish a secure communication channel through that socket.
- int **SSLReadWithTimeout** (int fd, char *cp, int size, uint32_t timeout)
SSL mirror of the ReadWithTiemout() function.
- uint32_t **SetMaxSessionReuseTime** (uint32_t newLifetimeTicks)
Allows one to specify at runtime what the longest duration that an established session can be used for before new connections must be fully renegotiated.
- bool **IsSSL_CertNKeyValid** (const char *keyPEM, ssize_t keylen, const char *certPEM, ssize_t certlen, uint16_t certBuffFormat=SSL_FILETYPE_PEM)
Validate candidate certificate and private key.
- bool **SSL_AddCertToClientCaList** (const unsigned char *certBuff=nullptr, uint32_t certBuffSize=0, uint16_t certBuffFormat=SSL_FILETYPE_PEM)
Attempts to add a cert to the Certificate Authority List to the client SSL context.
- bool **SSL_AddCertToServerCaList** (const unsigned char *certBuff=nullptr, uint32_t certBuffSize=0, uint16_t certBuffFormat=SSL_FILETYPE_PEM)
Attempts to add a cert to the Certificate Authority List to the client SSL context.
- bool **SSL_ClearClientCaList** ()
Clears certs that have been added to the client's Certificate Authority list.
- bool **SSL_ClearServerCaList** ()
Clears certs that have been added to the server's Certificate Authority list.
- uint16_t **SSLGetRxBufferSpaceUsed** (int fd)
Returns the amount of data buffered to be read from the socket.
- uint16_t **SSLGetTxBufferAvailSpace** (int fd)
Returns the amount of data buffered to be read from the socket.
- void **SslInit** ()
This function initializes the SSL library.
- void **SSL_DumpSSLSockets** ()
Prints the current state of all used sockets to stdio.
- void **SSL_ClientReadyCert** (const unsigned char *cert, const unsigned char *key, uint16_t certBuffFormat=SSL_FILETYPE_PEM)
Add a key and certificate to the client.
- void **SSL_ServerReadyCert** (const unsigned char *cert, const unsigned char *key, uint16_t certBuffFormat=SSL_FILETYPE_PEM)
Add a key and certificate to the server.
- **ExpireRetCodes** **SSL_IsCertExpired** (unsigned char *certBuff, int certLen, uint16_t certBuffFormat=SSL_FILETYPE_PEM)
Returns whether the given certificate has expired or not.

- bool [SSL_GetIssuerFromCert](#) (unsigned char *certBuff, int certLen, unsigned char *issBuf, int issBufLen, uint16_t certBuffFormat=SSL_FILETYPE_PEM)
Returns the issuer from a certificate blob in the PEM format.
- bool [SSL_GetSubjectNameFromCert](#) (unsigned char *certBuff, int certLen, unsigned char *subNamBuf, int subNamBufLen, uint16_t certBuffFormat=SSL_FILETYPE_PEM)
Returns the subject name from a certificate blob in the PEM format.
- bool [SSL_GetAltNameFromCert](#) (unsigned char *certBuff, int certLen, unsigned char *altNamBuf, int altNamBufLen, uint16_t certBuffFormat=SSL_FILETYPE_PEM)
Returns an alternate name from a certificate blob in the PEM format.
- bool [SSL_GetExpirationDate](#) (unsigned char *certBuff, int certLen, unsigned char *expBuf, int expBufLen, uint16_t certBuffFormat=SSL_FILETYPE_PEM)
Returns the expiration data of the certificate in question.
- bool [SSL_GetTimeToExpire](#) (unsigned char *certBuff, int certLen, uint16_t certBuffFormat, time_t &timeToExp)
Returns whether the given certificate has expired or not.
- bool [SSL_GetCnFromCert](#) (unsigned char *certBuff, int certLen, unsigned char *comNamBuf, int comNamBufLen, uint16_t certBuffFormat=SSL_FILETYPE_PEM)
Returns the common name from a certificate blob in the PEM format.
- const char * [GetCertificatePEM](#) ()
Function that returns a pointer to the compiled in server certificate.
- int [GetCertificateLen](#) ()
Function that returns the length of the compiled in server certificate.
- const char * [GetPrivateKeyPEM](#) ()
Function that returns a pointer to the compiled in server key.
- int [GetPrivateKeyLen](#) ()
Function that the length of the compiled in server key.
- void [SslSetVerifyPeerClient](#) (bool verifyPeer)
Sets whether or not verify peer should be enabled on the global SSL client context.
- void [SslSetVerifyPeerServer](#) (bool verifyPeer)
Sets whether or not verify peer should be enabled on the global SSL server context.
- bool [SSL_IsCertAddedToClientCaList](#) ()
Checks to see if any certificates have been added to the global SSL client context.
- bool [SSL_IsCertAddedToServerCaList](#) ()
Checks to see if any certificates have been added to the global SSL server context.

SSL Server Functions

- BOOL(* [IsSSLfd](#))(int fd)
Returns whether or not a file descriptor is an SSL file descriptor.
- int(* [SSL_GetRawSocket](#))(int sslSocketFd)
Returns the TCP file descriptor associated with an SSL socket file descriptor.
- int(* [SSL_setsockopt](#))(int fd, int option)
Set SSL TCP socket options.
- int(* [SSL_clrsockopt](#))(int fd, int option)
Clear SSL TCP socket options.
- int(* [SSL_getsockopt](#))(int fd)
Returns the options for the specified SSL TCP socket.
- int [SSL_accept](#) (int fdListen, [IPADDR](#) *address, uint16_t *port, uint16_t timeout, bool waitForNegotiation=true, bool verifyPeer=false, const unsigned char *certBuff=nullptr, uint32_t certBuffSize=0, uint16_t certBuffFormat=SSL_FILETYPE_PEM, const char *commonName=nullptr)
This call is a mirror of the TCP::accept() call.
- [IPADDR](#) [SSL_GetSocketRemoteAddr](#) (int fd)

- Returns the remote address of this connected socket.*
- uint16_t [SSL_GetSocketRemotePort](#) (int fd)
Returns the remote port of this connected socket.
- IPADDR [SSL_GetSocketLocalAddr](#) (int fd)
Returns the local address of this connected socket.
- uint16_t [SSL_GetSocketLocalPort](#) (int fd)
Returns the local port of this connected socket.
- int [SSL_GetSocketInterface](#) (int fd)
Return the network interface associated with an SSL TCP socket.
- uint16_t [SSL_GetTcpRtxCount](#) (int fd)
Returns the number of re-transmits that have occurred on the specified SSL connection.
- uint32_t [SSL_TcpGetLastRxTime](#) (int fd)
SSL mirror of [TcpGetLastRxTime\(\)](#).
- uint32_t [SSL_TcpGetLastRxInterval](#) (int fd)
SSL mirror of [TcpGetLastRxInterval\(\)](#).
- void [SSL_TcpSendKeepAlive](#) (int fd)
SSL mirror of [TcpSendKeepAlive\(\)](#).
- bool [SSLSocketsAvail](#) ()
Returns whether or not there are any sockets available.
- int [FreeSSLSockets](#) ()
Returns the number of currently free SSL sockets.

SSL socket option functions

- int [SSL_getsocketerror](#) (int fd)
Returns the socket error state for the specified SSL socket.

14.45.1 Detailed Description

The NetBurner SSL/TLS Library. See [SSL/TLS and Certificates](#) and examples in the examples/SSL folder.
`#include< ssl.h>`

14.45.2 Enumeration Type Documentation

14.45.2.1 ExpireRetCodes

```
enum class ExpireRetCodes : unsigned char [strong]
#include <ssl.h>
```

Return values for [SSL_IsCertExpired\(\)](#).

Enumerator

E_CERT_VALID	Certificate not expired.
E_CERT_EXPIRED	Certificate expired.
E_BAD_EXPIRATION_VALUE	Unable to read the expiration information from the certificate.
E_NO_SYSTEM_TIME	The system time has not been set.

14.45.3 Function Documentation

14.45.3.1 FreeSSLSockets()

```
int FreeSSLSockets ()  
#include <ssl.h>
```

Returns the number of currently free SSL sockets.

Returns

The number of free SSL sockets.

14.45.3.2 GetCertificateLen()

```
int GetCertificateLen ()  
#include <ssl.h>
```

Function that returns the length of the compiled in server certificate.

See the SSL documentation and examples for information on generating a self signed certificate and key. This function can be overridden to provide the ability to install uploaded certificates.

Returns

The length of the compiled in certificate.

See also

[GetCertificatePEM\(\)](#)

[GetPrivateKeyLen\(\)](#)

14.45.3.3 GetCertificatePEM()

```
const char * GetCertificatePEM ()  
#include <ssl.h>
```

Function that returns a pointer to the compiled in server certificate.

See the SSL documentation and examples for information on generating a self signed certificate and key.

Returns

A pointer to the compiled in certificate.

See also

[GetPrivateKeyPEM\(\)](#)

[GetCertificateLen\(\)](#)

14.45.3.4 GetPrivateKeyLen()

```
int GetPrivateKeyLen ()  
#include <ssl.h>
```

Function that the length of the compiled in server key.

See the SSL documentation and examples for information on generating a self signed certificate and key. This function can be overridden to provide the ability to install uploaded keys.

Returns

The length of the compiled in key.

See also

[GetPrivateKeyPEM\(\)](#)

[GetCertificateLen\(\)](#)

14.45.3.5 GetPrivateKeyPEM()

```
const char * GetPrivateKeyPEM ()
#include <ssl.h>
```

Function that returns a pointer to the compiled in server key.

See the SSL documentation and examples for information on generating a self signed certificate and key.

Returns

A pointer to the compiled in server key.

See also

[GetCertificatePEM\(\)](#)

[GetPrivateKeyLen\(\)](#)

14.45.3.6 IsSSL_CertNKeyValid()

```
bool IsSSL_CertNKeyValid (
    const char * keyPEM,
    ssize_t keylen,
    const char * certPEM,
    ssize_t certlen,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)
#include <ssl.h>
```

Validate candidate certificate and private key.

Determines if the passed in certificate and private key are valid or not.

Parameters

<i>keyPEM</i>	A buffer containing the private key
<i>keylen</i>	The length of the buffer containing the private key
<i>certPEM</i>	A buffer containing the certificate
<i>certlen</i>	The length of the buffer containing the certificate
<i>certBuffFormat</i>	The format of the cert being passed in. Either SSL_FILETYPE_PEM or WOLFSSL_FILETYPE_ASN1.

Return values

<code><code>true</code></code>	Certificate and key passed in are valid
<code><code>false</code></code>	Certificate and/or key passed in are not valid

14.45.3.7 SetMaxSessionReuseTime()

```
uint32_t SetMaxSessionReuseTime (
    uint32_t newLifetimeTicks)
#include <ssl.h>
```

Allows one to specify at runtime what the longest duration that an established session can be used for before new connections must be fully renegotiated.

This is a security trade-off feature. It will negatively impact performance to decrease this value. To disable maximum reuse lifetime, pass the argument as 0. This will force a new negotiation on every connection. Calling this function will not affect sessions that are currently stored until they have been used to establish a new connection.

The default value used for this purpose on startup is defined in [predef.h](#) as the macro `SSL_DEFAULT_MAX_SESSION_AGE_TICKS` and is set to one hour.

Parameters

<i>newLifetimeTicks</i>	How long (in ticks) to allow an established session to be used.

Returns

Returns previously assigned reuse maximum.

14.45.3.8 SSL_accept()

```
int SSL_accept (
    int fdListen,
    IPADDR * address,
    uint16_t * port,
    uint16_t timeout,
    bool waitForNegotiation = true,
    bool verifyPeer = false,
    const unsigned char * certBuff = nullptr,
    uint32_t certBuffSize = 0,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM,
    const char * commonName = nullptr)
```

#include <ssl.h>

This call is a mirror of the TCP::accept() call.

Parameters

<i>fdListen</i>	File descriptor of the TCP listening socket.
<i>*address</i>	IPADDR variable to hold the address of the connecting computer.
<i>*port</i>	Variable to receive the remote port of this connection.
<i>timeout</i>	Number of system time ticks to wait for a connection (regular TCP::accept()).
<i>waitForNegotiation</i>	Wait for negotiation to complete. Default is true.
<i>verifyPeer</i>	Default is false. Set to true to verify the client certificate on connect.
<i>certBuff</i>	Buffer containing the certificate data. Default is null.
<i>certBuffSize</i>	Size of the buffer containing the certificate data. Default is 0.
<i>certBuffFormat</i>	Format of the certificate buffer. Default is SSL_FILETYPE_PEM.
<i>commonName</i>	Common name to use for checking certificate validity. Default is null, which will accept any connection.

Return values

>0	The file descriptor of the connected SSL socket
<0	The error code for the failure from SSL Error Codes or TCP Socket Status

See also

[SSL_connect\(\)](#)
[SSL_GetSocketRemoteAddr\(\)](#)
[SSL_GetSocketRemotePort\(\)](#)
[SSL_GetSocketLocalAddr\(\)](#)
[SSL_GetSocketLocalPort\(\)](#)
[IsSSLfd\(\)](#)
[SSL_setsockopt\(\)](#)
[SSL_clrsockopt\(\)](#)
[SSL_getsockopt\(\)](#)

14.45.3.9 SSL_AddCertToClientCaList()

```
bool SSL_AddCertToClientCaList (
    const unsigned char * certBuff = nullptr,
    uint32_t certBuffSize = 0,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)
```

```
#include <ssl.h>
```

Attempts to add a cert to the Certificate Authority List to the client SSL context.

Parameters

<i>certBuff</i>	A buffer containing the certificate to add.
<i>certBuffSize</i>	The size of the cert being passed in.
<i>certBuffFormat</i>	The format of the cert being passed in. Either SSL_FILETYPE_PEM or WOLFSSL_FILETYPE_ASN1.

Return values

<code><code>true</code></code>	Successfully add the certificate to the CA List
<code><code>false</code></code>	Was not able to add the certificate to the CA List

14.45.3.10 SSL_AddCertToServerCaList()

```
bool SSL_AddCertToServerCaList (
    const unsigned char * certBuff = nullptr,
    uint32_t certBuffSize = 0,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)
```

```
#include <ssl.h>
```

Attempts to add a cert to the Certificate Authority List to the client SSL context.

Parameters

<i>certBuff</i>	A buffer containing the certificate to add.
<i>certBuffSize</i>	The size of the cert being passed in.
<i>certBuffFormat</i>	The format of the cert being passed in. Either SSL_FILETYPE_PEM or WOLFSSL_FILETYPE_ASN1.

Return values

<code><code>true</code></code>	Successfully add the certificate to the CA List
<code><code>false</code></code>	Was not able to add the certificate to the CA List

14.45.3.11 SSL_ClearClientCaList()

```
bool SSL_ClearClientCaList ()
```

```
#include <ssl.h>
```

Clears certs that have been added to the client's Certificate Authority list.

Return values

<code><code>true</code></code>	Successfully cleared the Certificate Authority list
<code><code>false</code></code>	Failed to clear the Certificate Authority list

14.45.3.12 SSL_ClearServerCaList()

```
bool SSL_ClearServerCaList ()
#include <ssl.h>
```

Clears certs that have been added to the server's Certificate Authority list.

Return values

<code><code>true</code></code>	Successfully cleared the Certificate Authority list
<code><code>false</code></code>	Failed to clear the Certificate Authority list

14.45.3.13 SSL_ClientReadyCert()

```
void SSL_ClientReadyCert (
    const unsigned char * cert,
    const unsigned char * key,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)
#include <ssl.h>
```

Add a key and certificate to the client.

Parameters

<i>cert</i>	Pointer to the buffer holding the PEM certificate data
<i>key</i>	Pointer to the buffer holding the PEM key data
<i>certBuffFormat</i>	The format of the cert being passed in. Either SSL_FILETYPE_PEM or WOLFSSL_FILETYPE_ASN1.

14.45.3.14 SSL_connect() [1/2]

```
int SSL_connect (
    IPADDR ip,
    uint16_t localPort,
    uint16_t remotePort,
    uint32_t timeout,
    const char * commonName,
    bool verifyPeer = false,
    bool waitForNegotiation = true,
    const unsigned char * certBuff = nullptr,
    uint32_t certBuffSize = 0,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM,
    int intf = -1)
#include <ssl.h>
```

Old version of SSL_Connect with local port.

Parameters

<i>ip</i>	The address to connect to.
<i>localPort</i>	The local port to use. 0 will pick one
<i>remotePort</i>	The remote port to connect to.
<i>timeout</i>	The number of ticks to wait for a connection.
<i>*commonName</i>	The common name to use for checking certificate validity passing in NULL will accept any connection.
<i>verifyPeer</i>	Should the client verify the server's certificate against a Certificate Authority list?
<i>waitForNegotiation</i>	Should the function wait until the negotiation is complete before returning?
<i>certBuff</i>	Buffer containing the certificate data

Parameters

<i>certBuffSize</i>	Size of the buffer containing the certificate data
<i>certBuffFormat</i>	The format of the certificate buffer
<i>intf</i>	Optional local interface number to use for connection

Return values

> 0	The file descriptor of the connected SSL socket
< 0	The error code for the failure from SSL Error Codes or TCP Socket Status

See also

[SSL_accept\(\)](#)
[SSL_GetSocketRemoteAddr\(\)](#)
[SSL_GetSocketRemotePort\(\)](#)
[SSL_GetSocketLocalAddr\(\)](#)
[SSL_GetSocketLocalPort\(\)](#)
[IsSSLfd\(\)](#)
[SSL_setsockopt\(\)](#)
[SSL_clrsockopt\(\)](#)
[SSL_getsockopt\(\)](#)

14.45.3.15 SSL_connect() [2/2]

```

int SSL_connect (
    IPADDR ip,
    uint16_t remotePort,
    uint32_t timeout,
    const char * commonName,
    bool verifyPeer = false,
    bool waitForNegotiation = true,
    const unsigned char * certBuff = nullptr,
    uint32_t certBuffSize = 0,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM,
    int intf = -1)

```

```
#include <ssl.h>
```

SSL mirror of the TCP::connect() call.

Parameters

<i>ip</i>	The address to connect to.
<i>remotePort</i>	The remote port to connect to.
<i>timeout</i>	The number of ticks to wait for a connection.
<i>*commonName</i>	The common name to use for checking certificate validity passing in NULL will accept any connection.
<i>verifyPeer</i>	Should the client verify the server's certificate against a Certificate Authority list?
<i>waitForNegotiation</i>	Should the function wait until the negotiation is complete before returning?
<i>certBuff</i>	Buffer containing the certificate data
<i>certBuffSize</i>	Size of the buffer containing the certificate data
<i>certBuffFormat</i>	The format of the certificate buffer
<i>intf</i>	Optional local interface number to use for connection

Return values

>0	The file descriptor of the connected SSL socket
<0	The error code for the failure from SSL Error Codes or TCP Socket Status

See also

[SSL_accept\(\)](#)
[SSL_GetSocketRemoteAddr\(\)](#)
[SSL_GetSocketRemotePort\(\)](#)
[SSL_GetSocketLocalAddr\(\)](#)
[SSL_GetSocketLocalPort\(\)](#)
[IsSSLfd\(\)](#)
[SSL_setsockopt\(\)](#)
[SSL_clrsockopt\(\)](#)
[SSL_getsockopt\(\)](#)

14.45.3.16 SSL_GetAltNameFromCert()

```

bool SSL_GetAltNameFromCert (
    unsigned char * certBuff,
    int certLen,
    unsigned char * altNamBuf,
    int altNamBufLen,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)

```

```
#include <ssl.h>
```

Returns an alternate name from a certificate blob in the PEM format.

Parameters

<i>certBuff</i>	The buffer containing the certificate data
<i>certLen</i>	The length of the certificate
<i>altNamBuf</i>	The buffer to store the alternate name in
<i>altNamBufLen</i>	The length of the buffer where the alternate name will be placed
<i>certBuffFormat</i>	The format of the cert being passed in. Either <code>SSL_FILETYPE_PEM</code> or <code>WOLFSSL_FILETYPE_ASN1</code> .

Return values

<code><code>true</code></code>	If successfully retrieved the alternate name from the cert
<code><code>false</code></code>	If unable to retrieved the alternate name from the cert

14.45.3.17 SSL_GetCnFromCert()

```

bool SSL_GetCnFromCert (
    unsigned char * certBuff,
    int certLen,
    unsigned char * comNamBuf,
    int comNamBufLen,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)

```

```
#include <ssl.h>
```

Returns the common name from a certificate blob in the PEM format.

Parameters

<i>certBuff</i>	The buffer containing the certificate data
<i>certLen</i>	The length of the certificate
<i>comNamBuf</i>	The buffer to store the common name in
<i>comNamBufLen</i>	The length of the buffer where the common name will be placed
<i>certBuffFormat</i>	The format of the cert being passed in. Either SSL_FILETYPE_PEM or WOLFSSL_FILETYPE_ASN1.

Return values

<code><code>true</code></code>	If successfully retrieved the common name from the cert
<code><code>false</code></code>	If unable to retrieved the common name from the cert

14.45.3.18 SSL_GetExpirationDate()

```
bool SSL_GetExpirationDate (
    unsigned char * certBuff,
    int certLen,
    unsigned char * expBuf,
    int expBufLen,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)
```

```
#include <ssl.h>
```

Returns the expiration data of the certificate in question.

Parameters

<i>certBuff</i>	The buffer containing the certificate data
<i>certLen</i>	The length of the certificate
<i>expBuf</i>	The buffer to store the expiration date in
<i>expBufLen</i>	The length of the buffer where the expiration date will be placed
<i>certBuffFormat</i>	The format of the cert being passed in. Either SSL_FILETYPE_PEM or WOLFSSL_FILETYPE_ASN1.

Return values

<code><code>true</code></code>	If successfully retrieved the alternate name from the cert
<code><code>false</code></code>	If unable to retrieved the alternate name from the cert

14.45.3.19 SSL_GetIssuerFromCert()

```
bool SSL_GetIssuerFromCert (
    unsigned char * certBuff,
    int certLen,
    unsigned char * issBuf,
    int issBufLen,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)
```

```
#include <ssl.h>
```

Returns the issuer from a certificate blob in the PEM format.

Parameters

<i>certBuff</i>	The buffer containing the certificate data

Parameters

<i>certLen</i>	The length of the certificate
<i>issBuf</i>	The buffer to store the issuer in
<i>issBufLen</i>	The length of the buffer where the issuer will be placed
<i>certBuffFormat</i>	The format of the cert being passed in. Either SSL_FILETYPE_PEM or WOLFSSL_FILETYPE_ASN1.

Return values

<code><code>true</code></code>	If successfully retrieved the issuer from the cert
<code><code>false</code></code>	If unable to retrieved the issuer from the cert

14.45.3.20 SSL_getsocketerror()

```
int SSL_getsocketerror (
    int fd)
```

```
#include <ssl.h>
```

Returns the socket error state for the specified SSL socket.

Parameters

<i>fd</i>	Socket file descriptor.

Returns

The error state for the specified socket

14.45.3.21 SSL_GetSocketInterface()

```
int SSL_GetSocketInterface (
    int fd) [inline]
```

```
#include <ssl.h>
```

Return the network interface associated with an SSL TCP socket.

Parameters

<i>fd</i>	The socket file descriptor.

Returns

The network interface number (undefined for listening sockets)

See also

[TcpGetSocketInterface\(\)](#)

14.45.3.22 SSL_GetSocketLocalAddr()

```
IPADDR SSL_GetSocketLocalAddr (
    int fd) [inline]
```

```
#include <ssl.h>
```

Returns the local address of this connected socket.

This function is used to retrieve the local address of an SSL fd. This function will also work correctly if you pass in an fd that is a TCP connection. This allows you to use one set of code for both normal TCP and SSL connections.

Parameters

<i>fd</i>	The file descriptor to test.

Returns

The local IP address of the TCP or SSL connection, or 0 otherwise.

See also

[IsSSLfd\(\)](#)
[SSL_GetSocketRemotePort\(\)](#)
[SSL_GetSocketRemoteAddr\(\)](#)
[SSL_GetSocketLocalPort\(\)](#)

14.45.3.23 SSL_GetSocketLocalPort()

```
uint16_t SSL_GetSocketLocalPort (
    int fd) [inline]
#include <ssl.h>
```

Returns the local port of this connected socket.

This function is used to retrieve the local port of an SSL fd. This function will also work correctly if you pass in an fd that is a TCP connection. This allows you to use one set of code for both normal TCP and SSL connections.

Parameters

<i>fd</i>	The file descriptor to test.

Returns

The local port number of TCP or SSL connection, or 0 otherwise.

See also

[IsSSLfd\(\)](#)
[SSL_GetSocketRemoteAddr\(\)](#)
[SSL_GetSocketRemotePort\(\)](#)
[SSL_GetSocketLocalPort\(\)](#)

14.45.3.24 SSL_GetSocketRemoteAddr()

```
IPADDR SSL_GetSocketRemoteAddr (
    int fd) [inline]
#include <ssl.h>
```

Returns the remote address of this connected socket.

This function is used to retrieve the remote address of an SSL fd. This function will also work correctly if you pass in an fd that is a TCP connection. This allows you to use one set of code for both normal TCP and SSL connections.

Parameters

<i>fd</i>	The file descriptor to test.

Returns

IP address of the TCP or SSL connection, or 0 otherwise.

See also

[IsSSLfd\(\)](#)
[SSL_GetSocketRemotePort\(\)](#)
[SSL_GetSocketLocalAddr\(\)](#)
[SSL_GetSocketLocalPort\(\)](#)

14.45.3.25 SSL_GetSocketRemotePort()

```
uint16_t SSL_GetSocketRemotePort (
    int fd) [inline]
#include <ssl.h>
```

Returns the remote port of this connected socket.

This function is used to retrieve the remote port of an SSL fd. This function will also work correctly if you pass in an fd that is a TCP connection. This allows you to use one set of code for both normal TCP and SSL connections.

Parameters

<i>fd</i>	The file descriptor to test.

Returns

The remote port number of TCP or SSL connection, or 0 otherwise.

See also

[IsSSLfd\(\)](#)
[SSL_GetSocketRemoteAddr\(\)](#)
[SSL_GetSocketLocalAddr\(\)](#)
[SSL_GetSocketLocalPort\(\)](#)

14.45.3.26 SSL_GetSubjectNameFromCert()

```
bool SSL_GetSubjectNameFromCert (
    unsigned char * certBuff,
    int certLen,
    unsigned char * subNamBuf,
    int subNamBufLen,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)
#include <ssl.h>
```

Returns the subject name from a certificate blob in the PEM format.

Parameters

<i>certBuff</i>	The buffer containing the certificate data
<i>certLen</i>	The length of the certificate
<i>subNamBuf</i>	The buffer to store the subject name in
<i>subNamBufLen</i>	The length of the buffer where the subject name will be placed
<i>certBuffFormat</i>	The format of the cert being passed in. Either SSL_FILETYPE_PEM or WOLFSSL_FILETYPE_ASN1.

Return values

<code><code>true</code></code>	If successfully retrieved the subject name from the cert
<code><code>false</code></code>	If unable to retrieved the subject name from the cert

14.45.3.27 SSL_GetTcpRtxCount()

```
uint16_t SSL_GetTcpRtxCount (
    int fd) [inline]
```

```
#include <ssl.h>
```

Returns the number of re-transmits that have occurred on the specified SSL connection.

Parameters

<i>fd</i>	The socket file descriptor.

Returns

The number of system time ticks since the last packet was received on the connection.

See also

[GetTcpRtxCount\(\)](#);

14.45.3.28 SSL_GetTimeToExpire()

```
bool SSL_GetTimeToExpire (
    unsigned char * certBuff,
    int certLen,
    uint16_t certBuffFormat,
    time_t & timeToExp)
```

```
#include <ssl.h>
```

Returns whether the given certificate has expired or not.

Parameters

	<i>certBuff</i>	The buffer containing the certificate data
	<i>certLen</i>	The length of the certificate
	<i>certBuffFormat</i>	The format of the cert being passed in. Either SSL_FILETYPE_PEM or WOLFSSL_FILETYPE_ASN1.
out	<i>timeToExp</i>	The time the certificate is set to expire

Return values

<code><code>true</code></code>	If the certificate expiration was found
<code><code>false</code></code>	If the certificate expiration was not found

14.45.3.29 SSL_IsCertAddedToClientCaList()

```
bool SSL_IsCertAddedToClientCaList ()
#include <ssl.h>
```

Checks to see if any certificates have been added to the global SSL client context.

Certificate Authority certificates can be added to the client context during calls to [SSL_connect\(\)](#) or with calls to [SSL_AddCertToClientCaList\(\)](#).

Returns

true If a certificate has been added to the certificate authority list.

false If a certificate has not been added to the certificate authority list.

14.45.3.30 SSL_IsCertAddedToServerCaList()

```
bool SSL_IsCertAddedToServerCaList ()
#include <ssl.h>
```

Checks to see if any certificates have been added to the global SSL server context.

Certificate Authority certificates can be added to the server context during calls to [SSL_connect\(\)](#) or with calls to [SSL_AddCertToServerCaList\(\)](#).

Returns

true If a certificate has been added to the certificate authority list.

false If a certificate has not been added to the certificate authority list.

14.45.3.31 SSL_IsCertExpired()

```
ExpireRetCodes SSL_IsCertExpired (
    unsigned char * certBuff,
    int certLen,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)
#include <ssl.h>
```

Returns whether the given certificate has expired or not.

Parameters

<i>certBuff</i>	The buffer containing the certificate data
<i>certLen</i>	The length of the certificate
<i>certBuffFormat</i>	The format of the cert being passed in. Either SSL_FILETYPE_PEM or WOLFSSL_FILETYPE_ASN1.

Return values

<code><code>ExpireRetCodes::E_CERT_EXPIRED</code></code>	If the certificate has expired
<code><code>ExpireRetCodes::E_CERT_VALID</code></code>	If the certificate has not expired
<code><code>ExpireRetCodes::E_BAD_EXPIRATION_VALUE</code></code>	If unable to get the expiration data from the certificate
<code><code>ExpireRetCodes::E_NO_SYSTEM_TIME</code></code>	If the system time has not been set

14.45.3.32 SSL_makeSecure()

```
int SSL_makeSecure (
    int tcpFd,
    const char * commonName,
    bool verifyPeer = false,
    bool waitForNegotiation = true,
    const unsigned char * certBuff = nullptr,
    uint32_t certBuffSize = 0,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM,
    bool bIsClient = true,
    uint32_t timeout = WAIT_FOREVER)
#include <ssl.h>
```

SSL mirror of the TCP::connect() call, but uses an existing TCP file descriptor.

Parameters

<i>tcpFd</i>	The file descriptor for the socket to be secured.
<i>*commonName</i>	The common name to use for checking certificate validity passing in NULL will accept any connection.

Parameters

<i>verifyPeer</i>	Should the client verify the server's certificate against a Certificate Authority list?
<i>waitForNegotiation</i>	Should the function wait until the negotiation is complete before returning?
<i>certBuff</i>	Buffer containing the certificate data
<i>certBuffSize</i>	Size of the buffer containing the certificate data
<i>certBuffFormat</i>	The format of the certificate buffer
<i>bIsClient</i>	Perform the TLS negotiation as the Client and not the Server?
<i>timeout</i>	The number of timeticks to wait for completion

Return values

> 0	The file descriptor of the connected SSL socket
< 0	The error code for the failure from SSL Error Codes or TCP Socket Status

See also

[SSL_accept\(\)](#)
[SSL_GetSocketRemoteAddr\(\)](#)
[SSL_GetSocketRemotePort\(\)](#)
[SSL_GetSocketLocalAddr\(\)](#)
[SSL_GetSocketLocalPort\(\)](#)
[IsSSLfd\(\)](#)
[SSL_setsockopt\(\)](#)
[SSL_clrsockopt\(\)](#)
[SSL_getsockopt\(\)](#)

14.45.3.33 SSL_negotiate()

```

int SSL_negotiate (
    int tcpFd,
    const char * common_name,
    bool reuseSessionPort = true,
    bool verifyPeer = false,
    const unsigned char * certBuff = nullptr,
    uint32_t certBuffSize = 0,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM,
    bool bIsClient = true)

```

```
#include <ssl.h>
```

Executes the TLS negotiation over a given socket, to establish a secure communication channel through that socket.

Parameters

<i>tcpFd</i>	The TCP file descriptor to associate the connection with.
<i>*common_name</i>	The common name to use for checking certificate validity passing in NULL will accept any connection.
<i>reuseSessionPort</i>	Should we look at the port for reuse lookup?
<i>verifyPeer</i>	Should the client verify the server's certificate against a Certificate Authority list?
<i>certBuff</i>	Buffer containing the certificate data
<i>certBuffSize</i>	Size of the buffer containing the certificate data
<i>certBuffFormat</i>	The format of the certificate buffer
<i>bIsClient</i>	Perform the TLS negotiation as the Client and not the Server?

Return values

>0	The file descriptor of the connected SSL socket
<0	The error code for the failure from SSL Error Codes or TCP Socket Status

See also

[SSL_accept\(\)](#)
[SSL_GetSocketRemoteAddr\(\)](#)
[SSL_GetSocketRemotePort\(\)](#)
[SSL_GetSocketLocalAddr\(\)](#)
[SSL_GetSocketLocalPort\(\)](#)
[IsSSLfd\(\)](#)
[SSL_setsockopt\(\)](#)
[SSL_clrsockopt\(\)](#)
[SSL_getsockopt\(\)](#)

14.45.3.34 SSL_ServerReadyCert()

```

void SSL_ServerReadyCert (
    const unsigned char * cert,
    const unsigned char * key,
    uint16_t certBuffFormat = SSL_FILETYPE_PEM)

```

#include <ssl.h>

Add a key and certificate to the server.

Parameters

<i>cert</i>	Pointer to the buffer holding the PEM certificate data
<i>key</i>	Pointer to the buffer holding the PEM key data
<i>certBuffFormat</i>	The format of the cert being passed in. Either SSL_FILETYPE_PEM or WOLFSSL_FILETYPE_ASN1.

14.45.3.35 SSL_TcpGetLastRxInterval()

```

uint32_t SSL_TcpGetLastRxInterval (
    int fd) [inline]

```

#include <ssl.h>

SSL mirror of [TcpGetLastRxInterval\(\)](#).

Parameters

<i>fd</i>	The socket file descriptor.

Returns

The number of system Time Ticks since the last packet was received on the connection.

See also

[SSL_TcpSendKeepAlive\(\)](#)
[TcpGetLastRxInterval\(\)](#)

14.45.3.36 SSL_TcpGetLastRxTime()

```
uint32_t SSL_TcpGetLastRxTime (
    int fd) [inline]
#include <ssl.h>
SSL mirror of TcpGetLastRxTime\(\).
```

Parameters

<i>fd</i>	The socket file descriptor.

Returns

The value of system Time Ticks when the last packet was received on the connection.

See also

[SSL_TcpSendKeepAlive\(\)](#)
[TcpGetLastRxTime\(\)](#)

14.45.3.37 SSL_TcpSendKeepAlive()

```
void SSL_TcpSendKeepAlive (
    int fd) [inline]
#include <ssl.h>
SSL mirror of TcpSendKeepAlive\(\).
```

Parameters

<i>fd</i>	The socket file descriptor.

See also

[SSL_TcpGetLastRxTime\(\)](#)
[TcpSendKeepAlive\(\)](#)

14.45.3.38 SSLGetRxBufferSpaceUsed()

```
uint16_t SSLGetRxBufferSpaceUsed (
    int fd)
#include <ssl.h>
Returns the amount of data buffered to be read from the socket.
```

Parameters

<i>fd</i>	The socket's file descriptor

Return values

≥ 0	The number of bytes available to be read
< 0	0 The requested socket does not exist

14.45.3.39 SSLGetTxBufferAvailSpace()

```
uint16_t SSLGetTxBufferAvailSpace (
    int fd)
#include <ssl.h>
Returns the amount of data buffered to be read from the socket.
```

Parameters

<i>fd</i>	The socket's file descriptor

Return values

≥ 0	The number of bytes available to be read
< 0	0 The requested socket does not exist

14.45.3.40 SslInit()

```
void SslInit ()
#include <ssl.h>
```

This function initializes the SSL library.

Note: It should only be called in the case that you want to verify peer certificates as a client and need to add the certificates individually. Otherwise, calls to [SSL_connect\(\)](#) will automatically call this function for you.

14.45.3.41 SSLReadWithTimeout()

```
int SSLReadWithTimeout (
    int fd,
    char * cp,
    int size,
    uint32_t timeout)
#include <ssl.h>
```

SSL mirror of the ReadWithTiemout() function.

This function will block until either the time-out expires or at least one byte is available to be read. This function can be used to read from stdio, TCP sockets, or Serial ports.

Important: This function operates like a read function in that it reads all available bytes and returns. The addition of a time-out does not cause the function to block until the maximum number of bytes specified in the function call is available. The application must use the return value of the function to determine how many bytes were read, and call the function again if necessary.

Parameters

<i>fd</i>	The file descriptor number.
<i>*cp</i>	A pointer to the read destination.
<i>size</i>	Maximum number of bytes to read.
<i>timeout</i>	The number of timer ticks to wait for data.

Return values

0	Invalid buf or size parameters were passed
> 0	The number of bytes read
<i>TCP_ERR_TIMEOUT</i>	No bytes were read before timeout
<i>TCP_ERR_CLOSING</i>	TCP connection is no longer valid, or the SSLSocket_Record no longer exists

See also

[ReadWithTimeout\(\)](#)

[SSL_negotiate\(\)](#)

14.45.3.42 SslSetVerifyPeerClient()

```
void SslSetVerifyPeerClient (
    bool verifyPeer)
```

```
#include <ssl.h>
```

Sets whether or not verify peer should be enabled on the global SSL client context.

Parameters

<i>verifyPeer</i>	True if verify should be enabled, or false otherwise.

14.45.3.43 SslSetVerifyPeerServer()

```
void SslSetVerifyPeerServer (
    bool verifyPeer)
```

```
#include <ssl.h>
```

Sets whether or not verify peer should be enabled on the global SSL server context.

Parameters

<i>verifyPeer</i>	True if verify should be enabled, or false otherwise.

14.45.3.44 SSLSocketsAvail()

```
bool SSLSocketsAvail ()
```

```
#include <ssl.h>
```

Returns whether or not there are any sockets available.

Return values

<i>true</i>	If there are sockets available.
<i>false</i>	If there are not any sockets available

See also

[FreeSSLSockets\(\)](#)

14.45.4 Variable Documentation**14.45.4.1 IsSSLfd**

```
BOOL(* IsSSLfd) (int fd) (
    int fd) [extern]
```

```
#include <ssl.h>
```

Returns whether or not a file descriptor is an SSL file descriptor.

This function is used to determine if the fd is an SSL connection. It can be used by things like the callback functions of the web server to determine how secure the fd is before sending sensitive information over it.

Parameters

<i>fd</i>	The file descriptor to test.

Return values

<i>TRUE</i>	If it is an SSL fd
<i>FALSE</i>	If it is not an SSL fd or if there was an error

See also

[SSL_GetSocketRemoteAddr\(\)](#)[SSL_GetSocketRemotePort\(\)](#)[SSL_GetSocketLocalAddr\(\)](#)[SSL_GetSocketLocalPort\(\)](#)**14.45.4.2 SSL_clrsocketoption**

```
int(* SSL_clrsocketoption) (int fd, int option) (  
    int fd,  
    int option) [extern]
```

#include <ssl.h>

Clear SSL TCP socket options.

Parameters

<i>fd</i>	Socket file descriptor.
<i>option</i>	Socket option to clear: TCP Socket Options .

Returns

A bitmask of the options for the specified socket

See also

[SSL_setsocketoption\(\)](#)[SSL_getsocketoption\(\)](#)**14.45.4.3 SSL_GetRawSocket**

```
int(* SSL_GetRawSocket) (int sslSocketFd) (  
    int sslSocketFd) [extern]
```

#include <ssl.h>

Returns the TCP file descriptor associated with an SSL socket file descriptor.

Parameters

<i>sslSocketFd</i>	The socket file descriptor.

Returns

The TCP file descriptor associated with the SSL socket.

See also

[SSL_GetSocketInterface\(\)](#)**14.45.4.4 SSL_getsocketoption**

```
int(* SSL_getsocketoption) (int fd) (  
    int fd) [extern]
```

#include <ssl.h>

Returns the options for the specified SSL TCP socket.

Parameters

<i>fd</i>	Socket file descriptor.

Returns

A bitmask of the options for the specified socket

See also

[SSL_setsockopt\(\)](#)

[SSL_clrsockopt\(\)](#)

14.45.4.5 SSL_setsockopt

```
int (* SSL_setsockopt) (int fd, int option) (  
    int fd,  
    int option) [extern]
```

#include <ssl.h>

Set SSL TCP socket options.

Parameters

<i>fd</i>	Socket file descriptor.
<i>option</i>	Socket option to set: TCP Socket Options .

Returns

A bitmask of the options for the specified socket

See also

[SSL_clrsockopt\(\)](#)

[SSL_getsockopt\(\)](#)

14.45.5 ACME Certificate Generation

Request valid public certificates from an ACME certificate authority for devices with a public IP and DNS.

Classes

- class [AcmeServletObject](#)
Main ACME Client Class.
- class [LetsEncryptAcmeServletObject](#)
ACME Client for Let's Encrypt.
- class [BuyPassAcmeServletObject](#)
ACME Client for BuyPass.

14.45.5.1 Detailed Description

Request valid public certificates from an ACME certificate authority for devices with a public IP and DNS.

To use, you can simply specify these constants, NTP, and the ACME Servlet in your program, editing as desired:

```
const char * DEFAULT_CERT_ALT_NAMES= "test.example.com,test2.example.com"; // alt name is required  
const char * DEFAULT_CERT_COM_NAME= "test.example.com";  
const char * DEFAULT_CERT_EMAIL= "changeme@example.com";  
NtpClientServlet ntp_client;
```

You have a choice of supported providers or a custom provider. Just add to your code:

- Let's Encrypt: `LetsEncryptAcmeServletObject acme_client;`
 - Let's Encrypt Staging (for testing, no rate limits): `LetsEncryptAcmeServletObject acme_client(true);`
- BuyPass: `BuyPassAcmeServletObject acme_client;`
- Custom: `AcmeServletObject acme_client("https://192.168.10.90:14000/dir");`

Then for debugging you can add `acme_client.SetDiag(true);` to your `UserMain()`

If your device is accessible on the Internet at those DNS names, Let's Encrypt should automatically issue you a certificate within about a minute of boot.

The NetBurner Config system accepts changing the certificate data under "CertData". You can also provide it in the constructor:

```
AcmeServletObject acme_client("https://acme.example.com/dir");
```

EAB (External Account Binding) support is planned for the future, which paid providers like ZeroSSL, SSL.com and Google require.

See the [ACME Certificate Generation \(Easy Servlet\)](#) example for more details.

14.45.6 Onboard Self-Signed Certificate Generation

See [On-board Cert Generation - Simple](#) and other examples in the examples/SSL folder.

Classes

- struct `CertGenData`

Used to store the information that is passed in when enabling onboard generated certificates, as well as the information needed to determine when they need to be regenerated.
- class `CertData`

Enumerations

- enum `SslKeyType_t` { }

Certificate Generation Key Types.
- enum `AltNameType_t` : char { `ALT_NAME_TYPE_NONE` = 0 , `ALT_NAME_TYPE_DNS` = (char)0x82 , `ALT_NAME_TYPE_IP` = (char)0x87 }

Certificate Generation Alternate Name Types.
- enum `CertGenReturnCode` : int {
`CERT_GEN_RETURN_SUCCESS` = 1 , `CERT_GEN_RETURN_NOT_REQUIRED` = 2 , `CERT_GEN_RETURN_INVALID_RANGE` = -1 , `CERT_GEN_RETURN_MALLOC_ERROR` = -2 , `CERT_GEN_RETURN_ECC_INIT_ERROR` = -3 , `CERT_GEN_RETURN_ECC_CREATE_ERROR` = -4 , `CERT_GEN_RETURN_RSA_INIT_ERROR` = -5 , `CERT_GEN_RETURN_RSA_CREATE_ERROR` = -6 , `CERT_GEN_RETURN_CERT_INIT_ERROR` = -7 , `CERT_GEN_RETURN_CERT_CREATE_ERROR` = -8 , `CERT_GEN_RETURN_CERT_SIGNING_ERROR` = -9 , `CERT_GEN_RETURN_CERT_SAVING_ERROR` = -10 , `CERT_GEN_RETURN_KEY_CONVERT_ERROR` = -11 , `CERT_GEN_RETURN_KEY_SAVING_ERROR` = -12 , `CERT_GEN_RETURN_SET_ISSUER_ERROR` = -13 }

Certificate Generation Function Return Codes.

Functions

- `CertGenReturnCode SSL_CreateNewSelfSignedCert (CertGenData &pGenData)`

Generates a new self-signed certificate that will be stored on the device.
- void `EnableOnboardCertificateCreation (CertGenReturnCode(*createCertFunc)(CertGenData &cGenData)=SSL_CreateNewSelfSignedCert)`

Enables the ability for the module to automatically generate self-signed certificates for use in SSL/TLS connections when acting as the server.
- void `FillInAltNamesASN1 (Cert &TheCert, NBString &altNameString)`

Helper function used to build ASN1 representation of alt names.

- [CertGenReturnCode](#) **CheckAndCreateHalCertAndKey** ()

Check for a valid SSL certificate and generate one if not found.

- [CertGenData](#) * [GetDataForCertGen](#) ()

A function to be overridden to pass certificate details to the certificate generator.

14.45.6.1 Detailed Description

See [On-board Cert Generation - Simple](#) and other examples in the examples/SSL folder.

14.45.6.2 Enumeration Type Documentation

14.45.6.2.1 AltNameType_t

```
enum AltNameType_t : char
#include <certgen.h>
```

Certificate Generation Alternate Name Types.

Enumerator

ALT_NAME_TYPE_NONE	None.
ALT_NAME_TYPE_DNS	DNS Name.
ALT_NAME_TYPE_IP	IP Address.

14.45.6.2.2 CertGenReturnCode

```
enum CertGenReturnCode : int
#include <certgen.h>
```

Certificate Generation Function Return Codes.

Enumerator

CERT_GEN_RETURN_SUCCESS	Returned when the certificate was successfully generated.
CERT_GEN_RETURN_NOT_REQUIRED	Returned when the a new certificate was not required to be generated.
CERT_GEN_RETURN_INVALID_RANDOM	Returned when the device was unable to generate a valid random number.
CERT_GEN_RETURN_MALLOC_ERROR	Returned when the device is unable to malloc enough space to generate the certificate.
CERT_GEN_RETURN_ECC_INIT_ERROR	Returned when the device was unable to initialize the required ECC object.
CERT_GEN_RETURN_ECC_CREATE_ERROR	Returned when the device was unable to create the ECC key.
CERT_GEN_RETURN_RSA_INIT_ERROR	Returned when the device was unable to initialize the required RSA object.
CERT_GEN_RETURN_RSA_CREATE_ERROR	Returned when the device was unable to create the RSA key.
CERT_GEN_RETURN_CERT_INIT_ERROR	Returned when the device was unable to initialize the required certificate object.
CERT_GEN_RETURN_CERT_CREATE_ERROR	Returned when the device was unable to create the certificate.
CERT_GEN_RETURN_CERT_SIGNING_ERROR	Returned when the device was unable to sign the created certificate.
CERT_GEN_RETURN_CERT_SAVING_ERROR	Returned when the device was unable to properly save the certificate.

Enumerator

CERT_GEN_RETURN_KEY_CONVERT_ERROR	Returned when the device was unable to convert the certificate from PEM to DER format.
CERT_GEN_RETURN_KEY_SAVING_ERROR	Returned when the device was unable to save the key.
CERT_GEN_RETURN_SET_ISSUER_ERROR	Returned when the device was unable to set the issuer if a CA was used.

14.45.6.2.3 SslKeyType_t

```
enum SslKeyType_t
```

```
#include <certgen.h>
```

Certificate Generation Key Types.

Enumerator

SSL_KEY_ECC_SECP192R1	Defined in wolfssl/wolfcrypt/ecc.h.
SSL_KEY_ECC_SECP224R1	Defined in wolfssl/wolfcrypt/ecc.h.
SSL_KEY_ECC_SECP256R1	Defined in wolfssl/wolfcrypt/ecc.h.
SSL_KEY_ECC_SECP384R1	Defined in wolfssl/wolfcrypt/ecc.h.
SSL_KEY_ECC_SECP521R1	Defined in wolfssl/wolfcrypt/ecc.h.
SSL_KEY_RSA_1024	RSA 1024.
SSL_KEY_RSA_2048	RSA 2048.
SSL_KEY_RSA_4096	RSA 4096.

14.45.6.3 Function Documentation

14.45.6.3.1 EnableOnboardCertificateCreation()

```
void EnableOnboardCertificateCreation (
```

```
    CertGenReturnCode (* createCertFunc ) (CertGenData &cGenData) = SSL_CreateNewSelfSignedCert)
```

```
#include <certgen.h>
```

Enables the ability for the module to automatically generate self-signed certificates for use in SSL/TLS connections when acting as the server.

If a certificate is manually loaded on the module prior to the SSL server being initialized, that certificate will be used and nothing will be generated. If a certificate had been previously generated and a new certificate is manually loaded on the device, the newly loaded certificate will be used in place of the automatically generated one, though the automatically generated one will still be available on the module.

Parameters

<i>createCertFunc</i>	Optional parameter to specify a custom create certificate function.

14.45.6.3.2 FillInAltNamesASN1()

```
void FillInAltNamesASN1 (
```

```
    Cert & TheCert,
```

```
    NBString & altNameString)
```

```
#include <certgen.h>
```

Helper function used to build ASN1 representation of alt names.

Parameters

<i>TheCert</i>	The WolfSSL Cert structure to put the altnames in/onto.
<i>altNameString</i>	a NBAStrng holding the comman seperated list of alt names. A single string, with possibly multiple altnames separated by a comma. You can either include a tag telling the system what kind of AltName it is... So say you wanted : IP:192.168.1.36 and DNS:demo.netburner.com The following strings would both work: "IP:192.168.1.36,DNS:demo.netburner.com" "192.168.1.36,demo.netburner.com"

or dual interface IP's...

"192.168.1.36,10.1.1.35"

or both V4 and V6.... "IP:fe80::203:f4ff:fe0a:447f,192.168.1.36"

14.45.6.3.3 GetDataForCertGen()

```
CertGenData * GetDataForCertGen () [extern]
#include <certgen.h>
```

A function to be overridden to pass certificate details to the certificate generator.

The library provides this function as a weak reference. The values returned by this weak reference are likely wrong for any real application and the user should provide their own function. Applicable for both self signed and ACME certificates generated onboard.

Returns

pointer to a [CertGenData](#) struct

14.45.6.3.4 SSL_CreateNewSelfSignedCert()

```
CertGenReturnCode SSL_CreateNewSelfSignedCert (
    CertGenData & pGenData)
#include <certgen.h>
```

Generates a new self-signed certificate that will be stored on the device.

The common name and alternative names used are taken from the parameters passed in.

By default, the certificate is generated using ECC with SECP384R1. This can be changed to an RSA key by undefining `ENABLE_ECCKEY_CREATE` in `<nndk_install>\libraries\crypto\platform\<module name>\user_settings.h`, and then rebuilding your application. To specify what curve or RSA key length is used in certificate generation, define `DEFAULT_KEY_TYPE` as one of the values defined by `SslKeyType_t`.

Parameters

<i>pGenData</i>	A pointer to structure used to hold data for certificate generation

Returns

[CertGenReturnCode](#)

14.45.7 SSL Error Codes

Macros

- `#define SSL_ERROR_FAILED_NEGOTIATION (-256)`
The SSL system failed to successfully negotiate a connection.
- `#define SSL_ERROR_CERTIFICATE_UNKNOWN (-258)`
The received certificate could not be decoded.
- `#define SSL_ERROR_CERTIFICATE_NAME_FAILED (-260)`
The connected name did not match common_name.
- `#define SSL_ERROR_CERTIFICATE_VERIFY_FAILED (-261)`

- The server returned a certificate that we did not trust.*
- #define **SSL_ERROR_NO_BUFFER** (-262)
Unable to get a buffer from the pool.
- #define **SSL_ERROR_HANDSHAKE_INCOMPLETE** (-265)
The SSL handshake is not complete yet.
- #define **SSL_ERROR_NO_SERVER_CERTIFICATE** (-257)
The server wasn't able to load the certificate.
- #define **SSL_ERROR_NO_SERVER_KEY** (-258)
The server wasn't able to load the key.
- #define **SSL_ERROR_UNABLE_TO_LOAD_CIPHERS** (-259)
The client was unable to load the cipher list.
- #define **SSL_ERROR_CERTIFICATE_CHECK** (-260)
The server was unable to check the certificate.

14.45.7.1 Detailed Description

Error codes that can be returned by the NetBurner SSL functions.

14.46 Serial Interfaces

Topics

- [Serial Port Error Codes](#)

Macros

- #define **ADDR_ESCAPE_CHAR** (0xFF)
Address escape character.
- #define [SimpleOpenSerial](#)(p, b) [OpenSerial](#)(p, b, 1, 8, [eParityNone](#))
Simple open a serial port.

Enumerations

- enum [parity_mode](#) {
 [eParityNone](#) , [eParityOdd](#) , [eParityEven](#) , [eParityMulti](#) ,
 [eParityMultiOdd](#) , [eParityMultiEven](#) }
Serial Parity Modes.

Functions

- int [OpenSerial](#) (int portnum, unsigned int baudrate, int stop_bits, int data_bits, [parity_mode](#) parity)
Open a serial port.
- int [OpenDefaultSerial](#) ()
Opens the Default serial port as defined by the Boot Config settings.
- void [SerialExpandRxBuffer](#) (int fd, int nb)
Expand the received serial buffer.
- int [SerialClose](#) (int portnum)
Close a serial port.
- void [SerialEnableTxFlow](#) (int port, int enab)
Enable transmit software flow control on the specified UART.
- void [SerialEnableRxFlow](#) (int port, int enab)
- void [SerialEnableHwTxFlow](#) (int port, int enab)
- void [SerialEnableHwRxFlow](#) (int port, int enab)
- void [Serial485HalfDupMode](#) (int port, int enab)

- void [SendBreak](#) (int port, uint32_t time)
- int [serwriteaddress](#) (int fd, const char c)
- int [GetUartErrorReg](#) (int fd)
- void [SetRTS](#) (int port, bool val)
- BOOL [SerialSendComplete](#) (int fd)

14.46.1 Detailed Description

```
#include< serial.h>
```

The NetBurner Serial API

14.46.2 Macro Definition Documentation

14.46.2.1 SimpleOpenSerial

```
#define SimpleOpenSerial(  
    p,  
    b) OpenSerial(p, b, 1, 8, eParityNone)
```

```
#include <serial.h>
```

Simple open a serial port.

Select the UART number and baud rate, default parameters will be 1 stop bit, no parity, 8 data bits.

Parameters

<i>p</i>	The UART to open. First port is 0. The maximum number is platform dependent
<i>b</i>	The speed of the serial port in bits per second

See also

[OpenSerial\(\)](#)

14.46.3 Enumeration Type Documentation

14.46.3.1 parity_mode

```
enum parity\_mode  
#include <serial.h>  
Serial Parity Modes.
```

Enumerator

eParityNone	No parity.
eParityOdd	Odd parity.
eParityEven	Even parity.
eParityMulti	Multi parity.
eParityMultiOdd	Multi mode, Odd parity.
eParityMultiEven	Multi mode, Even parity.

14.46.4 Function Documentation

14.46.4.1 GetUartErrorReg()

```
int GetUartErrorReg (  
    int fd)
```

```
#include <serial.h>
```

Gets the UART error register. This is only applicable on select platforms. Any errors are added to this register in a logical OR operation. Calling this function to read the error status will clear the register.

Bit Description

3 Received break 2 Framing error 1 Parity error 0 Overrun error

fd - The file descriptor associated with the UART serial port whose error register is to be retrieved.

return - The value of the error register if successful (positive int - the bitwise values are represented by the UART_↵
_ERR_* definitions found near the top of this file); otherwise, one of the error codes is returned: SERIAL_ERR_↵
NOSUCH_PORT (-1) SERIAL_ERR_PORT_NOTOPEN (-2)

14.46.4.2 OpenDefaultSerial()

```
int OpenDefaultSerial ()
```

```
#include <serial.h>
```

Opens the Default serial port as defined by the Boot Config settings.

This function opens the Default serial port as defined by the Boot Config settings. It is called automatically by the system as part of the system initialization process. Unlike OpenSerial and SimpleOpenSerial, OpenDefaultSerial will configure the Tx and Rx pins to the Boot Config settings.

Returns

The file descriptor of the default serial port (> 0), or [Serial Port Error Codes](#)

See also

[OpenSerial\(\)](#)

14.46.4.3 OpenSerial()

```
int OpenSerial (
    int portnum,
    unsigned int baudrate,
    int stop_bits,
    int data_bits,
    parity_mode parity)
#include <serial.h>
```

Open a serial port.

This function opens a serial port. Note that functions to open a serial port, such as [OpenSerial\(\)](#) and [SimpleOpenSerial\(\)](#), must be called before any functions or pin assignments that configure the operation of the serial port. You must also configure all pins that have the same UART capability. For example, if UART 1 TX can be configured to come out on more than one pin (as a second or third alternate function), you must set those pins to some function other than UART 1 TX. This is because the functions that open a serial port will configure all serial port functions to default operation and default pins. For example, hardware flow control will be off, and the default pin assignments will be used on those platform that have multiple outputs.

Note

MODM7AE70 and SBE70LC platforms have two UART types: USART for serial ports 0 and 1, and two wire UART for serial ports 2 and above that only support RS-232, 1 stop bit , 8 data bits, no parity.

Parameters

<i>portnum</i>	The UART to open. First port is 0. The maximum number is platform dependent
<i>baudrate</i>	The speed of the serial port in bits per second
<i>stop_bits</i>	The number of stop bits, 1 or 2 (Hardware may not support all values)
<i>data_bits</i>	The number of data bits sent per character or frame: 5, 6, 7 or 8 (Hardware may not support all values)
<i>parity</i>	The type of parity checking to use: eParityNone, eParityEven, eParityOdd or eParityMulti

Returns

The file descriptor of the serial port (> 0), or [Serial Port Error Codes](#)

See also

[SimpleOpenSerial\(\)](#)

14.46.4.4 SendBreak()

```
void SendBreak (
    int port,
    uint32_t time)
#include <serial.h>
```

Sets a break in the UART transmission for a given period of time. The break starts when character transmission completes. The break is delayed until any character in the transmitter shift register is sent. Any character in the transmitter holding register is sent after the break.

Note: This feature may not be supported on all UART modules. Please refer to the processor's datasheet for details on supported features.

port - The UART whose transmitter will be forced low (start break). time - Specifies the amount of time in ticks that the break will hold; when time expires, the break will be stopped; 20 ticks equal 1 second by default.

return - Nothing to return.

14.46.4.5 Serial485HalfDupMode()

```
void Serial485HalfDupMode (
    int port,
    int enab)
#include <serial.h>
```

Enables or disables RS-485 half-duplex mode. Full-duplex mode is automatically enabled when half-duplex mode is disabled. This must be explicitly called before RS-485 functionality can be used. Jumpers may also be needed depending on the hardware device and/or development board used.

port - The UART port to use; the UART that can only be used for RS-485 depends on the platform (e.g., MOD52xx mounted on the MOD-DEV-100 development board use only UART 0 for RS-485, CB34-EX use only UART 1 for RS-485). enab - '0' disables half-duplex (enables full-duplex); '1' enables half- duplex (disables full-duplex)

return - Nothing to return.

14.46.4.6 SerialClose()

```
int SerialClose (
    int portnum)
#include <serial.h>
```

Close a serial port.

Parameters

<i>portnum</i>	The UART to close. First port is 0. The maximum number is platform dependent

Returns

0 if successful, otherwise [Serial Port Error Codes](#)

See also

[OpenSerial\(\)](#)

14.46.4.7 SerialEnableHwRxFlow()

```
void SerialEnableHwRxFlow (
    int port,
    int enab)
```

```
#include <serial.h>
```

Enables or disables request-to-send hardware flow control on receive (RxRTS). When enabled, receive throttles the transmitter on the other end. Jumpers may also be needed to enable the RxRTS line depending on the device and/or development board used.

(NB Device) Rx <--<--< Tx (Other Device) (NB Device) RxRTS >-->--> TxCTS (Other Device)

port - The UART whose flow control will be toggled. enab - '0' disables flow control; '1' (or any non-zero number) will enable it.

return - Nothing to return.

14.46.4.8 SerialEnableHwTxFlow()

```
void SerialEnableHwTxFlow (
    int port,
    int enab)
```

```
#include <serial.h>
```

Enables or disables clear-to-send hardware flow control on transmit (TxCTS). When enabled, transmit is throttled by the receiver on the other end. Jumpers may also be needed to enable the TxCTS line depending on the device and/or development board used.

(NB Device) Tx >-->--> Rx (Other Device) (NB Device) TxCTS <--<--< RxRTS (Other Device)

Note: Calling this function to disable Tx hardware flow control after enabling RS-485 full-duplex mode configures for RS-422 mode (Tx stays actively driven).

port - The UART whose flow control will be toggled. enab - '0' disables flow control; '1' (or any non-zero number) will enable it.

return - Nothing to return.

14.46.4.9 SerialEnableRxFlow()

```
void SerialEnableRxFlow (
    int port,
    int enab)
```

```
#include <serial.h>
```

Enables or disables software flow control (XON/XOFF) on receive. When enabled, the NetBurner device will send special XON and XOFF characters to another device in order to control the flow of incoming data.

port - The UART whose flow control will be toggled. enab - '0' disables flow control; '1' (or any non-zero number) will enable it.

return - Nothing to return.

14.46.4.10 SerialEnableTxFlow()

```
void SerialEnableTxFlow (
    int port,
    int enab)
```

```
#include <serial.h>
```

Enable transmit software flow control on the specified UART.

Enables or disables software flow control (XON/XOFF) on transmit. When enabled, the NetBurner device will recognize the special XON and XOFF characters being sent from another device in order to throttle the output.

Parameters

<i>port</i>	UART number
<i>enab</i>	0 disables flow control, 1 enables flow control

14.46.4.11 SerialExpandRxBuffer()

```
void SerialExpandRxBuffer (
    int fd,
    int nb)
```

```
#include <serial.h>
```

Expand the received serial buffer.

This function expands the number of buffer descriptors the specific serial port is allowed to use. Each buffer descriptor counts for 1500 bytes so setting this to 10 would make the serial Rx buffer 15000 bytes.

14.46.4.12 SerialSendComplete()

```
BOOL SerialSendComplete (
    int fd)
```

```
#include <serial.h>
```

Determines if data waiting in transmitter holding/shift registers are already sent out.

return - True if finished, false if data still sending

14.46.4.13 serwriteaddress()

```
int serwriteaddress (
    int fd,
    const char c)
```

```
#include <serial.h>
```

Sends an address character via the UART port number associated with the given file descriptor. This function can only be utilized if the UART is initialized in multidrop mode.

fd - The file descriptor associated with the UART serial port that will be used. c - The address character to be sent.

return - '1' if successful; otherwise, one of the error codes is returned (note that SERIAL_ERR_PORT_NOTOPEN is also returned if UART is not initialized to be in multidrop mode): SERIAL_ERR_NOSUCH_PORT (-1) SERIAL_ERR_PORT_NOTOPEN (-2)

14.46.4.14 SetRTS()

```
void SetRTS (
    int port,
    bool val)
```

```
#include <serial.h>
```

Sets or clears the Request To Send signal for the specified serial port.

port - The UART port whose RTS signal will be set or cleared. (0 or 1) val - The Boolean value to configure the signal; TRUE sets it, while FALSE clears it.

return - Nothing to return.

14.46.5 Serial Port Error Codes**Macros**

- **#define SERIAL_ERR_NOSUCH_PORT (-1)**
Port number does not exist.
- **#define SERIAL_ERR_PORT_NOTOPEN (-2)**
Port is not open.
- **#define SERIAL_ERR_PORT_ALREADYOPEN (-3)**
Port is already open.
- **#define SERIAL_ERR_PARAM_ERROR (-4)**
Parameter error.

14.46.5.1 Detailed Description

```
#include< serial.h>
```

14.47 Shutdown Notifications

Topics

- [ShutdownReasons](#)

Shutdown Reasons

```
#include< ShutDownNotifications.h>
```

Functions

- bool [NBApproveShutdown](#) (int reason)
Approve action that will result in a reboot.
- void [NBFaultNotify](#) ()
Emergency notification of a fault shutdown.

14.47.1 Detailed Description

```
#include< ShutDownNotifications.h>
```

The NBApproveShutdown callback function can be used by an application to put the system in a safe state before an application update, configuration update, or reboot event occurs. For example, closing active TCP sockets, ensuring Flash and/or file system write operations are complete, and putting critical peripherals in a safe state. The [NBApproveShutdown\(\)](#) function as a weak reference to a system function that always returns true by default. If an application creates its own function using the same signature, that function will be used instead.

A reason for the reboot request is passed to the function. The system will automatically call NBApproveShutdown() for the following reasons:

```
#define SHUTDOWN_CODEUPDATE (1) // A code update is requested
#define SHUTDOWN_CONFIGURE_REBOOT (2) // Configuration values have been modified with a requested reboot
```

An application can choose to ignore the parameters or add its own. For custom reasons, the application should call [NBApproveShutdown\(\)](#) with the appropriate reason:

```
// Custom reboot reason. System reasons start at 1, so pick something much larger
#define SHUTDOWN_CUSTOM_REBOOT 100

if ( NBApproveShutdown (SHUTDOWN_CUSTOM_REBOOT) )
{
    OSTimeDly (TICKS_PER_SECOND * 5);
    ForceReboot ();
}
```

14.47.2 Function Documentation

14.47.2.1 NBApproveShutdown()

```
bool NBApproveShutdown (
    int reason)
#include <ShutDownNotifications.h>
Approve action that will result in a reboot.
```

Note

Called for application updates or configuration changes that include the reboot option

This function will be called when the configuration values have changed with the option to reboot, or for an application update (always requires a reboot).

If the process fails at some time after this notification is approved, the system may not execute a reboot. This function is satisfied as a weak reference in the NetBurner library. In order to use it in your application, create a function of the same name (callback function).

Parameters

<i>reason</i>	The reason for the shutdown request (ShutdownReasons)

Return values

<i>true</i>	Ok to proceed with the modification and reboot
<i>false</i>	Abort the update

14.47.2.2 NBFaultNotify()

```
void NBFaultNotify ()  
#include <ShutDownNotifications.h>  
Emergency notification of a fault shutdown.
```

Note

Called after a critical error, such as a trap or memory fault before a reboot

If the system has a critical fault of some kind it will call this function before rebooting. Since this is a sever critical fault, the function is being called in an exception context with a very small stack. It should execute whatever is necessary and return immediately with a minimal number of variables. It is extremely limited in the actions it can perform. It cannot execute:

- Any RTOS functions
- printf family functions
- Writes to network sockets
- TCP or UDP functions
- Writes to serial ports
- RTOS delay functions
- Writes to Flash memory
- File system operations

Since this is a critical fault condition and the system is unstable, writing directly to I/O registers has the most chance of success.

14.47.3 ShutdownReasons

Shutdown Reasons

```
#include< ShutDownNotifications.h>
```

.

Macros

- **#define SHUTDOWN_CODEUPDATE (1)**
A code update is requested.
- **#define SHUTDOWN_CONFIGURE_REBOOT (2)**
Configuration values have been modified with a requested reboot.

14.47.3.1 Detailed Description

Shutdown Reasons

```
#include< ShutDownNotifications.h>
```

.

14.48 Signed Application Update

Topics

- [NBUpdate Function Return Values](#)

Functions

- void [RegisterAppSigningPublicKey](#) (const char *pKey)
Enable APP Signing by registering a RSA PEM or DER format public key.
- int [ProgramApplication](#) (uint32_t where, uint8_t *pAppImage)
Program an appliction image into Flash memory.
- int [UpdateFromStream](#) (int fd, AppUpdateRecord *&pu, uint32_t timeout)
Program/update an application image from a data stream.

14.48.1 Detailed Description

```
#include< nbupdate.h>
```

14.48.2 Function Documentation

14.48.2.1 ProgramApplication()

```
int ProgramApplication (
    uint32_t where,
    uint8_t * pAppImage)
```

```
#include <nbupdate.h>
```

Program an appliction image into Flash memory.

Parameters

<i>where</i>	Location in Flash memory
<i>pAppImage</i>	Pointer to the application image to program

Return values

0	Success
-1	Application image pointer invalid
-2	Invalid platform header

14.48.2.2 RegisterAppSigningPublicKey()

```
void RegisterAppSigningPublicKey (
    const char * pKey)
```

```
#include <nbupdate.h>
```

Enable APP Signing by registering a RSA PEM or DER format public key.

This function should be called BEFORE the [init\(\)](#) function if you want to guarantee security. It is ok to register a temporary text blob early, then overwrite it once the key via other means.

Parameters

<i>pKey</i>	Pointer to the public key to register

14.48.2.3 UpdateFromStream()

```
int UpdateFromStream (
    int fd,
    AppUpdateRecord *& pu,
    uint32_t timeout)
```

```
#include <nbupdate.h>
```

Program/update an application image from a data stream.

Parameters

<i>fd</i>	File descriptor of data stream
<i>pu</i>	Application record
<i>timeout</i>	Timeout value in system time ticks (eg TICKS_PER*SECOND * 10)

Return values

0	Success
-1	Application image pointer invalid
-2	Invalid platform header

14.48.3 NBUpdate Function Return Values

Macros

- **#define NBUP_ERR_NO_ERR 0**
No errors.
- **#define NBUP_ERR_BAD_SOCKET -1**
File descriptor socket is invlaid.
- **#define NBUP_ERR_TIMEOUT -2**
Timeout.
- **#define NBUP_ERR_TOO_LARGE -3**
Application update record is too large.
- **#define NBUP_ERR_WRONG_PLAT -4**
Device platform name does not match.
- **#define NBUP_ERR_BAD_PROG -5**
Programming failed, invalid checksum.
- **#define NBUP_ERR_BAD_AUTH -6**
Application update record authentication failed.

14.48.3.1 Detailed Description

```
#include< nbupdate.h>
```

14.49 Stream Update**Topics**

- [Stream Update Return Values](#)

Functions

- int [ReadBinaryApplicationCodeFromStream](#) (int fd)
Read a binary application image from an input stream and program flash memory.
- int [ReadS19ApplicationCodeFromStream](#) (int fd)
Read an ASCII application image in s-record .s19 format from an input stream and program flash memory.

14.49.1 Detailed Description

```
#include< StreamUpdate.h>
```

Functions to read an application form a source with a file descriptor. Useful for programming/updating an application image from sources such as flash memory cards, posts from other devices, and files received from FTP.

14.49.2 Function Documentation**14.49.2.1 ReadBinaryApplicationCodeFromStream()**

```
int ReadBinaryApplicationCodeFromStream (
    int fd)
```

```
#include <StreamUpdate.h>
```

Read a binary application image from an input stream and program flash memory.

Read a binary application image from an input stream and program the application image in flash memory. Flash memory will not be modified unless the entire application is received without error.

A reboot is required to run the new application. Before any reboot, the running application should free, close and clean up any items necessary to put the device in a safe state. For example, it would be better to close active TCP or FTP connections rather than just reset so that any connected clients can close normally. If you wish to reboot from a running application you can use the

```
void ForceReboot()
```

```
function ForceReboot().
```

Parameters

<i>fd</i>	The File Descriptor to read the input data stream from

Returns

[Stream Update Return Values](#)

14.49.2.2 ReadS19ApplicationCodeFromStream()

```
int ReadS19ApplicationCodeFromStream (
    int fd)
```

```
#include <StreamUpdate.h>
```

Read an ASCII application image in s-record .s19 format from an input stream and program flash memory.

Read a ASCII application image in s-record .s19 format from an input stream and program the application image in flash memory. Flash memory will not be modified unless the entire application is received without error.

A reboot is required to run the new application. Before any reboot, the running application should free, close and clean up any items necessary to put the device in a safe state. For example, it would be better to close active TCP or FTP connections rather than just reset so that any connected clients can close normally. If you wish to reboot from a running application you can use the

```
void ForceReboot()
```

```
function ForceReboot().
```

Parameters

<i>fd</i>	The File Descriptor to read the input data stream from

Returns

[Stream Update Return Values](#)

14.49.3 Stream Update Return Values

Macros

- `#define STREAM_UP_FAIL (0)`
Action failed.
- `#define STREAM_UP_OK (1)`
Action succeeded.

14.49.3.1 Detailed Description

```
#include< StreamUpdate.h>
```

14.50 Synchronous Audio Interface (SOMRT1061)

Topics

- [SAI Callback Types](#)
- [SAI Configuration Enumerations](#)
- [SAI Configuration Structures](#)
- [SAI Driver Class](#)
- [SAI Driver Constants](#)
- [SAI Predefined Configurations](#)

Files

- file [sai.h](#)
Synchronous Audio Interface (SAI/I2S) Driver for i.MX RT10xx.

Variables

- [SAICtx_t sai](#)
Global SAI driver instance.

14.50.1 Detailed Description

Note

Supported Platforms: SOMRT1061

14.50.2 Variable Documentation

14.50.2.1 sai

```
SAICtx_t sai [extern]
#include <sai.h>
```

Global SAI driver instance.

This is the main driver instance for SAI operations. Most applications will use this single global instance rather than creating their own.

```
extern SAICtx_t sai;
```

```
// Use the global instance
sai.Init(SAI_I2S_SLAVE_24_TXRX_RK);
```

14.50.3 SAI Callback Types

Typedefs

- typedef void(* [SAI_BufferDoneFn_t](#)) (void *buffer, int doneInvalid)

Callback function type for buffer completion notifications.

14.50.3.1 Detailed Description

14.50.3.2 Typedef Documentation

14.50.3.2.1 SAI_BufferDoneFn_t

```
typedef void(* SAI_BufferDoneFn_t) (void *buffer, int doneInvalid)
#include <sai.h>
```

Callback function type for buffer completion notifications.

This callback is invoked from interrupt context when a buffer transfer completes or encounters an error. The callback should return quickly and defer any lengthy processing to a task context.

Parameters

<i>buffer</i>	Pointer to the buffer that completed (same pointer passed to TransmitBuffer or ReadyReceiveBuffer)
<i>doneInvalid</i>	Status indicator for the buffer completion: • **< -1**: Channel error, buffer state unchanged (may be retried) • -1: Transfer error, buffer retired (do not reuse immediately) • 0: Half transfer complete (if half-notify enabled), second half still in progress • **> 0**: Buffer completely transferred successfully

Note

This callback executes in interrupt context. Keep processing minimal:

- Do NOT call blocking functions
- Do NOT perform long computations
- Use RTOS signaling (semaphores, flags) to defer work to tasks
- Avoid excessive printing

Warning

The buffer pointer is only valid if the completion status is non-negative. For error conditions (< 0), the buffer may be in an indeterminate state.

```
void myBufferCallback(void *buffer, int doneInvalid) {
    if (doneInvalid > 0) {
        // Success - buffer fully transferred
        OSSemPost(&bufferDoneSemaphore);
    } else if (doneInvalid == 0) {
        // Half complete - can start processing first half
        ProcessFirstHalf(buffer);
    } else {
        // Error condition
        iprintf("Buffer error: %d\n", doneInvalid);
    }
}
```

14.50.4 SAI Configuration Enumerations**Enumerations**

- enum `clkSrc_t` { `CLK_SRC_MCK` , `CLK_SRC_EXT` }
Clock source selection for SAI bit clock generation.
- enum `clkCh_t` { `CLK_CH_CH` , `CLK_CH_RK` , `CLK_CH_TK` }
Clock channel selection for determining which clock signal to use.
- enum `clkPol_t` { `CLK_ACTIVE_HIGH` , `CLK_ACTIVE_LOW` }
Clock polarity configuration.
- enum `frameSyncCh_t` { `SYNC_CH_CH` , `SYNC_CH_RX` , `SYNC_CH_TX` }
Frame synchronization channel selection.
- enum `startCond_t` {
 `START_CONTINUOUS` , `START_SYNC_RX_TX` , `START_FRAME_LOW` , `START_FRAME_HIGH` ,
 `START_FRAME_FALLING` , `START_FRAME_RISING` , `START_FRAME_LEVEL` , `START_FRAME_EDGE` ,
 `START_CMP_0` }
Starting conditions that trigger SAI data transfers.
- enum `clkGate_t` { `CLK_GATE_CONTINUOUS` , `CLK_GATE_FRAME_LOW` , `CLK_GATE_FRAME_HIGH` }
Clock gating mode for power optimization.
- enum `frameEdge_t` { `FRAME_SYNC_RISING` , `FRAME_SYNC_FALLING` }
Frame sync edge that indicates the start of a new frame.
- enum `clkOut_t` { `CLK_OUT_INPUT` , `CLK_OUT_CONTINUOUS` , `CLK_OUT_TRANSFER` }
Bit clock output mode configuration.
- enum `frameSyncOut_t` { `FRAME_SYNC_INPUT` , `FRAME_SYNC_NEGATIVE` , `FRAME_SYNC_POSITIVE` }
Frame synchronization signal output mode.
- enum `dataPacking_t` {
 `BUFFER_WORD8` , `BUFFER_WORD16_LEFT` , `BUFFER_WORD16_RIGHT` , `BUFFER_WORD32_LEFT` ,
 `BUFFER_WORD32_RIGHT` }
Buffer word format and data justification.
- enum `bitOrder_t` { `LEAST_SIG_FIRST` , `MOST_SIG_FIRST` }
Bit transmission order within each word.
- enum `bufferDepletionBehavior_t` { `DEPLETED_PAUSE` , `DEPLETED_REPEAT_LAST` }
DMA behavior when all queued buffers are exhausted.
- enum `rxTxSync_t` { `RXTX_ASYNC` , `RXTX_SYNC_RX` , `RXTX_SYNC_TX` }
Synchronization mode between RX and TX channels.

14.50.4.1 Detailed Description**14.50.4.2 Enumeration Type Documentation****14.50.4.2.1 bitOrder_t**

```
enum bitOrder_t
#include <sai.h>
```

Bit transmission order within each word.

Determines whether the least significant bit or most significant bit is transmitted first on the data line.

Enumerator

LEAST_SIG_FIRST	LSB transmitted first (uncommon for audio)
MOST_SIG_FIRST	MSB transmitted first (standard for I2S and most audio protocols)

14.50.4.2.2 bufferDepletionBehavior_t

```
enum bufferDepletionBehavior_t
#include <sai.h>
```

DMA behavior when all queued buffers are exhausted.

Defines what the DMA channel should do when it runs out of buffers to process. This is critical for preventing underruns/overruns.

Enumerator

DEPLETED_PAUSE	Pause transfers when buffers depleted (recommended for most uses)
DEPLETED_REPEAT_LAST	Continuously repeat the last buffer in the chain (useful for streaming)

14.50.4.2.3 clkCh_t

```
enum clkCh_t
#include <sai.h>
```

Clock channel selection for determining which clock signal to use.

Allows selection between transmit clock, receive clock, or channel default. Used for clock sharing between TX and RX channels.

Enumerator

CLK_CH_CH	Use the channel's default clock source
CLK_CH_RK	Use the receive clock (BCLK_RX) for this channel
CLK_CH_TK	Use the transmit clock (BCLK_TX) for this channel

14.50.4.2.4 clkGate_t

```
enum clkGate_t
#include <sai.h>
```

Clock gating mode for power optimization.

Controls when the bit clock is actively driven to reduce power consumption and EMI when no data transfer is occurring.

Enumerator

CLK_GATE_CONTINUOUS	Clock runs continuously (always enabled)
CLK_GATE_FRAME_LOW	Clock enabled only when frame sync is low
CLK_GATE_FRAME_HIGH	Clock enabled only when frame sync is high

14.50.4.2.5 clkOut_t

```
enum clkOut_t
#include <sai.h>
```

Bit clock output mode configuration.

Controls whether the bit clock is driven as an output (master mode) or used as an input (slave mode), and when it is active.

Enumerator

CLK_OUT_INPUT	Clock is an input; output driver is disabled (slave mode)
CLK_OUT_CONTINUOUS	Clock is continuously driven as output (master mode)
CLK_OUT_TRANSFER	Clock is driven only during active transfers (master mode with gating)

14.50.4.2.6 clkPol_t

```
enum clkPol_t
#include <sai.h>
```

Clock polarity configuration.

Determines the active edge of the bit clock for data sampling and driving.

Enumerator

CLK_ACTIVE_HIGH	Clock is active high: drive data on rising edge, sample on falling edge
CLK_ACTIVE_LOW	Clock is active low: drive data on falling edge, sample on rising edge

14.50.4.2.7 clkSrc_t

```
enum clkSrc_t
#include <sai.h>
```

Clock source selection for SAI bit clock generation.

The SAI can generate its bit clock from different sources depending on whether the device is operating as a master or slave.

Enumerator

CLK_SRC_MCK	Clock source is the divided peripheral master clock (for master mode)
CLK_SRC_EXT	Clock source is an external clock input (for slave mode or clock sharing)

14.50.4.2.8 dataPacking_t

```
enum dataPacking_t
#include <sai.h>
```

Buffer word format and data justification.

Defines how data is packed in memory buffers and whether it is left-justified or right-justified within the word. This affects how the DMA interprets the buffer data when moving it to/from the SAI FIFO.

Note

Left-justified packs MSBs at the upper bits of the word. Right-justified packs LSBs at the lower bits of the word.

Enumerator

BUFFER_WORD8	Each sample is 8 bits (1 byte per sample)

Enumerator

BUFFER_WORD16_LEFT	Each sample is 16 bits, left-justified in the word
BUFFER_WORD16_RIGHT	Each sample is 16 bits, right-justified in the word
BUFFER_WORD32_LEFT	Each sample is 32 bits, left-justified in the word
BUFFER_WORD32_RIGHT	Each sample is 32 bits, right-justified in the word

14.50.4.2.9 frameEdge_t

```
enum frameEdge_t
```

```
#include <sai.h>
```

Frame sync edge that indicates the start of a new frame.

Determines whether a rising or falling edge of the frame sync signal marks the beginning of a data frame.

Enumerator

FRAME_SYNC_RISING	New frame begins on rising edge of frame sync
FRAME_SYNC_FALLING	New frame begins on falling edge of frame sync

14.50.4.2.10 frameSyncCh_t

```
enum frameSyncCh_t
```

```
#include <sai.h>
```

Frame synchronization channel selection.

Determines which frame sync signal is used for synchronizing frame boundaries. Allows sharing of frame sync between TX and RX channels.

Enumerator

SYNC_CH_CH	Use the channel's default frame sync signal
SYNC_CH_RX	Use the receive frame sync (FRAME_SYNC_RX) for this channel
SYNC_CH_TX	Use the transmit frame sync (FRAME_SYNC_TX) for this channel

14.50.4.2.11 frameSyncOut_t

```
enum frameSyncOut_t
```

```
#include <sai.h>
```

Frame synchronization signal output mode.

Configures whether the frame sync is an input or output, and the polarity of the pulse when operating as an output.

Enumerator

FRAME_SYNC_INPUT	Frame sync is an input; output driver disabled (slave mode)
FRAME_SYNC_NEGATIVE	Frame sync driven with negative pulse at frame start (active low)
FRAME_SYNC_POSITIVE	Frame sync driven with positive pulse at frame start (active high)

14.50.4.2.12 rxTxSync_t

```
enum rxTxSync_t
```

```
#include <sai.h>
```

Synchronization mode between RX and TX channels.

Determines whether the receive and transmit channels operate independently or share clock and frame sync signals.

Enumerator

RXTX_ASYNC	RX and TX are completely independent (separate clocks and frame sync)
RXTX_SYNC_RX	TX synchronized to RX (both use RX clock and frame sync)
RXTX_SYNC_TX	RX synchronized to TX (both use TX clock and frame sync)

14.50.4.2.13 startCond_t

```
enum startCond_t
#include <sai.h>
```

Starting conditions that trigger SAI data transfers.

Defines when the SAI peripheral should begin transferring data. Different conditions are appropriate for different protocols and operating modes.

Enumerator

START_CONTINUOUS	Continuous operation: transfers start immediately when previous ends
START_SYNC_RX_TX	Synchronous start: transfers begin when opposite channel starts
START_FRAME_LOW	Start when frame sync signal is low
START_FRAME_HIGH	Start when frame sync signal is high
START_FRAME_FALLING	Start on falling edge of frame sync
START_FRAME_RISING	Start on rising edge of frame sync
START_FRAME_LEVEL	Start on any level change of frame sync
START_FRAME_EDGE	Start on any edge (rising or falling) of frame sync
START_CMP_0	Start when received data matches Compare 0 register value (RX only)

14.50.5 SAI Configuration Structures

Classes

- struct [SAI_rtx_cfg_t](#)
Configuration structure for one direction (TX or RX) of the SAI module.
- struct [SAI_cfg_t](#)
Complete configuration structure for the SAI driver.

14.50.5.1 Detailed Description**14.50.6 SAI Driver Class**

Classes

- class [SAICtx_t](#)
Main driver class for SAI peripheral operation.

14.50.6.1 Detailed Description**14.50.7 SAI Driver Constants**

Macros

- #define [SAI_HW_COUNT](#) 3
- #define [SAI_TXBD_COUNT](#) 4
- #define [SAI_RXBD_COUNT](#) 4

14.50.7.1 Detailed Description**14.50.7.2 Macro Definition Documentation****14.50.7.2.1 SAI_HW_COUNT**

```
#define SAI_HW_COUNT 3
#include <sai.h>
```

Number of SAI hardware instances available

14.50.7.2.2 SAI_RXBD_COUNT

```
#define SAI_RXBD_COUNT 4
#include <sai.h>
```

Number of receive buffer descriptors (maximum queued RX buffers)

14.50.7.2.3 SAI_TXBD_COUNT

```
#define SAI_TXBD_COUNT 4
#include <sai.h>
```

Number of transmit buffer descriptors (maximum queued TX buffers)

14.50.8 SAI Predefined Configurations**I2S Slave Mode Configurations**

These configurations set up the SAI as an I2S slave (receives clock from external master). Configurations are available for 8, 16, and 24-bit word widths, with various clock routing:

- `_RK`: Uses receive clock (BCLK from RX pins)
- `_TK`: Uses transmit clock (BCLK from TX pins)
- `_TKRK`: TX uses TX clock, RX uses RX clock (independent clocks)
- `const SAI_cfg_t SAI_I2S_SLAVE_24_TXRX_RK`
- `const SAI_cfg_t SAI_I2S_SLAVE_24_TXRX_TK`
- `const SAI_cfg_t SAI_I2S_SLAVE_24_TXRX_TKRK`
- `const SAI_cfg_t SAI_I2S_SLAVE_16_TXRX_RK`
- `const SAI_cfg_t SAI_I2S_SLAVE_16_TXRX_TK`
- `const SAI_cfg_t SAI_I2S_SLAVE_16_TXRX_TKRK`
- `const SAI_cfg_t SAI_I2S_SLAVE_8_TXRX_RK`
- `const SAI_cfg_t SAI_I2S_SLAVE_8_TXRX_TK`
- `const SAI_cfg_t SAI_I2S_SLAVE_8_TXRX_TKRK`

Left-Justified Slave Mode Configurations

These configurations set up the SAI for left-justified audio format (data starts immediately with frame sync, no delay). Available in 8, 16, and 24-bit variants.

- `const SAI_cfg_t SAI_LJUST_SLAVE_24_TXRX_RK`
- `const SAI_cfg_t SAI_LJUST_SLAVE_24_TXRX_TK`
- `const SAI_cfg_t SAI_LJUST_SLAVE_24_TXRX_TKRK`
- `const SAI_cfg_t SAI_LJUST_SLAVE_16_TXRX_RK`
- `const SAI_cfg_t SAI_LJUST_SLAVE_16_TXRX_TK`
- `const SAI_cfg_t SAI_LJUST_SLAVE_16_TXRX_TKRK`
- `const SAI_cfg_t SAI_LJUST_SLAVE_8_TXRX_RK`
- `const SAI_cfg_t SAI_LJUST_SLAVE_8_TXRX_TK`
- `const SAI_cfg_t SAI_LJUST_SLAVE_8_TXRX_TKRK`

14.50.8.1 Detailed Description**14.50.8.2 Variable Documentation****14.50.8.2.1 SAI_I2S_SLAVE_16_TXRX_RK**

```
const SAI_cfg_t SAI_I2S_SLAVE_16_TXRX_RK [extern]
#include <sai.h>
```

I2S Slave: 16-bit words, TX+RX enabled, clock from RX pins

14.50.8.2.2 SAI_I2S_SLAVE_16_TXRX_TK

```
const SAI_cfg_t SAI_I2S_SLAVE_16_TXRX_TK [extern]
#include <sai.h>
```

I2S Slave: 16-bit words, TX+RX enabled, clock from TX pins

14.50.8.2.3 SAI_I2S_SLAVE_16_TXRX_TKRK

```
const SAI_cfg_t SAI_I2S_SLAVE_16_TXRX_TKRK [extern]
#include <sai.h>
```

I2S Slave: 16-bit words, TX+RX enabled, independent clocks

14.50.8.2.4 SAI_I2S_SLAVE_24_TXRX_RK

```
const SAI_cfg_t SAI_I2S_SLAVE_24_TXRX_RK [extern]
#include <sai.h>
```

I2S Slave: 24-bit words, TX+RX enabled, clock from RX pins, frame sync from RX

14.50.8.2.5 SAI_I2S_SLAVE_24_TXRX_TK

```
const SAI_cfg_t SAI_I2S_SLAVE_24_TXRX_TK [extern]
#include <sai.h>
```

I2S Slave: 24-bit words, TX+RX enabled, clock from TX pins, frame sync from TX

14.50.8.2.6 SAI_I2S_SLAVE_24_TXRX_TKRK

```
const SAI_cfg_t SAI_I2S_SLAVE_24_TXRX_TKRK [extern]
#include <sai.h>
```

I2S Slave: 24-bit words, TX+RX enabled, independent clocks on TX and RX

14.50.8.2.7 SAI_I2S_SLAVE_8_TXRX_RK

```
const SAI_cfg_t SAI_I2S_SLAVE_8_TXRX_RK [extern]
#include <sai.h>
```

I2S Slave: 8-bit words, TX+RX enabled, clock from RX pins

14.50.8.2.8 SAI_I2S_SLAVE_8_TXRX_TK

```
const SAI_cfg_t SAI_I2S_SLAVE_8_TXRX_TK [extern]
#include <sai.h>
```

I2S Slave: 8-bit words, TX+RX enabled, clock from TX pins

14.50.8.2.9 SAI_I2S_SLAVE_8_TXRX_TKRK

```
const SAI_cfg_t SAI_I2S_SLAVE_8_TXRX_TKRK [extern]
#include <sai.h>
```

I2S Slave: 8-bit words, TX+RX enabled, independent clocks

14.50.8.2.10 SAI_LJUST_SLAVE_16_TXRX_RK

```
const SAI_cfg_t SAI_LJUST_SLAVE_16_TXRX_RK [extern]
#include <sai.h>
Left-Justified Slave: 16-bit words, TX+RX enabled, clock from RX pins
```

14.50.8.2.11 SAI_LJUST_SLAVE_16_TXRX_TK

```
const SAI_cfg_t SAI_LJUST_SLAVE_16_TXRX_TK [extern]
#include <sai.h>
Left-Justified Slave: 16-bit words, TX+RX enabled, clock from TX pins
```

14.50.8.2.12 SAI_LJUST_SLAVE_16_TXRX_TKRK

```
const SAI_cfg_t SAI_LJUST_SLAVE_16_TXRX_TKRK [extern]
#include <sai.h>
Left-Justified Slave: 16-bit words, TX+RX enabled, independent clocks
```

14.50.8.2.13 SAI_LJUST_SLAVE_24_TXRX_RK

```
const SAI_cfg_t SAI_LJUST_SLAVE_24_TXRX_RK [extern]
#include <sai.h>
Left-Justified Slave: 24-bit words, TX+RX enabled, clock from RX pins
```

14.50.8.2.14 SAI_LJUST_SLAVE_24_TXRX_TK

```
const SAI_cfg_t SAI_LJUST_SLAVE_24_TXRX_TK [extern]
#include <sai.h>
Left-Justified Slave: 24-bit words, TX+RX enabled, clock from TX pins
```

14.50.8.2.15 SAI_LJUST_SLAVE_24_TXRX_TKRK

```
const SAI_cfg_t SAI_LJUST_SLAVE_24_TXRX_TKRK [extern]
#include <sai.h>
Left-Justified Slave: 24-bit words, TX+RX enabled, independent clocks
```

14.50.8.2.16 SAI_LJUST_SLAVE_8_TXRX_RK

```
const SAI_cfg_t SAI_LJUST_SLAVE_8_TXRX_RK [extern]
#include <sai.h>
Left-Justified Slave: 8-bit words, TX+RX enabled, clock from RX pins
```

14.50.8.2.17 SAI_LJUST_SLAVE_8_TXRX_TK

```
const SAI_cfg_t SAI_LJUST_SLAVE_8_TXRX_TK [extern]
#include <sai.h>
Left-Justified Slave: 8-bit words, TX+RX enabled, clock from TX pins
```

14.50.8.2.18 SAI_LJUST_SLAVE_8_TXRX_TKRK

```
const SAI_cfg_t SAI_LJUST_SLAVE_8_TXRX_TKRK [extern]
#include <sai.h>
Left-Justified Slave: 8-bit words, TX+RX enabled, independent clocks
```

14.51 Syslog

Functions

- int [SysLog](#) (const char *format,...)

Send a formatted log message to the syslog server.

- int [SysLogVia](#) (int interfaceNum, const char *format,...)

Send a formatted log message via a specific network interface.

Variables

- [IPADDR SysLogAddress](#)

Destination IP address for syslog messages.

14.51.1 Detailed Description

```
#include< syslog.h>
```

14.51.2 Function Documentation

14.51.2.1 SysLog()

```
int SysLog (  
    const char * format,  
    ...)
```

```
#include <syslog.h>
```

Send a formatted log message to the syslog server.

This function sends a formatted text message to the configured syslog destination using UDP. The message format follows printf() conventions with full support for format specifiers including floating-point numbers.

The function:

- Formats the message according to the format string and arguments
- Prepends a syslog header (priority and timestamp information)
- Sends the message via UDP to SysLogAddress on port 514
- Uses the default network interface for transmission
- Returns immediately without waiting for acknowledgment (UDP is connectionless)

Parameters

<i>format</i>	Printf-style format string supporting all standard format specifiers: • d, i - signed decimal integer • u - unsigned decimal integer • x, X - hexadecimal • f, e, g - floating-point (increases code size) • s - string • c - character • p - pointer • Plus width, precision, and length modifiers
...	Variable arguments corresponding to format specifiers in the format string

Returns

- Number of characters sent (positive value) on success
- Negative value on error (e.g., network unavailable, invalid parameters)
- 0 if SysLogAddress is not configured (0.0.0.0)

Note

This function includes floating-point formatting support which adds approximately 30-40KB to the application binary size. If code size is critical, consider alternatives for logging.

Messages are sent immediately without buffering. High-frequency logging can impact network performance.

Maximum message length is typically limited by the UDP packet size (approximately 1400 bytes for safe transmission).

Warning

Do not call this function from interrupt context or with interrupts disabled, as it performs network I/O operations.

Basic Usage Examples:

```
#include <syslog.h>

void BasicLogging() {
    // Simple text message
    SysLog("Application started");

    // Formatted with integers
    int temperature = 72;
    SysLog("Temperature: %d degrees", temperature);

    // Multiple values
    int sensor1 = 100, sensor2 = 200;
    SysLog("Sensors: S1=%d, S2=%d", sensor1, sensor2);

    // Floating-point values
    float voltage = 3.3f;
    SysLog("Supply voltage: %.2f V", voltage);

    // Hexadecimal output
    uint32_t address = 0x08000000;
    SysLog("Flash base address: 0x%08X", address);
}
```

Advanced Usage Examples:

```
// Error logging with context
void LogError(const char *function, int errorCode) {
    SysLog("ERROR in %s: code=%d", function, errorCode);
}

// Network event logging
void OnNetworkEvent(const char *event, IPADDR addr) {
    SysLog("Network: %s from %hI", event, addr);
}

// Performance monitoring
void LogPerformance(const char *operation, uint32_t ticks) {
    float milliseconds = (float)ticks / TICKS_PER_SECOND * 1000.0f;
    SysLog("PERF: %s took %.2f ms", operation, milliseconds);
}

// Structured logging with severity
#define LOG_ERROR(msg, ...) SysLog("ERROR: " msg, ##__VA_ARGS__)
#define LOG_WARN(msg, ...) SysLog("WARN: " msg, ##__VA_ARGS__)
#define LOG_INFO(msg, ...) SysLog("INFO: " msg, ##__VA_ARGS__)

void StructuredLogging() {
    LOG_ERROR("Connection failed to 192.168.1.100");
    LOG_WARN("Buffer usage at 90%%");
    LOG_INFO("Processing request #%d", requestCount);
}
```

Conditional Logging:

```
// Enable/disable logging at compile time
#ifdef DEBUG_LOGGING
    #define DEBUG_LOG(msg, ...) SysLog("DEBUG: " msg, ##__VA_ARGS__)
#else
    #define DEBUG_LOG(msg, ...) do {} while(0)
#endif

void ConditionalLogging() {
    DEBUG_LOG("Entering function");
    // ... function code ...
    DEBUG_LOG("Exiting function");
}
```

See also[SysLogVia\(\)](#)[SysLogAddress](#)**14.51.2.2 SysLogVia()**

```
int SysLogVia (
    int interfaceNum,
    const char * format,
    ...)
```

```
#include <syslog.h>
```

Send a formatted log message via a specific network interface.

This function is identical to SysLog() but allows explicit selection of the network interface to use for transmission.

This is primarily useful when:

- Logging from network driver code
- Multiple network interfaces are active
- Routing or interface-specific debugging is needed
- Sending logs on a specific VLAN or network segment

The function operates identically to SysLog() but routes the UDP packet through the specified interface rather than using the default routing table.

Parameters

<i>interfaceNum</i>	Zero-based index of the network interface to use for transmission: • 0: First network interface (usually primary Ethernet) • 1: Second network interface (if available) • etc. Invalid interface numbers will cause the message to be dropped.
<i>format</i>	Printf-style format string (same as SysLog())
...	Variable arguments corresponding to format specifiers

Returns

- Number of characters sent (positive value) on success
- Negative value on error (e.g., invalid interface, network unavailable)
- 0 if SysLogAddress is not configured (0.0.0.0)

Note

This function is typically only needed for specialized network driver development or multi-interface debugging scenarios.

The specified interface must be active and have a valid IP configuration for the message to be sent successfully.

Warning

Ensure the interfaceNum is valid for your hardware platform. Invalid interface numbers will fail silently or return an error code.

Usage Examples:

```
#include <syslog.h>
#include <netinterface.h>

// Log via specific interface
void LogViaInterface() {
    // Log via primary interface (eth0)
    SysLogVia(0, "Message from primary interface");

    // Log via secondary interface (eth1), if available
    if (GetNumberOfInterfaces() > 1) {
        SysLogVia(1, "Message from secondary interface");
    }
}

// Network driver debugging
void DriverLog(int ifNum, const char *event) {
    SysLogVia(ifNum, "IF%d: %s", ifNum, event);
}

// Multi-interface monitoring
void MonitorAllInterfaces() {
    int numInterfaces = GetNumberOfInterfaces();

    for (int i = 0; i < numInterfaces; i++) {
        InterfaceBlock *pifb = GetInterfaceBlock(i);
        if (pifb) {
            SysLogVia(i, "Interface %d: IP=%hI, Link=%s",
                i, pifb->ip_addr,
                pifb->IsLinkActive() ? "UP" : "DOWN");
        }
    }
}

// VLAN-specific logging
void LogToVLAN(int vlanInterface, const char *message) {
    SysLogVia(vlanInterface, "VLAN: %s", message);
}
```

Network Driver Usage:

```
// Typical use in network driver code
void NetworkDriverRx(int ifNum, PoolPtr pp, uint16_t size) {
    // Log received packet on specific interface
    SysLogVia(ifNum, "RX on IF%d: %d bytes", ifNum, size);

    // Process packet...
}

void NetworkDriverTx(int ifNum, uint16_t size) {
    // Log transmitted packet
    SysLogVia(ifNum, "TX on IF%d: %d bytes", ifNum, size);
}
```

See also

[SysLog\(\)](#)

[SysLogAddress](#)

[GetNumberOfInterfaces\(\)](#)

[GetInterfaceBlock\(\)](#)

14.51.3 Variable Documentation

14.51.3.1 SysLogAddress

`IPADDR SysLogAddress [extern]`

`#include <syslog.h>`

Destination IP address for syslog messages.

This global variable specifies the IP address of the host or server where syslog messages should be sent. The syslog protocol uses UDP port 514 by default (this is hardcoded and cannot be changed).

Default behavior:

- If set to `IPADDR(0,0,0,0)` or not initialized: Messages are sent to the UDP broadcast address 255.255.255.255
- If set to a specific IP address: Messages are sent directly to that host

Note

The destination address can be changed at any time during runtime to redirect logging to different hosts or servers.

Broadcast messages (255.255.255.255) may not cross router boundaries. For logging across subnets, specify an explicit unicast address.

Usage Examples:

```
#include <syslog.h>
#include <ipaddr.h>

void SetupSyslog() {
    // Send to specific syslog server
    SysLogAddress = IPADDR(192, 168, 1, 100);
    SysLog("Syslog initialized, sending to 192.168.1.100");

    // Send to broadcast (all hosts on local network)
    SysLogAddress = IPADDR(255, 255, 255, 255);
    SysLog("Broadcasting syslog messages");

    // Disable syslog by setting to zero
    SysLogAddress = IPADDR(0, 0, 0, 0);
}

// Dynamic syslog server selection
void SelectSyslogServer(const char *serverIP) {
    IPADDR addr;
    if (AsciiToIp(serverIP, &addr) {
        SysLogAddress = addr;
        SysLog("Syslog server changed to %s", serverIP);
    } else {
        fprintf("Invalid IP address: %s\r\n", serverIP);
    }
}
```

See also

`SysLog()`

`SysLogVia()`

14.52 System Functions

NetBurner General System Functions.

Functions

- int `SaveUserParameters` (void *pCopyFrom, int len)
Save data to the on-chip flash memory User Parameter area.
- void * `GetUserParameters` (void)
Returns a void pointer to the user parameter area.
- const char * `GetReleaseTag` ()
Returns the NNDK release tag information.

14.52.1 Detailed Description

NetBurner General System Functions.

```
#include< system.h>
```

14.52.2 Function Documentation

14.52.2.1 GetReleaseTag()

```
const char * GetReleaseTag ()  
#include <system.h>
```

Returns the NNDK release tag information.

Returns

Pointer to a string representing the development tools release tag

14.52.2.2 GetUserParameters()

```
void * GetUserParameters (  
    void )  
#include <system.h>
```

Returns a void pointer to the user parameter area.

On most platforms the total amount of space is 8k bytes. Typically used to retrieve a structure.

This function is here for reverse compatibility with releases prior to NetBurner 3.0. Alternatives are to use the 3.0 configuration tree, or the EFFS STD file system.

Returns

Void pointer to the User Parameter area in flash memory

See also

[SaveUserParameters\(\)](#)

14.52.2.3 SaveUserParameters()

```
int SaveUserParameters (  
    void * pCopyFrom,  
    int len)  
#include <system.h>
```

Save data to the on-chip flash memory User Parameter area.

On most platforms the total amount of space is 8k bytes. Typically used to save a structure.

This function is here for reverse compatibility with releases prior to NetBurner 3.0. Alternatives are to use the 3.0 configuration tree, or the EFFS STD file system.

Parameters

<i>pCopyFrom</i>	Pointer to the data to copy and save
<i>len</i>	Length of data

Returns

The number of items copied, or <= 0 on failure

See also

[GetUserParameters\(\)](#)

14.53 TCP

TCP socket programming interface.

Topics

- [TCP Notification System](#)
Advanced callback mechanism for asynchronous TCP socket event notification.
- [TCP Socket Options](#)
Configuration flags for TCP socket behavior.
- [TCP Socket State](#)
TCP connection state machine values (RFC 793)
- [TCP Socket Status](#)
Status codes and error values for TCP socket operations.

Functions

- `int accept (int listening_socket, IPADDR *address, uint16_t *port, uint16_t timeout)`
Accept an incoming connection on a listening socket.
- `int accept (int listening_socket, IPADDR *address, uint16_t *port, TickTimeout &timeout)`
Accept an incoming connection on a listening socket with timeout object.
- `int connect (const IPADDR &ipAddress, uint16_t remotePort, uint32_t timeout)`
Establish a TCP connection to a remote host.
- `int connectvia (const IPADDR &ipAddress, uint16_t remotePort, uint32_t timeout, const IPADDR &localIpAddress)`
Establish a TCP connection using a specific local interface IP address.
- `int connectvia (const IPADDR &ipAddress, uint16_t remotePort, uint32_t timeout, int ifnum)`
Establish a TCP connection using a specific local interface number.
- `int connectwlocal (const IPADDR &ipAddress, uint16_t localPort, uint16_t remotePort, uint32_t timeout, const IPADDR &localIpAddress=IPADDR::NullIP(), int intf=-1)`
Establish a TCP connection with full control over local parameters.
- `int connect (const IPADDR &ipAddress, uint16_t remotePort, TickTimeout &timeout)`
Establish a TCP connection to a remote host with TickTimeout object.
- `int connectvia (const IPADDR &ipAddress, uint16_t remotePort, TickTimeout &timeout, const IPADDR &localIpAddress)`
Establish a TCP connection using a specific local interface IP with TickTimeout.
- `int connectvia (const IPADDR &ipAddress, uint16_t remotePort, TickTimeout &timeout, int ifnum)`
Establish a TCP connection using a specific local interface number with TickTimeout.
- `int connectwlocal (const IPADDR &ipAddress, uint16_t localPort, uint16_t remotePort, TickTimeout timeout, const IPADDR &localIpAddress=IPADDR::NullIP(), int intf=-1)`
Establish a TCP connection with full control over local parameters and TickTimeout.
- `int listen (const IPADDR &addr, uint16_t port, uint8_t maxpend=5)`
Create a listening socket to accept incoming TCP connections.
- `int listenvia (const IPADDR &addr, uint16_t port, int ifn, uint8_t maxpend=5)`
Create a listening socket on a specific network interface.
- `int listenvia (const IPADDR &addr, uint16_t port, const IPADDR &localIpAddress, uint8_t maxpend=5)`
Create a listening socket bound to a specific interface IP address.
- `int NoBlockConnect (const IPADDR &ipAddress, uint16_t remotePort)`
Initiate a non-blocking TCP connection.
- `int NoBlockConnectVia (const IPADDR &ipAddress, uint16_t remotePort, const IPADDR &interfaceIpAddress=IPADDR::NullIP())`
Initiate a non-blocking TCP connection via a specific interface.
- `int NoBlockConnectVia (const IPADDR &ipAddress, uint16_t remotePort, int ifnum)`

- Initiate a non-blocking TCP connection via a specific interface number.*
- `int NoBlockConnectwlocal` (const `IPADDR` &ipAddress, `uint16_t` localPort, `uint16_t` remotePort, `IPADDR` interfaceIpAddress=IPADDR::NullIP(), `int` ifn=-1)
 - Initiate a non-blocking TCP connection with full local parameter control.*
- `int getsocketerror` (`int` fd)
 - Get the error status of a socket.*
- `IPADDR GetSocketRemoteAddr` (`int` fd)
 - Get the remote host IP address for a socket connection.*
- `IPADDR GetSocketLocalAddr` (`int` fd)
 - Get the local interface IP address for a socket connection.*
- `uint16_t GetSocketRemotePort` (`int` fd)
 - Get the remote host port number for a socket connection.*
- `uint16_t GetSocketLocalPort` (`int` fd)
 - Get the local port number for a socket connection.*
- `uint32_t TcpGetLastRxTime` (`int` fd)
 - Get the system tick count when the last packet was received.*
- `void TcpSendKeepAlive` (`int` fd)
 - Send a TCP keepalive probe packet.*
- `uint32_t TcpGetLastRxInterval` (`int` fd)
 - Get the time elapsed since the last received packet.*
- `int GetTcpRtxCount` (`int` fd)
 - Get the number of TCP retransmissions on a socket.*
- `uint8_t SetOutOfOrderBuffers` (`int` fd, `uint8_t` max)
 - Configure the maximum number of out-of-order buffers for a TCP socket.*
- `int setsockopt` (`int` fd, `int` option)
 - Enable TCP socket options.*
- `int clrsockopt` (`int` fd, `int` option)
 - Disable TCP socket options.*
- `int getsockopt` (`int` fd)
 - Query current TCP socket options.*
- `int SetSocketUnackBuffers` (`int` fd, `uint8_t` val)
 - Configure the maximum number of unacknowledged transmit buffers.*
- `int SetSocketRxBuffers` (`int` fd, `int` n)
 - Configure the number of TCP receive buffers.*
- `int SetSocketTxBuffers` (`int` fd, `int` n)
 - Configure the number of TCP transmit buffers.*
- `int abortsocket` (`int` fd)
 - Abort a TCP connection immediately.*
- `int SocketReadWithTimeout` (`int` fd, `char` *buf, `int` nbytes, `uint32_t` timeout)
 - Read from a TCP socket with a timeout.*
- `char SocketPeek` (`int` fd)
 - Peek at the next available byte without removing it from the receive buffer.*
- `void DumpTcpDebug` ()
 - Display current TCP debug information to the console.*
- `void EnableTcpDebug` (`uint16_t` dbFlags)
 - Enable TCP debug output with specified flags.*
- `int TcpGetSocketInterface` (`int` fd)
 - Get the network interface number associated with a TCP socket.*
- `uint8_t TcpGetSocketState` (`int` fd)
 - Get the current TCP state of a socket.*
- `uint16_t TcpGetRxBufferSpaceUsed` (`int` fd)

- Get the number of bytes currently in the receive buffer.*
- uint16_t [TcpGetTxBufferAvailSpace](#) (int fd)
- Get the available space in the transmit buffer.*
- uint16_t [TcpGetTxDataWaiting](#) (int fd)
- Get the number of bytes queued for transmission.*
- BOOL [TcpAllDataAked](#) (int socket)
- Check if all transmitted data has been acknowledged.*
- BOOL [WaitForSocketFlush](#) (int fd, uint32_t ticks)
- Wait for all transmitted data to be acknowledged.*

14.53.1 Detailed Description

TCP socket programming interface.

```
#include< tcp.h>
```

14.53.2 Overview

The NetBurner TCP API provides a robust, Berkeley-sockets-style interface for TCP/IP networking with support for both blocking and non-blocking operations. This API is designed specifically for embedded systems and real-time applications, offering fine-grained control over socket behavior, buffer management, and connection state.

Key Features:

- **Berkeley Sockets Compatibility:** Familiar API for developers with TCP/IP experience
- **IPv4 and IPv6 Support:** Conditional compilation for dual-stack or single-stack configurations
- **Blocking and Non-blocking Modes:** Choose between synchronous and asynchronous operation
- **Advanced Buffer Management:** Configurable TX/RX buffers and out-of-order packet handling
- **Connection Monitoring:** Real-time state tracking, keepalive probes, and connection diagnostics
- **Event-Driven Notifications:** Register callbacks for read/write events
- **Multi-Interface Support:** Control over which network interface to use for connections
- **TLS Ready:** Designed to work seamlessly with NetBurner's TLS implementation

14.53.3 Basic Usage Patterns

The TCP API supports three primary usage patterns:

14.53.3.1 Server Pattern (Listen/Accept)

Create a listening socket to accept incoming client connections:

1. Call [listen\(\)](#) to create a listening socket on a specific port
2. Call [accept\(\)](#) to wait for and accept incoming connections
3. Use [read\(\)](#) / [write\(\)](#) for data transfer on accepted connections
4. Call [close\(\)](#) when done with each connection

14.53.3.2 Client Pattern (Connect)

Establish an outbound connection to a remote server:

1. Call `connect()` to establish a connection to a remote host
2. Use `read()` / `write()` for data transfer
3. Call `close()` when done with the connection

14.53.3.3 Non-Blocking Pattern

For applications requiring immediate return without waiting:

1. Call `NoBlockConnect()` to initiate a connection
2. Poll connection state with `TcpGetSocketState()` or `writeavail()`
3. Once `TCP_STATE_ESTABLISHED`, use `read()` / `write()` for data transfer
4. Register read/write notification callbacks for event-driven operation

14.53.4 Important Concepts

14.53.4.1 Entry Point

NetBurner applications start in `UserMain()`, not `main()`. Always call `init()` at the beginning of `UserMain()` to initialize the network stack before using TCP functions.

14.53.4.2 Timeout Values

Many functions accept timeout parameters in system ticks. Use the `TICKS_PER_SECOND` constant for conversion:

- 5 second timeout: `TICKS_PER_SECOND * 5`
- 30 second timeout: `TICKS_PER_SECOND * 30`
- Wait forever: `0`

14.53.4.3 File Descriptors

TCP sockets are represented by integer file descriptors (fd). Functions that create sockets (`listen()`, `accept()`, `connect()`) return:

- **Positive value:** Valid file descriptor
- **Negative value:** Error code (see [TCP Socket Status](#))

Always check return values:

```
int fd = connect(serverAddr, 80, TICKS_PER_SECOND * 5);
if (fd < 0) {
    printf("Connection failed with error: %d\n", fd);
    return;
}
```

14.53.4.4 Buffer Management

TCP sockets use system buffer pools for transmit and receive operations. The default buffer allocation is conservative (5 buffers per socket) to support many simultaneous connections. For high-throughput applications, you can increase buffer allocations:

- `SetSocketRxBuffers()`: Increase receive buffer count
- `SetSocketTxBuffers()`: Increase transmit buffer count
- `SetSocketUnackBuffers()`: Increase unacknowledged buffer limit
- `SetOutOfOrderBuffers()`: Increase out-of-order packet handling

14.53.4.5 Socket States

TCP connections progress through well-defined states (RFC 793). Use [TcpGetSocketState\(\)](#) to query the current state:

- `TCP_STATE_CLOSED`: No connection
- `TCP_STATE_LISTEN`: Accepting connections
- `TCP_STATE_SYN_SENT`: Connection in progress
- `TCP_STATE_ESTABLISHED`: Connected and ready for data transfer
- `TCP_STATE_CLOSE_WAIT`: Remote closed, local close pending
- `TCP_STATE_CLOSING`: Connection closing

14.53.4.6 Resource Management

Always call [close\(\)](#) on sockets when finished to free system resources. This applies even if the socket enters an error state or `TCP_STATE_CLOSING`. Failing to close sockets will leak file descriptors and network buffers.

14.53.5 Function Categories

14.53.5.1 Connection Management

- [listen\(\)](#) / [listenvia\(\)](#): Create listening sockets for servers
- [accept\(\)](#): Accept incoming connections
- [connect\(\)](#) / [connectvia\(\)](#) / [connectwlocal\(\)](#): Establish outbound connections
- [NoBlockConnect\(\)](#) / [NoBlockConnectVia\(\)](#): Non-blocking connection initiation
- [close\(\)](#): Close connections and free resources
- [abortsocket\(\)](#): Forcefully terminate connection with RST packet

14.53.5.2 Data Transfer

- [read\(\)](#) / [write\(\)](#): Standard I/O operations (in `unistd.h`)
- [readall\(\)](#) / [writeall\(\)](#): Guaranteed complete transfers (in [utils.h](#))
- [SockReadWithTimeout\(\)](#): Read with explicit timeout
- [SocketPeek\(\)](#): Examine next byte without consuming it

14.53.5.3 Socket Query Functions

- [TcpGetSocketState\(\)](#): Get current TCP state
- [GetSocketRemoteAddr\(\)](#) / [GetSocketRemotePort\(\)](#): Remote endpoint information
- [GetSocketLocalAddr\(\)](#) / [GetSocketLocalPort\(\)](#): Local endpoint information
- [TcpGetSocketInterface\(\)](#): Get network interface number
- [dataavail\(\)](#) / [writeavail\(\)](#): Check buffer status (in [utils.h](#))
- [TcpGetRxBufferSpaceUsed\(\)](#): Bytes in receive buffer
- [TcpGetTxBufferAvailSpace\(\)](#): Available transmit space
- [TcpGetTxDataWaiting\(\)](#): Bytes queued for transmission

14.53.5.4 Configuration Functions

- [setsockopt\(\)](#) / [clrsockopt\(\)](#) / [getsockopt\(\)](#): Socket options (Nagle, PUSH)
- [SetSocketRxBuffers\(\)](#): Configure receive buffer count
- [SetSocketTxBuffers\(\)](#): Configure transmit buffer count
- [SetSocketUnackBuffers\(\)](#): Configure unacknowledged buffer limit
- [SetOutOfOrderBuffers\(\)](#): Configure out-of-order packet handling

14.53.5.5 Monitoring and Diagnostics

- [TcpGetLastRxTime\(\)](#) / [TcpGetLastRxInterval\(\)](#): Track connection activity
- [TcpSendKeepAlive\(\)](#): Send keepalive probe
- [GetTcpRtxCount\(\)](#): Get retransmission count
- [TcpAllDataAked\(\)](#): Check if all data acknowledged
- [WaitForSocketFlush\(\)](#): Wait for acknowledgments
- [getsocketerror\(\)](#): Get error code for socket

14.53.5.6 Event Notification

- [RegisterTCPReadNotify\(\)](#): Register callback for read events
- [RegisterTCPWriteNotify\(\)](#): Register callback for write events

14.53.5.7 Throughput Monitoring (#define TCP_THROUGHPUT_INFO)

- [TCPGetPayloadBytesSent\(\)](#): Payload bytes transmitted
- [TCPGetPayloadBytesReceived\(\)](#): Payload bytes received
- [TCPGetTotalBytesSent\(\)](#): Total bytes sent (including headers)
- [TCPGetTotalBytesReceived\(\)](#): Total bytes received (including headers)

14.53.6 Error Handling

TCP functions use negative return values to indicate errors. Always check return values and handle errors appropriately:

Common error codes (see [TCP Socket Status](#)):

- [TCP_ERR_TIMEOUT](#) (-1): Operation timed out
- [TCP_ERR_NOCON](#) (-2): No active connection
- [TCP_ERR_CLOSING](#) (-3): Socket is closing
- [TCP_ERR_NOSUCH_SOCKET](#) (-4): Invalid socket descriptor
- [TCP_ERR_NONE_AVAIL](#) (-5): No sockets available
- [TCP_ERR_CON_RESET](#) (-6): Connection reset by peer
- [TCP_ERR_CON_ABORT](#) (-7): Connection aborted

For detailed return code information, causes and solutions, see: [Programmers Guide TCP Error Codes](#)

14.53.7 Advanced Features

14.53.7.1 Multi-Interface Control

When multiple network interfaces are available, you can control which interface is used for connections:

- `connectvia(addr, port, timeout, localIpAddress)`: Use interface with specific IP
- `connectvia(addr, port, timeout, ifnum)`: Use interface by number
- `listenvia()`: Listen on specific interface

14.53.7.2 Keepalive Implementation

Detect stale connections using the keepalive mechanism:

1. Record lastRxTime with `TcpGetLastRxTime()`
2. Send keepalive probe with `TcpSendKeepAlive()`
3. Wait for response (2-3 seconds)
4. Check if lastRxTime changed
5. If unchanged, connection may be lost

14.53.7.3 Nagle Algorithm Control

The Nagle algorithm buffers small packets to improve efficiency but can add latency for interactive applications. Use `SO_NONAGLE` option to disable:

```
setsockopt(fd, SO_NONAGLE); // Disable Nagle, send immediately
```

14.53.7.4 Event-Driven Programming

For event-driven architectures, register notification callbacks instead of polling:

```
void OnDataReady(int fd) {  
    char buffer[1024];  
    int n = read(fd, buffer, sizeof(buffer));  
    // Process data...  
}
```

```
RegisterTCPReadNotify(fd, OnDataReady);
```

14.53.8 Best Practices

14.53.8.1 DO:

- Always call `init()` before using TCP functions
- Check return values from all TCP functions
- Use `TICKS_PER_SECOND` for timeout calculations
- Call `close()` on all sockets when finished
- Use `readall()` / `writeall()` for complete data transfers
- Implement timeouts for all blocking operations
- Use notification callbacks for event-driven designs
- Increase buffer allocations for high-throughput applications

14.53.8.2 DON'T:

- Don't forget to close sockets (causes resource leaks)
- Don't ignore negative return values (indicates errors)
- Don't assume `read()` / `write()` transfer all requested bytes
- Don't specify local port numbers unless absolutely necessary
- Don't use fixed buffer sizes without checking actual data size
- Don't block indefinitely without user approval
- Don't call `TcpGetLastRxTime()` more than once per second

14.53.9 IPv6 Support

The TCP API conditionally compiles for IPv4-only (IPV4ONLY) or dual-stack IPv6 support. When IPv6 is enabled:

- IPADDR type represents both IPv4 and IPv6 addresses
- Functions with '6' suffix operate on IPv6: `connect6()`, `listen6()`
- Functions with '4' suffix operate on IPv4: `connect4()`, `listen4()`
- Generic functions (`connect`, `listen`) use the default IP version

14.53.10 Performance Tuning

14.53.10.1 Bulk Data Transfer

For maximum throughput when transferring large amounts of data:

- Increase receive buffers: `SetSocketRxBuffers(fd, 43) // Maximum effective`
- Increase transmit buffers: `SetSocketTxBuffers(fd, 20)`
- Increase unack buffers: `SetSocketUnackBuffers(fd, 20)`
- Disable Nagle: `setsockopt(fd, SO_NONAGLE)`
- Use large `write()` calls (multiple KB per write)

14.53.10.2 Low Latency

For interactive or real-time applications requiring minimal latency:

- Disable Nagle: `setsockopt(fd, SO_NONAGLE)`
- Use small buffers (default 5 is fine)
- Use non-blocking mode with notification callbacks
- Process data immediately upon arrival

14.53.10.3 Many Simultaneous Connections

For servers handling many concurrent connections:

- Keep default buffer allocations (5 per socket)
- Use event-driven design with notification callbacks
- Implement connection timeouts to reclaim stale sockets
- Monitor system resource usage with `EnableSystemDiagnostics()`
- Consider increasing `SYSTEM_TCP_SOCKETS` in `predef.h`

14.53.11 Thread Safety

The NetBurner TCP implementation is designed for use in a multi-threaded RTOS environment. However, individual socket operations are not automatically thread-safe:

- **Single socket, single thread:** No synchronization needed
- **Single socket, multiple threads:** Application must provide synchronization
- **Multiple sockets, multiple threads:** Each socket can be used independently
- **Listening socket:** Can be shared by multiple threads calling [accept\(\)](#)
- **Notification callbacks:** Execute in TCP stack context, must be thread-safe

14.53.12 Related APIs

The TCP API works in conjunction with other NetBurner APIs:

- **utils.h:** [readall\(\)](#), [writeall\(\)](#), [dataavail\(\)](#), [writeavail\(\)](#)
- **unistd.h:** [read\(\)](#), [write\(\)](#), [close\(\)](#)
- **ip.h:** IPADDR class, [AsciiToIp\(\)](#), IP address utilities
- **dns.h:** [GetHostByName\(\)](#) for hostname resolution
- **ssl.h:** TLS encryption for secure connections
- **http.h:** High-level HTTP server built on TCP
- **websocket.h:** WebSocket protocol over TCP

14.53.13 Further Reading

- **NetBurner Documentation:** Comprehensive guides at docs.netburner.com
- **TCP/IP RFCs:** RFC 793 (TCP), RFC 1122 (Requirements)
- **Code Examples:** See NetBurner example applications in the SDK
- **API Reference:** Individual function documentation in this header

See also

[TCP](#)

[TCP Socket Status](#)

[TCP Socket Options](#)

[TCP Socket State](#)

[TCP Notification System](#)

14.53.14 Function Documentation

14.53.14.1 abortsocket()

```
int abortsocket (
    int fd)
#include <tcp.h>
```

Abort a TCP connection immediately.

Forcefully terminates a TCP connection by sending a RST (reset) packet to the remote host. Unlike a normal [close\(\)](#), this does not perform a graceful shutdown and any pending data will be lost. Use this only when you need to terminate a connection immediately without the normal shutdown handshake.

Parameters

<i>fd</i>	Socket file descriptor

Returns

Status code: [TCP Socket Status](#)

See also

[close\(\)](#)

Expand for Example Usage

Examples

Basic Usage - Aborting a Connection

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

// Something went wrong, abort immediately
int result = abortsocket(fd);
if (result == TCP_ERR_NONE) {
    printf("Connection aborted\n");
}
```

Timeout Handler - Aborting Unresponsive Connection

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

uint32_t startTime = Secs;
char response[100];

while (Secs - startTime < 30) { // 30 second timeout
    int n = read(fd, response, sizeof(response));
    if (n > 0) {
        // Process response
        break;
    }
    OSTimeDly(TICKS_PER_SECOND);
}

if (Secs - startTime >= 30) {
    printf("Connection timeout, aborting...\n");
    abortsocket(fd); // Force immediate termination
} else {
    close(fd); // Normal graceful close
}
```

Error Recovery - Abort vs Close

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

if (write(fd, data, dataLen) < 0) {
    // Critical error occurred
    printf("Write failed, aborting connection\n");
    abortsocket(fd); // RST - immediate termination, no graceful shutdown
} else {
    // Normal operation complete
    close(fd); // FIN - graceful shutdown with handshake
}
```

Protocol Violation - Immediate Abort

```
void HandleConnection(int clientFd) {
    char buffer[256];
    int n = read(clientFd, buffer, sizeof(buffer));

    // Check for protocol violation
    if (n > 0 && !IsValidProtocolHeader(buffer)) {
        printf("Invalid protocol header detected\n");
        abortsocket(clientFd); // Immediate abort for security
        return;
    }

    // Normal processing...
    ProcessRequest(buffer, n);
    close(clientFd); // Graceful close for valid connections
}
```

14.53.14.2 accept() [1/2]

```
int accept (  
    int listening_socket,  
    IPADDR * address,  
    uint16_t * port,  
    TickTimeout & timeout)  
#include <tcp.h>
```

Accept an incoming connection on a listening socket with timeout object.

Accepts a pending connection request from the listen queue using a TickTimeout object for more flexible timeout management. The listening socket must have been created with [listen\(\)](#) before calling this function. Each successful accept frees one slot in the listen queue for additional incoming connections.

This overload of [accept\(\)](#) uses a TickTimeout object instead of a simple tick count, providing more sophisticated timeout behavior. The TickTimeout object can track elapsed time across multiple operations and provides precise timeout management for complex scenarios.

Parameters

<i>listening_socket</i>	File descriptor of the listening socket (obtained from listen())
<i>address</i>	Pointer to IPADDR to store the remote client's IP address. Pass NULL if you don't need the client's IP address
<i>port</i>	Pointer to uint16_t to store the remote client's port number. Pass NULL if you don't need the client's port
<i>timeout</i>	Reference to TickTimeout object specifying the maximum wait time. The TickTimeout tracks elapsed time and can be reused across multiple operations. Create with TickTimeout(ticks) where ticks is the total timeout period in system ticks

Returns

Positive file descriptor on success (use this for read/write with the client), negative [TCP Socket Status](#) error code on failure:

- TCP_ERR_TIMEOUT: No connection within timeout period
- TCP_ERR_NOSUCH_SOCKET: Invalid listening socket
- TCP_ERR_NONE_AVAIL: No available sockets in system pool

Notes:

- The TickTimeout object is updated by this function to reflect the remaining time. This allows multiple operations to share a common timeout budget.
- Always call [close\(\)](#) on the returned file descriptor when finished with the client connection to free system resources.
- TickTimeout provides more accurate timeout tracking than simple tick counts, especially when performing multiple sequential operations with a shared deadline.

Warning

The timeout parameter is passed by reference and will be modified by this function. If you need to preserve the original timeout value, create a copy.

Expand for Example Usage

Examples

Basic Accept with TickTimeout

```

#include <tcp.h>
#include <init.h>

void ServerWithTickTimeout(void *pd) {
    init();

    int fdListen = listen(INADDR_ANY, 8080, 5);
    printf("Server listening on port 8080\n");

    // Create TickTimeout object for 30 second timeout
    TickTimeout timeout(TICKS_PER_SECOND * 30);

    IPADDR clientAddr;
    uint16_t clientPort;

    int fdClient = accept(fdListen, &clientAddr, &clientPort, timeout);

    if (fdClient > 0) {
        printf("Client connected from %s:%d\n",
            clientAddr.print(), clientPort);

        // Check remaining time in timeout
        printf("Remaining timeout: %lu ticks\n", timeout.GetTimeLeft());

        // Use connection...
        close(fdClient);
    } else if (fdClient == TCP_ERR_TIMEOUT) {
        printf("No connection received within 30 seconds\n");
    } else {
        printf("Accept failed: %d\n", fdClient);
    }

    close(fdListen);
}

```

Shared Timeout Across Multiple Operations

```

#include <tcp.h>
#include <init.h>

// Accept connection and perform handshake within total time budget
bool AcceptAndHandshake(int fdListen, uint32_t totalTimeoutTicks) {
    // Create single TickTimeout for all operations
    TickTimeout timeout(totalTimeoutTicks);

    printf("Total timeout budget: %lu seconds\n", totalTimeoutTicks / TICKS_PER_SECOND);

    // Step 1: Accept connection (uses part of timeout)
    IPADDR clientAddr;
    uint16_t clientPort;
    int fdClient = accept(fdListen, &clientAddr, &clientPort, timeout);

    if (fdClient < 0) {
        printf("Accept failed: %d\n", fdClient);
        return false;
    }

    printf("Client connected, remaining time: %lu ticks\n", timeout.GetTimeLeft());

    // Step 2: Read handshake message (uses remaining timeout)
    char handshake[64];
    int n = ReadWithTimeout(fdClient, handshake, sizeof(handshake), timeout);

    if (n <= 0) {
        printf("Handshake read failed: %d\n", n);
        close(fdClient);
        return false;
    }

    printf("Handshake received, remaining time: %lu ticks\n", timeout.GetTimeLeft());

    // Step 3: Send response (uses remaining timeout)
    const char *response = "OK\r\n";
    n = WriteWithTimeout(fdClient, response, strlen(response), timeout);

    if (n <= 0) {
        printf("Response write failed: %d\n", n);
        close(fdClient);
        return false;
    }

    printf("Handshake complete, time used: %lu ticks\n", totalTimeoutTicks - timeout.GetTimeLeft());

    close(fdClient);
    return true;
}

```

```

}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int fdListen = listen(INADDR_ANY, 8080, 5);

    // Give entire handshake process 10 seconds to complete
    bool success = AcceptAndHandshake(fdListen, TICKS_PER_SECOND * 10);

    printf("Handshake %s\n", success ? "succeeded" : "failed");
    close(fdListen);

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Multiple Accept Attempts with Shared Timeout

```

#include <tcp.h>
#include <init.h>

// Try to accept connections from multiple clients within time limit
int AcceptMultipleClients(int fdListen, int maxClients, uint32_t totalTimeout) {
    TickTimeout timeout(totalTimeout);
    int clientFds[10];
    int acceptedCount = 0;

    printf("Accepting up to %d clients within %lu seconds\n",
           maxClients, totalTimeout / TICKS_PER_SECOND);

    while (acceptedCount < maxClients && !timeout.Expired()) {
        IPADDR clientAddr;
        uint16_t clientPort;

        printf("Waiting for client %d (timeout left: %lu ticks)...\n",
               acceptedCount + 1, timeout.GetTimeLeft());

        int fdClient = accept(fdListen, &clientAddr, &clientPort, timeout);

        if (fdClient > 0) {
            clientFds[acceptedCount] = fdClient;
            acceptedCount++;
            printf(" Client %d connected: %s:%d\n", acceptedCount, clientAddr.print(), clientPort);
        } else if (fdClient == TCP_ERR_TIMEOUT) {
            printf(" Timeout reached\n");
            break;
        } else {
            printf(" Accept error: %d\n", fdClient);
            break;
        }
    }

    printf("Accepted %d clients\n", acceptedCount);

    // Broadcast message to all clients
    char msg[64];
    snprintf(msg, sizeof(msg), "Group of %d clients ready\r\n", acceptedCount);

    for (int i = 0; i < acceptedCount; i++) {
        write(clientFds[i], msg, strlen(msg));
        close(clientFds[i]);
    }

    return acceptedCount;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int fdListen = listen(INADDR_ANY, 9000, 5);

    // Accept up to 5 clients within 60 seconds
    int count = AcceptMultipleClients(fdListen, 5, TICKS_PER_SECOND * 60);

    printf("Session complete with %d clients\n", count);
    close(fdListen);

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Polling Loop with Decaying Timeout

```

#include <tcp.h>
#include <init.h>
#include <nbrtos.h>

void PollingServerWithTimeout(void *pd) {
    init();

    int fdListen = listen(INADDR_ANY, 8080, 5);
    printf("Server listening on port 8080\n");

    // Give server 5 minutes to handle all connections
    TickTimeout serverTimeout(TICKS_PER_SECOND * 300);
    int connectionCount = 0;

    while (!serverTimeout.Expired()) {
        // Calculate time spent so far
        uint32_t elapsed = serverTimeout.GetElapsedTime();
        uint32_t remaining = serverTimeout.GetTimeLeft();

        printf("\n[%d connections] Elapsed: %lu s, Remaining: %lu s\n",
            connectionCount,
            elapsed / TICKS_PER_SECOND,
            remaining / TICKS_PER_SECOND);

        // Create short-term timeout for this accept attempt
        TickTimeout acceptTimeout(TICKS_PER_SECOND * 5);

        IPADDR clientAddr;
        uint16_t clientPort;
        int fdClient = accept(fdListen, &clientAddr, &clientPort, acceptTimeout);

        if (fdClient > 0) {
            connectionCount++;
            printf("Client %d connected: %s:%d\n", connectionCount, clientAddr.print(), clientPort);

            // Quick interaction
            char buffer[256];
            int n = read(fdClient, buffer, sizeof(buffer));
            if (n > 0) {
                write(fdClient, buffer, n);
            }

            close(fdClient);
        } else if (fdClient == TCP_ERR_TIMEOUT) {
            printf("No connection in last 5 seconds\n");
        } else {
            printf("Accept error: %d\n", fdClient);
        }

        // Check if we should continue
        if (serverTimeout.GetTimeLeft() < (TICKS_PER_SECOND * 10)) {
            printf("Less than 10 seconds remaining - final call\n");
        }
    }

    printf("Server timeout reached after %d connections\n", connectionCount);
    close(fdListen);
}

```

Timeout Monitoring and Adjustment

```

#include <tcp.h>
#include <init.h>

void AdaptiveTimeoutServer(void *pd) {
    init();

    int fdListen = listen(INADDR_ANY, 7777, 5);

    while (1) {
        // Start with 30 second timeout
        TickTimeout timeout(TICKS_PER_SECOND * 30);

        IPADDR clientAddr;
        uint16_t clientPort;

        uint32_t startTicks = TimeTick;
        int fdClient = accept(fdListen, &clientAddr, &clientPort, timeout);
        uint32_t acceptDuration = TimeTick - startTicks;

        if (fdClient > 0) {
            printf("Client connected after %lu ticks (%lu ms)\n",
                acceptDuration,
                acceptDuration * 1000 / TICKS_PER_SECOND);
        }
    }
}

```

```

        printf("Timeout had %lu ticks remaining\n", timeout.GetTimeLeft());

        // Adjust future timeout based on how long this took
        if (acceptDuration < (TICKS_PER_SECOND * 5)) {
            printf("Fast connection - might reduce future timeout\n");
        } else if (acceptDuration > (TICKS_PER_SECOND * 20)) {
            printf("Slow connection - might increase future timeout\n");
        }

        close(fdClient);

    } else if (fdClient == TCP_ERR_TIMEOUT) {
        printf("Accept timeout - timeout object expired: %s\n", timeout.Expired() ? "yes" :
"no");
        printf("Time left: %lu ticks\n", timeout.GetTimeLeft());
    }
}
}

```

Reusing TickTimeout Object Across Calls

```

#include <tcp.h>
#include <init.h>

void ReuseTimeoutExample(void *pd) {
    init();

    int fdListen = listen(INADDR_ANY, 8080, 5);

    while (1) {
        // Create fresh timeout for each iteration
        TickTimeout timeout(TICKS_PER_SECOND * 20);

        printf("\n=== New accept cycle ===\n");
        printf("Initial timeout: %lu ticks\n", timeout.GetTimeLeft());
        printf("Expired: %s\n", timeout.Expired() ? "yes" : "no");

        IPADDR clientAddr;
        uint16_t clientPort;
        int fdClient = accept(fdListen, &clientAddr, &clientPort, timeout);

        if (fdClient > 0) {
            printf("Connection accepted\n");
            printf("After accept - remaining: %lu ticks\n", timeout.GetTimeLeft());
            printf("After accept - elapsed: %lu ticks\n", timeout.GetElapsedTime());
            printf("After accept - expired: %s\n", timeout.Expired() ? "yes" : "no");

            close(fdClient);
        } else {
            printf("Accept failed or timeout: %d\n", fdClient);
            printf("Final timeout state - remaining: %lu ticks\n", timeout.GetTimeLeft());
            printf("Final timeout state - expired: %s\n", timeout.Expired() ? "yes" : "no");
        }

        OSTimeDly(TICKS_PER_SECOND); // Brief pause between cycles
    }
}

```

Connection Rate Limiting

```

#include <tcp.h>
#include <init.h>
#include <nbrtos.h>

// Accept connections with rate limiting
void RateLimitedServer(void *pd) {
    init();

    int fdListen = listen(INADDR_ANY, 8080, 5);

    const int MAX_CONNECTIONS_PER_MINUTE = 10;
    int connectionsThisMinute = 0;

    // One minute window
    TickTimeout windowTimeout(TICKS_PER_SECOND * 60);

    while (1) {
        // Reset counter when window expires
        if (windowTimeout.Expired()) {
            printf("Minute window expired - accepted %d connections\n", connectionsThisMinute);
            connectionsThisMinute = 0;
            windowTimeout.Reset(TICKS_PER_SECOND * 60);
        }

        // Check if we've hit the rate limit
    }
}

```

```

    if (connectionsThisMinute >= MAX_CONNECTIONS_PER_MINUTE) {
        uint32_t waitTime = windowTimeout.GetTimeLeft();
        printf("Rate limit reached - waiting %lu seconds\n", waitTime / TICKS_PER_SECOND);
        OSTimeDly(waitTime);
        continue;
    }

    // Accept with short timeout
    TickTimeout acceptTimeout(TICKS_PER_SECOND * 2);
    IPADDR clientAddr;
    uint16_t clientPort;

    int fdClient = accept(fdListen, &clientAddr, &clientPort, acceptTimeout);

    if (fdClient > 0) {
        connectionsThisMinute++;
        printf("Connection %d/%d this minute from %s:%d\n",
            connectionsThisMinute, MAX_CONNECTIONS_PER_MINUTE,
            clientAddr.print(), clientPort);

        // Handle connection...
        close(fdClient);
    }
}
}

```

Graceful Shutdown with Timeout

```

#include <tcp.h>
#include <init.h>

volatile bool g_shutdownRequested = false;

void GracefulShutdownServer(void *pd) {
    init();

    int fdListen = listen(INADDR_ANY, 8080, 5);
    printf("Server started - will shutdown gracefully when requested\n");

    while (!g_shutdownRequested) {
        // Use short timeout so we check shutdown flag frequently
        TickTimeout timeout(TICKS_PER_SECOND * 2);

        IPADDR clientAddr;
        uint16_t clientPort;
        int fdClient = accept(fdListen, &clientAddr, &clientPort, timeout);

        if (fdClient > 0) {
            printf("Client connected: %s:%d\n", clientAddr.print(), clientPort);

            // Check shutdown flag before processing
            if (g_shutdownRequested) {
                printf("Shutdown requested - rejecting client\n");
                const char *msg = "Server shutting down\r\n";
                write(fdClient, msg, strlen(msg));
                close(fdClient);
                break;
            }

            // Normal client handling...
            close(fdClient);
        } else if (fdClient == TCP_ERR_TIMEOUT) {
            // Timeout is normal - gives us a chance to check shutdown flag
            continue;
        } else {
            printf("Accept error: %d\n", fdClient);
        }
    }

    close(fdListen);
    printf("Server shutdown complete\n");
}

```

See also

[listen\(\)](#), [close\(\)](#), [accept\(int, IPADDR*, uint16_t*, uint16_t\)](#), [TickTimeout](#)

14.53.14.3 accept() [2/2]

```

int accept (
    int listening_socket,

```

```
IPADDR * address,
uint16_t * port,
uint16_t timeout)
```

```
#include <tcp.h>
```

Accept an incoming connection on a listening socket.

Accepts a pending connection request from the listen queue. The listening socket must have been created with [listen\(\)](#) before calling this function. Each successful accept frees one slot in the listen queue for additional incoming connections.

This function blocks until either a connection arrives or the timeout expires. When a client connects, [accept\(\)](#) returns a new socket file descriptor for communicating with that specific client. The original listening socket remains open and continues to accept new connections.

Parameters

<i>listening_socket</i>	File descriptor of the listening socket (obtained from listen())
<i>address</i>	Pointer to IPADDR to store the remote client's IP address. Pass NULL if you don't need the client's IP address
<i>port</i>	Pointer to uint16_t to store the remote client's port number. Pass NULL if you don't need the client's port
<i>timeout</i>	Maximum wait time in system ticks: • 0 = wait indefinitely until a connection arrives • Use TICKS_PER_SECOND * n for n second timeout • Example: TICKS_PER_SECOND * 5 for 5 second timeout • Example: TICKS_PER_SECOND * 30 for 30 second timeout

Returns

Positive file descriptor on success (use this for read/write with the client), negative [TCP Socket Status](#) error code on failure:

- TCP_ERR_TIMEOUT: No connection within timeout period
- TCP_ERR_NOSUCH_SOCKET: Invalid listening socket
- TCP_ERR_NONE_AVAIL: No available sockets in system pool

Notes:

- Always call [close\(\)](#) on the returned file descriptor when finished with the client connection to free system resources.
- The listening socket is not consumed by [accept\(\)](#) and can be used repeatedly to accept multiple client connections.
- If you pass NULL for both address and port parameters, the function still works but you won't know which client connected.

Warning

- When timeout is 0 (infinite wait), the calling task will block indefinitely until a connection arrives. Use finite timeouts in production code to prevent hanging tasks.
- If the system runs out of available sockets (TCP_ERR_NONE_AVAIL), you must close existing sockets before new connections can be accepted.

Expand for Example Usage

Examples

Basic Accept with Infinite Wait

```
#include <tcp.h>
#include <init.h>

void ServerTask(void *pd) {
    init();

    // Create listening socket on port 8080
    int fdListen = listen(INADDR_ANY, 8080, 5);
    if (fdListen < 0) {
        printf("Failed to create listening socket\n");
        return;
    }

    printf("Server listening on port 8080\n");

    // Accept connections (wait indefinitely)
    IPADDR clientAddr;
    uint16_t clientPort;
    int fdClient = accept(fdListen, &clientAddr, &clientPort, 0);

    if (fdClient > 0) {
        printf("Client connected from %s:%d\n",
            clientAddr.print(), clientPort);

        // Communicate with client...
        char buffer[256];
        int n = read(fdClient, buffer, sizeof(buffer));

        close(fdClient);
        printf("Client disconnected\n");
    }

    close(fdListen);
}
```

Accept with Timeout

```
#include <tcp.h>
#include <init.h>

void ServerWithTimeout(void *pd) {
    init();

    int fdListen = listen(INADDR_ANY, 9000, 5);
    printf("Waiting for connection (30 second timeout)...\n");

    IPADDR clientAddr;
    uint16_t clientPort;

    // Wait up to 30 seconds for a connection
    int fdClient = accept(fdListen, &clientAddr, &clientPort,
        TICKS_PER_SECOND * 30);

    if (fdClient == TCP_ERR_TIMEOUT) {
        printf("No connection received within timeout period\n");
    } else if (fdClient < 0) {
        printf("Accept failed with error: %d\n", fdClient);
    } else {
        printf("Client connected: %s:%d\n",
            clientAddr.print(), clientPort);

        // Handle client connection...
        close(fdClient);
    }

    close(fdListen);
}
```

Accept with Error Handling

```
#include <tcp.h>
#include <init.h>

void RobustAccept(int fdListen) {
    IPADDR clientAddr;
    uint16_t clientPort;

    int fdClient = accept(fdListen, &clientAddr, &clientPort,
        TICKS_PER_SECOND * 60);

    // Comprehensive error handling
    if (fdClient > 0) {
        printf("Connection accepted from %s:%d (fd=%d)\n",
            clientAddr.print(), clientPort, fdClient);
    }
}
```

```

        // Use connection...
        close(fdClient);

    } else {
        // Handle specific error codes
        switch (fdClient) {
            case TCP_ERR_TIMEOUT:
                printf("Accept timeout - no client connected within 60 seconds\n");
                break;

            case TCP_ERR_NOSUCH_SOCKET:
                printf("Invalid listening socket - was it closed?\n");
                break;

            case TCP_ERR_NONE_AVAIL:
                printf("No sockets available - too many open connections!\n");
                printf("Close unused sockets to free resources\n");
                break;

            case TCP_ERR_NOCON:
                printf("No connection on listening socket\n");
                break;

            default:
                printf("Accept failed with error code: %d\n", fdClient);
                break;
        }
    }
}

```

See also

[listen\(\)](#), [close\(\)](#), [connect\(\)](#), [read\(\)](#), [write\(\)](#), [writeall\(\)](#), [readall\(\)](#)

14.53.14.4 clrsockoption()

```

int clrsockoption (
    int fd,
    int option)
#include <tcp.h>

```

Disable TCP socket options.

Clears one or more option flags for the specified socket. Socket options control various aspects of TCP behavior such as the Nagle algorithm and PUSH flag handling. Multiple options can be combined using bitwise OR.

Parameters

<i>fd</i>	Socket file descriptor
<i>option</i>	Socket option flags to disable: TCP Socket Options

Returns

Bitmask of all currently enabled options for the socket, or negative value on error

Notes:

- This function does not affect options that are not specified in the parameter. Only the specified options are cleared.
- After clearing options, the return value shows the remaining enabled options.
- Multiple options can be cleared simultaneously using bitwise OR: `clrsockoption(fd, SO_NODELAY | SO_PUSH);`

See also

[setsockopt\(\)](#), [getsockopt\(\)](#), [SetSocketUnackBuffers\(\)](#), [SetSocketRxBuffers\(\)](#), [SetSocketTxBuffers\(\)](#)

Expand for Example Usage

Examples

Re-enable Nagle Algorithm

```
#include <tcp.h>
#include <nbrtos.h>

void BulkTransferWithNagle(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Initially disable Nagle for handshake
    setsockopt(fd, SO_NODELAY);
    printf("SO_NODELAY enabled for handshake\r\n");

    // Send handshake
    const char *handshake = "READY\r\n";
    write(fd, handshake, strlen(handshake));

    // Read acknowledgment
    char ack[64];
    read(fd, ack, sizeof(ack) - 1, TICKS_PER_SECOND * 5);

    // Now switch to bulk transfer mode - re-enable Nagle for efficiency
    int result = clrsockopt(fd, SO_NODELAY);

    if (result >= 0)
    {
        printf("SO_NODELAY disabled for bulk transfer\r\n");
        printf("Nagle algorithm now active - current options: 0x%X\r\n", result);
    }

    // Send large amount of data efficiently with Nagle
    char buffer[512];
    for (int i = 0; i < 1000; i++)
    {
        snprintf(buffer, sizeof(buffer), "Data packet %d\r\n", i);
        write(fd, buffer, strlen(buffer));
    }

    printf("Bulk transfer complete with Nagle optimization\r\n");

    close(fd);
}
```

Dynamic Option Management

```
#include <tcp.h>
#include <nbrtos.h>

void AdaptiveSocketOptions(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Start with low-latency configuration
    setsockopt(fd, SO_NODELAY | SO_PUSH);
    printf("Initial config: Low-latency mode\r\n");

    // Send interactive commands
    const char *cmd1 = "STATUS\r\n";
    write(fd, cmd1, strlen(cmd1));

    OSTimeDly(TICKS_PER_SECOND);
}
```

```

// Check if we're about to do bulk transfer
const char *cmd2 = "START_BULK\r\n";
write(fd, cmd2, strlen(cmd2));

// Switch to bulk transfer mode
printf("Switching to bulk transfer mode...\r\n");

// Clear low-latency options
int result = clrsockoption(fd, SO_NODELAY | SO_PUSH);

printf("Options cleared - remaining: 0x%X\r\n", result);
printf("Nagle algorithm and normal buffering now active\r\n");

// Perform bulk transfer with default TCP behavior
char data[1024];
for (int i = 0; i < 500; i++)
{
    snprintf(data, sizeof(data), "BULK_DATA_%d", i);
    write(fd, data, strlen(data));
}

close(fd);
}

```

Selective Option Clearing

```

#include <tcp.h>
#include <nbrtos.h>

void SelectiveOptionControl(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Enable multiple options
    int options = SO_NODELAY | SO_PUSH | SO_KEEPAIVE;
    int result = setsockoption(fd, options);

    printf("Initial options: 0x%X\r\n", result);
    printf(" SO_NODELAY: %s\r\n", (result & SO_NODELAY) ? "ON" : "OFF");
    printf(" SO_PUSH: %s\r\n", (result & SO_PUSH) ? "ON" : "OFF");
    printf(" SO_KEEPAIVE: %s\r\n", (result & SO_KEEPAIVE) ? "ON" : "OFF");

    OSTimeDly(TICKS_PER_SECOND * 2);

    // Clear only SO_NODELAY, keep others
    result = clrsockoption(fd, SO_NODELAY);

    printf("\r\nAfter clearing SO_NODELAY: 0x%X\r\n", result);
    printf(" SO_NODELAY: %s\r\n", (result & SO_NODELAY) ? "ON" : "OFF");
    printf(" SO_PUSH: %s\r\n", (result & SO_PUSH) ? "ON" : "OFF");
    printf(" SO_KEEPAIVE: %s\r\n", (result & SO_KEEPAIVE) ? "ON" : "OFF");

    close(fd);
}

```

Reset All Options to Default

```

#include <tcp.h>
#include <nbrtos.h>

void ResetSocketOptions(int fd)
{
    // Get current options
    int currentOptions = getsockoption(fd);

    if (currentOptions < 0)
    {
        printf("Failed to get current options\r\n");
        return;
    }

    printf("Current options: 0x%X\r\n", currentOptions);

    if (currentOptions != 0)
    {
        // Clear all options
        int result = clrsockoption(fd, currentOptions);
    }
}

```

```

        printf("All options cleared - remaining: 0x%X\r\n", result);
        printf("Socket reset to default behavior\r\n");
    }
    else
    {
        printf("No options to clear - already at defaults\r\n");
    }
}

void OptionResetExample(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Set various options
    setsockopt(fd, SO_NODELAY | SO_PUSH);
    printf("Options configured\r\n");

    // Do some work...
    const char *msg = "Test\r\n";
    write(fd, msg, strlen(msg));

    // Reset to defaults before closing
    printf("\r\nResetting socket options...\r\n");
    ResetSocketOptions(fd);

    close(fd);
}

```

14.53.14.5 connect() [1/2]

```

int connect (
    const IPADDR & ipAddress,
    uint16_t remotePort,
    TickTimeout & timeout) [inline]
#include <tcp.h>

```

Establish a TCP connection to a remote host with TickTimeout object.

Creates a new TCP socket and attempts to connect to the specified remote IP address and port using a TickTimeout object for timeout management. This overload provides more sophisticated timeout tracking compared to the simple tick count version, allowing the timeout to be shared across multiple operations and providing precise elapsed time tracking.

The function automatically selects the local interface based on the routing table and assigns an available local port. The TickTimeout object is updated to reflect the remaining time after the connection attempt, enabling accurate timeout management across sequential operations.

Parameters

<i>ipAddress</i>	IP address of the remote host. Can be created using AsciiToIp() or obtained from DNS resolution. Example: AsciiToIp("192.168.1.100")
<i>remotePort</i>	Port number on the remote host (1-65535). Common ports include: • 80 for HTTP • 443 for HTTPS • 23 for Telnet • 22 for SSH
<i>timeout</i>	Reference to TickTimeout object specifying the maximum wait time. The TickTimeout tracks elapsed time and can be reused across multiple operations. Create with TickTimeout(ticks) where ticks is the total timeout period in system ticks. The object is updated by this function to reflect remaining time

Returns

Positive file descriptor on success (use for read/write operations), negative [TCP Socket Status](#) error code on failure:

- `TCP_ERR_TIMEOUT`: Connection not established within timeout period
- `TCP_ERR_NONE_AVAIL`: No available sockets in system pool
- `TCP_ERR_CON_RESET`: Connection was reset during establishment
- `TCP_ERR_CON_ABORT`: Connection was aborted

Notes:

- The `TickTimeout` object is updated by this function to reflect the remaining time. This allows multiple operations to share a common timeout budget.
- The local port number is automatically assigned from the ephemeral port range.
- The local network interface is automatically selected based on the routing table for the destination IP address.
- Always call [close\(\)](#) on the returned file descriptor when finished to free system resources, even if read/write operations fail.
- This function blocks the calling task until the connection is established or the timeout expires. For non-blocking connection attempts, use [NoBlockConnect\(\)](#).
- `TickTimeout` provides more accurate timeout tracking than simple tick counts, especially when performing multiple sequential operations with a shared deadline.

Warning

The timeout parameter is passed by reference and will be modified by this function. If you need to preserve the original timeout value, create a copy.

Connection timeouts that are too short (less than 3 seconds) may fail on networks with moderate latency. Use at least 5 seconds for reliable operation.

Expand for Example Usage

Examples

Basic Connection with `TickTimeout`

```
#include <tcp.h>
#include <init.h>

void ConnectWithTickTimeout(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Create TickTimeout for 15 second timeout
    TickTimeout timeout(TICKS_PER_SECOND * 15);

    printf("Connecting with 15 second timeout...\n");
    printf("Initial timeout: %lu ticks\n", timeout.GetTimeLeft());

    int fd = connect(serverAddr, 8080, timeout);

    if (fd > 0) {
        printf("Connected successfully!\n");
        printf("Connection time: %lu ticks\n", timeout.GetElapsedTime());
        printf("Remaining timeout: %lu ticks\n", timeout.GetTimeLeft());

        close(fd);
    } else {
        printf("Connection failed: %d\n", fd);
        printf("Time spent: %lu ticks\n", timeout.GetElapsedTime());
    }
}
```

Shared Timeout Across Multiple Operations

```

#include <tcp.h>
#include <init.h>
#include <utils.h>

// Connect and perform complete handshake within total time budget
bool ConnectAndHandshake(IPADDR serverAddr, uint16_t port,
                        uint32_t totalTimeoutTicks) {
    // Create single TickTimeout for entire operation sequence
    TickTimeout timeout(totalTimeoutTicks);

    printf("Total time budget: %lu seconds\n",
           totalTimeoutTicks / TICKS_PER_SECOND);

    // Step 1: Connect (uses part of timeout)
    printf("\nStep 1: Connecting...\n");
    int fd = connect(serverAddr, port, timeout);

    if (fd < 0) {
        printf("Connection failed: %d\n", fd);
        return false;
    }

    printf("Connected! Remaining time: %lu ticks\n", timeout.GetTimeLeft());

    // Step 2: Send handshake (uses remaining timeout)
    printf("\nStep 2: Sending handshake...\n");
    const char *handshake = "HELLO\r\n";
    int sent = WriteWithTimeout(fd, handshake, strlen(handshake), timeout);

    if (sent != strlen(handshake)) {
        printf("Handshake send failed\n");
        close(fd);
        return false;
    }

    printf("Handshake sent! Remaining time: %lu ticks\n", timeout.GetTimeLeft());

    // Step 3: Receive response (uses remaining timeout)
    printf("\nStep 3: Receiving response...\n");
    char response[64];
    int received = ReadWithTimeout(fd, response, sizeof(response), timeout);

    if (received <= 0) {
        printf("Response read failed\n");
        close(fd);
        return false;
    }

    response[received] = '\0';
    printf("Received: %s\n", response);
    printf("Remaining time: %lu ticks\n", timeout.GetTimeLeft());

    // Summary
    uint32_t totalUsed = timeout.GetElapsedTime();
    printf("\n=== Summary ===\n");
    printf("Total time used: %lu ticks (%.1f seconds)\n",
           totalUsed, (float)totalUsed / TICKS_PER_SECOND);
    printf("Time remaining: %lu ticks\n", timeout.GetTimeLeft());
    printf("Handshake successful!\n");

    close(fd);
    return true;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Give entire sequence 20 seconds to complete
    bool success = ConnectAndHandshake(serverAddr, 8080, TICKS_PER_SECOND * 20);

    printf("\nOperation %s\n", success ? "SUCCEEDED" : "FAILED");

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Multiple Connection Attempts with Shared Timeout

```
#include <tcp.h>
#include <init.h>

// Try connecting to multiple backup servers within time limit
int ConnectToAnyServer(IPADDR servers[], int numServers,
                      uint16_t port, uint32_t totalTimeout) {
    TickTimeout timeout(totalTimeout);

    printf("Trying %d servers with total timeout of %lu seconds\n",
           numServers, totalTimeout / TICKS_PER_SECOND);

    for (int i = 0; i < numServers && !timeout.Expired(); i++) {
        printf("\nAttempt %d: Connecting to %s...",
               i + 1, servers[i].print());
        printf(" (time left: %lu ticks)\n", timeout.GetTimeLeft());

        int fd = connect(servers[i], port, timeout);

        if (fd > 0) {
            printf("Success! Connected to %s\n", servers[i].print());
            printf("Total time used: %lu ticks\n", timeout.GetElapsedTime());
            return fd;
        }

        printf("Failed: %d\n", fd);
    }

    if (timeout.Expired()) {
        printf("\nTimeout expired after trying %d servers\n", numServers);
    } else {
        printf("\nAll %d servers failed\n", numServers);
    }

    return -1;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Array of backup servers
    IPADDR servers[] = {
        AsciiToIp("192.168.1.10"), // Primary
        AsciiToIp("192.168.1.20"), // Backup 1
        AsciiToIp("192.168.1.30"), // Backup 2
        AsciiToIp("192.168.1.40")  // Backup 3
    };

    // Try all servers within 30 seconds total
    int fd = ConnectToAnyServer(servers, 4, 8080, TICKS_PER_SECOND * 30);

    if (fd > 0) {
        // Use connection...
        close(fd);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

Connection with Performance Monitoring

```
#include <tcp.h>
#include <init.h>

void MonitorConnectionPerformance(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Test with different timeout values
    uint32_t timeouts[] = {
        TICKS_PER_SECOND * 5,
        TICKS_PER_SECOND * 10,
        TICKS_PER_SECOND * 20
    };

    for (int i = 0; i < 3; i++) {
        TickTimeout timeout(timeouts[i]);
        uint32_t timeoutSecs = timeouts[i] / TICKS_PER_SECOND;

        printf("\n=== Test %d: %lu second timeout ===\n", i + 1, timeoutSecs);
    }
}
```

```

uint32_t startTick = TimeTick;
int fd = connect(serverAddr, 8080, timeout);
uint32_t endTick = TimeTick;

uint32_t actualTicks = endTick - startTick;
uint32_t actualMs = (actualTicks * 1000) / TICKS_PER_SECOND;

if (fd > 0) {
    printf("SUCCESS\n");
    printf("  Connection time: %lu ms\n", actualMs);
    printf("  Timeout object elapsed: %lu ticks\n",
           timeout.GetElapsedTime());
    printf("  Timeout object remaining: %lu ticks\n",
           timeout.GetTimeLeft());
    printf("  Timeout expired: %s\n",
           timeout.Expired() ? "yes" : "no");

    // Check efficiency
    float efficiency = ((float)actualTicks / timeouts[i]) * 100.0f;
    printf("  Timeout efficiency: %.1f%%\n", efficiency);

    close(fd);
} else {
    printf("FAILED: %d\n", fd);
    printf("  Time spent: %lu ms\n", actualMs);
    printf("  Timeout expired: %s\n",
           timeout.Expired() ? "yes" : "no");
}

OSTimeDly(TICKS_PER_SECOND * 2);
}
}

```

Timeout Budget Management

```

#include <tcp.h>
#include <init.h>

// Manage a time budget for multiple connection attempts
void TimebudgetedConnections(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // 60 second total budget
    TickTimeout totalBudget(TICKS_PER_SECOND * 60);
    int successfulConnections = 0;
    int failedConnections = 0;

    printf("Starting connection attempts with 60 second total budget\n\n");

    int attempt = 0;
    while (!totalBudget.Expired()) {
        attempt++;

        uint32_t remainingTime = totalBudget.GetTimeLeft();
        printf("Attempt %d (budget remaining: %lu sec)\n",
               attempt, remainingTime / TICKS_PER_SECOND);

        // Each attempt gets its own 5 second timeout
        TickTimeout attemptTimeout(TICKS_PER_SECOND * 5);

        int fd = connect(serverAddr, 8080, attemptTimeout);

        if (fd > 0) {
            successfulConnections++;
            printf("  Connected! (took %lu ticks)\n",
                   attemptTimeout.GetElapsedTime());

            // Do some work
            const char *msg = "PING\r\n";
            write(fd, msg, strlen(msg));

            close(fd);
        } else {
            failedConnections++;
            printf("  Failed: %d\n", fd);
        }

        // Small delay between attempts
        if (!totalBudget.Expired()) {
            OSTimeDly(TICKS_PER_SECOND * 2);
        }
    }

    printf("\n=== Summary ===\n");
}

```

```

    printf("Total attempts: %d\n", attempt);
    printf("Successful: %d\n", successfulConnections);
    printf("Failed: %d\n", failedConnections);
    printf("Time budget exhausted: %lu ticks used\n",
           totalBudget.GetElapsedTime());
}

```

Connection Retry with Exponential Backoff

```

#include <tcp.h>
#include <init.h>

// Retry connection with exponential backoff, respecting total timeout
int ConnectWithBackoff(IPADDR serverAddr, uint16_t port,
                      uint32_t totalTimeoutTicks) {
    TickTimeout timeout(totalTimeoutTicks);

    uint32_t attemptTimeout = TICKS_PER_SECOND * 3; // Start with 3 seconds
    int attempt = 0;

    printf("Connecting with exponential backoff\n");
    printf("Total timeout: %lu seconds\n",
           totalTimeoutTicks / TICKS_PER_SECOND);

    while (!timeout.Expired()) {
        attempt++;

        // Don't let attempt timeout exceed remaining time
        uint32_t remainingTime = timeout.GetTimeLeft();
        uint32_t thisAttemptTimeout = (attemptTimeout < remainingTime) ?
                                       attemptTimeout : remainingTime;

        printf("Attempt %d: timeout = %lu sec, remaining = %lu sec\n",
               attempt,
               thisAttemptTimeout / TICKS_PER_SECOND,
               remainingTime / TICKS_PER_SECOND);

        TickTimeout attemptTimer(thisAttemptTimeout);
        int fd = connect(serverAddr, port, attemptTimer);

        if (fd > 0) {
            printf("Success on attempt %d!\n", attempt);
            return fd;
        }

        printf("  Failed: %d\n", fd);

        // Exponential backoff (double the timeout, max 30 seconds)
        attemptTimeout *= 2;
        if (attemptTimeout > TICKS_PER_SECOND * 30) {
            attemptTimeout = TICKS_PER_SECOND * 30;
        }

        // Brief pause before retry
        if (!timeout.Expired()) {
            OSTimeDly(TICKS_PER_SECOND);
        }
    }

    printf("All attempts exhausted after %d tries\n", attempt);
    return -1;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Try for up to 2 minutes total
    int fd = ConnectWithBackoff(serverAddr, 8080, TICKS_PER_SECOND * 120);

    if (fd > 0) {
        // Use connection...
        close(fd);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```


Comparing TickTimeout vs Simple Timeout

```
#include <tcp.h>
#include <init.h>

void CompareTimeoutMethods(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Method 1: Simple timeout (uint32_t)
    printf("=== Method 1: Simple Timeout ===\n");
    uint32_t startTime1 = Secs;
    int fd1 = connect(serverAddr, 8080, TICKS_PER_SECOND * 10);
    uint32_t elapsed1 = Secs - startTime1;

    if (fd1 > 0) {
        printf("Connected in %lu seconds\n", elapsed1);
        printf("Cannot determine remaining timeout\n");
        printf("Cannot track elapsed time precisely\n");
        close(fd1);
    }

    OSTimeDly(TICKS_PER_SECOND * 2);

    // Method 2: TickTimeout object
    printf("\n=== Method 2: TickTimeout Object ===\n");
    TickTimeout timeout(TICKS_PER_SECOND * 10);

    printf("Initial state:\n");
    printf("  Time left: %lu ticks\n", timeout.GetTimeLeft());
    printf("  Expired: %s\n", timeout.Expired() ? "yes" : "no");

    int fd2 = connect(serverAddr, 8080, timeout);

    if (fd2 > 0) {
        printf("Connected!\n");
        printf("After connection:\n");
        printf("  Elapsed time: %lu ticks (%.2f seconds)\n",
            timeout.GetElapsedTime(),
            (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
        printf("  Time left: %lu ticks (%.2f seconds)\n",
            timeout.GetTimeLeft(),
            (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);
        printf("  Expired: %s\n", timeout.Expired() ? "yes" : "no");

        printf("\nAdvantages of TickTimeout:\n");
        printf("  - Precise elapsed time tracking\n");
        printf("  - Remaining time available\n");
        printf("  - Can be shared across operations\n");
        printf("  - Explicit expiration checking\n");

        close(fd2);
    }
}
```

Real-World Application: Multi-Step Protocol Handshake

```
#include <tcp.h>
#include <init.h>
#include <utils.h>

// Complex protocol with multiple request/response exchanges
bool PerformProtocolHandshake(IPADDR serverAddr, uint16_t port) {
    // Give entire protocol handshake 30 seconds
    TickTimeout protocolTimeout(TICKS_PER_SECOND * 30);

    printf("Starting protocol handshake (30 second timeout)\n\n");

    // Phase 1: Connect
    printf("Phase 1: Connecting...\n");
    int fd = connect(serverAddr, port, protocolTimeout);
    if (fd < 0) {
        printf("Connection failed: %d\n", fd);
        return false;
    }
    printf("Connected (%.1f sec elapsed, %.1f sec remaining)\n\n",
        (float)protocolTimeout.GetElapsedTime() / TICKS_PER_SECOND,
        (float)protocolTimeout.GetTimeLeft() / TICKS_PER_SECOND);

    // Phase 2: Send HELLO
    printf("Phase 2: Sending HELLO...\n");
    const char *hello = "HELLO v1.0\r\n";
    if (WriteWithTimeout(fd, hello, strlen(hello), protocolTimeout) < 0) {
        printf("HELLO send failed\n");
        close(fd);
    }
}
```

```

        return false;
    }
    printf("HELLO sent (%.1f sec remaining)\n\n",
        (float)protocolTimeout.GetTimeLeft() / TICKS_PER_SECOND);

    // Phase 3: Receive HELLO_ACK
    printf("Phase 3: Receiving HELLO_ACK...\n");
    char ackBuf[64];
    int n = ReadWithTimeout(fd, ackBuf, sizeof(ackBuf), protocolTimeout);
    if (n <= 0) {
        printf("HELLO_ACK receive failed\n");
        close(fd);
        return false;
    }
    ackBuf[n] = '\0';
    printf("Received: %s", ackBuf);
    printf("(%.1f sec remaining)\n\n",
        (float)protocolTimeout.GetTimeLeft() / TICKS_PER_SECOND);

    // Phase 4: Send AUTH
    printf("Phase 4: Sending AUTH...\n");
    const char *auth = "AUTH user:pass\r\n";
    if (WriteWithTimeout(fd, auth, strlen(auth), protocolTimeout) < 0) {
        printf("AUTH send failed\n");
        close(fd);
        return false;
    }
    printf("AUTH sent (%.1f sec remaining)\n\n",
        (float)protocolTimeout.GetTimeLeft() / TICKS_PER_SECOND);

    // Phase 5: Receive AUTH_OK
    printf("Phase 5: Receiving AUTH_OK...\n");
    n = ReadWithTimeout(fd, ackBuf, sizeof(ackBuf), protocolTimeout);
    if (n <= 0) {
        printf("AUTH_OK receive failed\n");
        close(fd);
        return false;
    }
    ackBuf[n] = '\0';
    printf("Received: %s", ackBuf);

    // Summary
    printf("\n== Handshake Complete ==\n");
    printf("Total time: %.1f seconds\n",
        (float)protocolTimeout.GetElapsedTime() / TICKS_PER_SECOND);
    printf("Time budget remaining: %.1f seconds\n",
        (float)protocolTimeout.GetTimeLeft() / TICKS_PER_SECOND);
    printf("Success!\n");

    close(fd);
    return true;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    bool success = PerformProtocolHandshake(serverAddr, 9000);

    printf("\nProtocol handshake %s\n", success ? "SUCCEEDED" : "FAILED");

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

See also

[connect\(const IPADDR&, uint16_t, uint32_t\)](#), [connectvia\(\)](#), [connectwlocal\(\)](#), [NoBlockConnect\(\)](#), [TickTimeout](#), [ReadWithTimeout\(\)](#), [WriteWithTimeout\(\)](#)

14.53.14.6 connect() [2/2]

```

int connect (
    const IPADDR & ipAddress,
    uint16_t remotePort,
    uint32_t timeout) [inline]

```

#include <tcp.h>

Establish a TCP connection to a remote host.

Creates a new TCP socket and attempts to connect to the specified remote IP address and port. This is a blocking call that waits up to the specified timeout period for the connection to be established. The function automatically selects the local interface based on the routing table and assigns an available local port.

This is the most commonly used function for establishing outbound TCP connections. The operating system automatically selects the appropriate local interface and port number based on the destination address and routing configuration.

Parameters

<i>ipAddress</i>	IP address of the remote host. Can be created using <code>AsciiToIp()</code> or obtained from DNS resolution. Example: <code>AsciiToIp("192.168.1.100")</code>
<i>remotePort</i>	Port number on the remote host (1-65535). Common ports include: • 80 for HTTP • 443 for HTTPS • 23 for Telnet • 22 for SSH
<i>timeout</i>	Maximum connection time in system ticks. Use <code>TICKS_PER_SECOND</code> macro: • <code>TICKS_PER_SECOND * 5</code> for 5 second timeout • <code>TICKS_PER_SECOND * 10</code> for 10 second timeout • <code>TICKS_PER_SECOND * 30</code> for 30 second timeout Typical timeout values are 5-30 seconds

Returns

Positive file descriptor on success (use for read/write operations), negative [TCP Socket Status](#) error code on failure:

- `TCP_ERR_TIMEOUT`: Connection not established within timeout period
- `TCP_ERR_NONE_AVAIL`: No available sockets in system pool
- `TCP_ERR_CON_RESET`: Connection was reset during establishment
- `TCP_ERR_CON_ABORT`: Connection was aborted

Notes:

- The local port number is automatically assigned from the ephemeral port range. You should not attempt to control the local port unless you have a specific protocol requirement (see [connectwlocal\(\)](#) for that use case).
- The local network interface is automatically selected based on the routing table for the destination IP address. For multi-homed systems where you need to control which interface is used, see [connectvia\(\)](#).
- Always call [close\(\)](#) on the returned file descriptor when finished to free system resources, even if read/write operations fail.
- This function blocks the calling task until the connection is established or the timeout expires. For non-blocking connection attempts, use [NoBlockConnect\(\)](#).

Warning

- Connection timeouts that are too short (less than 3 seconds) may fail on networks with moderate latency. Use at least 5 seconds for reliable operation.
- If the remote host is unreachable (network down, host down, firewall blocking), the function will wait the full timeout period before returning an error.

Expand for Example Usage

Examples

Basic HTTP Client Connection

```
#include <tcp.h>
#include <init.h>

void SimpleHttpClient(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Connect to web server
    IPADDR serverAddr = AsciiToIp("93.184.216.34"); // example.com
    int fd = connect(serverAddr, 80, TICKS_PER_SECOND * 10);

    if (fd < 0) {
        printf("Connection failed: %d\n", fd);
        return;
    }

    printf("Connected to server!\n");

    // Send HTTP GET request
    const char *request =
        "GET / HTTP/1.1\r\n"
        "Host: example.com\r\n"
        "Connection: close\r\n"
        "\r\n";

    writeall(fd, request, strlen(request));

    // Read response
    char buffer[512];
    int n = read(fd, buffer, sizeof(buffer) - 1);
    if (n > 0) {
        buffer[n] = '\0';
        printf("Received:\n%s\n", buffer);
    }

    close(fd);
    printf("Connection closed\n");
}
```

Connection with Error Handling

```
#include <tcp.h>
#include <init.h>

int ConnectWithRetry(IPADDR serverAddr, uint16_t port, int maxAttempts) {
    int attempt = 0;

    while (attempt < maxAttempts) {
        attempt++;
        printf("Connection attempt %d/%d to %s:%d\n",
            attempt, maxAttempts, serverAddr.print(), port);

        int fd = connect(serverAddr, port, TICKS_PER_SECOND * 10);

        if (fd > 0) {
            printf("Connection successful on attempt %d\n", attempt);
            return fd;
        }

        // Handle specific error codes
        switch (fd) {
            case TCP_ERR_TIMEOUT:
                printf(" Timeout - server not responding\n");
                break;

            case TCP_ERR_NONE_AVAIL:
                printf(" No sockets available - waiting for resources\n");
                OSTimeDly(TICKS_PER_SECOND * 5);
                break;

            case TCP_ERR_CON_RESET:
                printf(" Connection reset by server\n");
                break;

            case TCP_ERR_CON_ABORT:
                printf(" Connection aborted\n");
                break;

            default:
                printf(" Unknown error: %d\n", fd);
                break;
        }

        if (attempt < maxAttempts) {
            printf(" Retrying in 2 seconds...\n");
        }
    }
}
```

```

        OSTimeDly(TICKS_PER_SECOND * 2);
    }
}

printf("Failed to connect after %d attempts\n", maxAttempts);
return -1;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int fd = ConnectWithRetry(serverAddr, 8080, 3);

    if (fd > 0) {
        // Use connection...
        close(fd);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Telnet Client Example

```

#include <tcp.h>
#include <init.h>
#include <utils.h>

void TelnetClient(const char *hostname, uint16_t port) {
    // Resolve hostname to IP
    IPADDR serverAddr = AsciiToIp(hostname);
    if (serverAddr == IPADDR::NullIP()) {
        printf("Invalid hostname: %s\n", hostname);
        return;
    }

    printf("Connecting to %s:%d...\n", hostname, port);

    // Connect to telnet server
    int fd = connect(serverAddr, port, TICKS_PER_SECOND * 15);

    if (fd < 0) {
        printf("Connection failed: %d\n", fd);
        return;
    }

    printf("Connected! Remote: %s:%d, Local: %s:%d\n",
        GetSocketRemoteAddr(fd).print(),
        GetSocketRemotePort(fd),
        GetSocketLocalAddr(fd).print(),
        GetSocketLocalPort(fd));

    // Send command
    const char *command = "help\r\n";
    writeall(fd, command, strlen(command));

    // Read response (simple version)
    char buffer[1024];
    int totalRead = 0;

    while (1) {
        int n = read(fd, buffer, sizeof(buffer) - 1);
        if (n <= 0) break;

        buffer[n] = '\0';
        printf("%s", buffer);
        totalRead += n;

        if (n < sizeof(buffer) - 1) break; // No more data available
    }

    printf("\nTotal received: %d bytes\n", totalRead);
    close(fd);
}

```

Multiple Sequential Connections

```

#include <tcp.h>
#include <init.h>

// Connect to multiple servers in sequence
void SequentialConnections(void *pd) {
    init();
}

```

```

WaitForActiveNetwork(TICKS_PER_SECOND * 5);

// List of servers to connect to
struct ServerInfo {
    const char *name;
    IPADDR addr;
    uint16_t port;
};

ServerInfo servers[] = {
    {"Web Server",    AsciiToIp("192.168.1.10"), 80},
    {"Database",      AsciiToIp("192.168.1.20"), 5432},
    {"API Server",     AsciiToIp("192.168.1.30"), 8080},
    {"Sensor Node",    AsciiToIp("192.168.1.40"), 9000}
};

int numServers = sizeof(servers) / sizeof(servers[0]);
int successCount = 0;

printf("Checking connectivity to %d servers...\n\n", numServers);

for (int i = 0; i < numServers; i++) {
    printf("[%d/%d] Connecting to %s (%s:%d)...",
           i + 1, numServers,
           servers[i].name,
           servers[i].addr.print(),
           servers[i].port);

    uint32_t startTime = Secs;
    int fd = connect(servers[i].addr, servers[i].port,
                    TICKS_PER_SECOND * 5);
    uint32_t elapsed = Secs - startTime;

    if (fd > 0) {
        printf(" OK (%lu sec)\n", elapsed);
        successCount++;
        close(fd);
    } else {
        printf(" FAILED (error %d after %lu sec)\n", fd, elapsed);
    }
}

printf("\nConnectivity check complete: %d/%d successful\n",
        successCount, numServers);
}

```

Timeout Comparison

```

#include <tcp.h>
#include <init.h>

// Test different timeout values
void TimeoutComparison(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Try connecting to unreachable host with different timeouts
    IPADDR unreachable = AsciiToIp("10.255.255.1"); // Typically unreachable

    uint32_t timeouts[] = {
        TICKS_PER_SECOND * 3,    // 3 seconds
        TICKS_PER_SECOND * 5,    // 5 seconds
        TICKS_PER_SECOND * 10,   // 10 seconds
        TICKS_PER_SECOND * 30,   // 30 seconds
    };

    for (int i = 0; i < 4; i++) {
        uint32_t timeoutSecs = timeouts[i] / TICKS_PER_SECOND;

        printf("\nAttempting connection with %lu second timeout...\n",
                timeoutSecs);

        uint32_t startTime = Secs;
        int fd = connect(unreachable, 80, timeouts[i]);
        uint32_t actualTime = Secs - startTime;

        if (fd < 0) {
            printf(" Failed: %d\n", fd);
            printf(" Expected timeout: %lu seconds\n", timeoutSecs);
            printf(" Actual time: %lu seconds\n", actualTime);
            printf(" Difference: %ld seconds\n",
                   (long)(actualTime - timeoutSecs));
        } else {
            printf(" Unexpectedly succeeded!\n");
            close(fd);
        }
    }
}

```

```
}
```

Simple Data Exchange Client

```
#include <tcp.h>
#include <init.h>
#include <utils.h>

bool SendCommand(IPADDR server, uint16_t port,
                 const char *command, char *response, int responseSize) {
    // Connect to server
    int fd = connect(server, port, TICKS_PER_SECOND * 10);
    if (fd < 0) {
        printf("Connection failed: %d\n", fd);
        return false;
    }

    printf("Connected to %s:%d\n", server.print(), port);

    // Send command
    int sent = writeall(fd, command, strlen(command));
    if (sent != strlen(command)) {
        printf("Write failed\n");
        close(fd);
        return false;
    }

    printf("Sent: %s", command);

    // Read response with timeout
    int n = ReadWithTimeout(fd, response, responseSize - 1,
                           TICKS_PER_SECOND * 5);

    close(fd);

    if (n > 0) {
        response[n] = '\0';
        printf("Received: %s\n", response);
        return true;
    }

    printf("Read failed or timeout\n");
    return false;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR server = AsciiToIp("192.168.1.100");
    char response[256];

    if (SendCommand(server, 9000, "GET STATUS\r\n", response, sizeof(response))) {
        printf("Command successful\n");
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

Connection Monitoring

```
#include <tcp.h>
#include <init.h>

void MonitoredConnection(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR server = AsciiToIp("192.168.1.100");
    uint16_t port = 8080;

    printf("Attempting connection to %s:%d\n", server.print(), port);
    printf("Timeout: 15 seconds\n");

    // Record start time
    uint32_t startTick = TimeTick;

    int fd = connect(server, port, TICKS_PER_SECOND * 15);

    uint32_t elapsedTicks = TimeTick - startTick;
    uint32_t elapsedMs = (elapsedTicks * 1000) / TICKS_PER_SECOND;

    if (fd > 0) {
        printf("\n=== Connection Successful ===\n");
    }
}
```

```

printf(" Connection time: %lu ms\n", elapsedMs);
printf(" Local endpoint: %s:%d\n",
    GetSocketLocalAddr(fd).print(),
    GetSocketLocalPort(fd));
printf(" Remote endpoint: %s:%d\n",
    GetSocketRemoteAddr(fd).print(),
    GetSocketRemotePort(fd));
printf(" Socket state: %d\n", TcpGetSocketState(fd));
printf(" File descriptor: %d\n", fd);

// Test write availability
int writeSpace = writeavail(fd);
printf(" Write space: %d bytes\n", writeSpace);

close(fd);
printf("\nConnection closed\n");

} else {
printf("\n=== Connection Failed ===\n");
printf(" Error code: %d\n", fd);
printf(" Time taken: %lu ms\n", elapsedMs);

if (fd == TCP_ERR_TIMEOUT) {
printf(" Diagnosis: Server did not respond within timeout\n");
printf(" Suggestions:\n");
printf(" - Verify server is running\n");
printf(" - Check network connectivity\n");
printf(" - Verify firewall rules\n");
printf(" - Try longer timeout\n");
}
}
}
}

```

Persistent Connection with Reconnect

```

#include <tcp.h>
#include <init.h>
#include <nbrtos.h>

// Maintain persistent connection with automatic reconnect
void PersistentClient(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR server = AsciiToIp("192.168.1.100");
    uint16_t port = 9000;
    int fd = -1;

    printf("Starting persistent client to %s:%d\n",
        server.print(), port);

    while (1) {
        // Establish connection if needed
        if (fd < 0) {
            printf("Connecting...\n");
            fd = connect(server, port, TICKS_PER_SECOND * 10);

            if (fd < 0) {
                printf("Connection failed: %d, retrying in 5 seconds\n", fd);
                OSTimeDly(TICKS_PER_SECOND * 5);
                continue;
            }

            printf("Connected successfully\n");
        }

        // Use connection
        const char *heartbeat = "PING\r\n";
        int sent = write(fd, heartbeat, strlen(heartbeat));

        if (sent <= 0) {
            printf("Write failed, connection lost\n");
            close(fd);
            fd = -1;
            continue;
        }

        // Check for response
        char buffer[64];
        int n = read(fd, buffer, sizeof(buffer));

        if (n <= 0) {
            printf("Read failed, connection lost\n");
            close(fd);
            fd = -1;
            continue;
        }
    }
}

```



```

        buffer[n] = '\0';
        printf("Server response: %s", buffer);

        // Wait before next heartbeat
        OSTimeDly(TICKS_PER_SECOND * 10);
    }
}

```

See also

[connectvia\(\)](#), [connectwlocal\(\)](#), [NoBlockConnect\(\)](#), [listen\(\)](#), [accept\(\)](#), [close\(\)](#)

14.53.14.7 connectvia() [1/4]

```

int connectvia (
    const IPADDR & ipAddress,
    uint16_t remotePort,
    TickTimeout & timeout,
    const IPADDR & localIpAddress) [inline]
#include <tcp.h>

```

Establish a TCP connection using a specific local interface IP with TickTimeout.

Creates a new TCP socket and connects to the remote host using the specified local network interface identified by its IP address, with TickTimeout object for sophisticated timeout management. This combines the benefits of explicit interface selection with precise timeout tracking across multiple operations.

This function is ideal for multi-homed systems where you need both controlled interface selection and the ability to share timeout budgets across sequential operations. The TickTimeout object is updated to reflect remaining time, enabling accurate timeout management for complex connection sequences.

Parameters

<i>ipAddress</i>	IP address of the remote host. Can be IPv4 or IPv6 depending on build configuration
<i>remotePort</i>	Port number on the remote host (1-65535)
<i>timeout</i>	Reference to TickTimeout object specifying the maximum wait time. The TickTimeout tracks elapsed time and can be reused across multiple operations. The object is updated by this function to reflect remaining time after the connection attempt
<i>localIpAddress</i>	IP address of the local interface to use for this connection. The system will select the network interface that has this IP address assigned. Use IPADDR::NullIP() to fall back to automatic interface selection based on routing table

Returns

Positive file descriptor on success (use for read/write operations), negative [TCP Socket Status](#) error code on failure:

- `TCP_ERR_TIMEOUT`: Connection not established within timeout period
- `TCP_ERR_NONE_AVAIL`: No available sockets in system pool
- `TCP_ERR_CON_RESET`: Connection was reset during establishment
- `TCP_ERR_CON_ABORT`: Connection was aborted

Notes:

- The TickTimeout object is updated by this function to reflect the remaining time. This allows multiple operations to share a common timeout budget.
- The local port is automatically assigned from the ephemeral port range.
- If the specified local IP address does not exist on any interface, the behavior depends on the network stack implementation. Generally, the connection will fail or fall back to the default interface.
- [GetInterfaceIP\(\)](#) to obtain the IP address of a specific interface number before calling this function.

- Always call `close()` on the returned file descriptor when finished to free system resources.

-is function blocks the calling task until the connection is established or the timeout expires. For non-blocking connection attempts, use `NoBlockConnectVia()`.

Warning

- The timeout parameter is passed by reference and will be modified by this function. If you need to preserve the original timeout value, create a copy.
- Ensure the specified local IP address is actually assigned to an interface before calling this function. Use `GetInterfaceIP()` to verify.

Expand for Example Usage

Examples

Basic Interface Selection with TickTimeout

```
#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void ConnectViaInterfaceWithTimeout(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    IPADDR localIP = GetInterfaceIP(1); // Use interface 1

    printf("Connecting via interface %s\n", localIP.print());

    // Create TickTimeout for 15 seconds
    TickTimeout timeout(TICKS_PER_SECOND * 15);

    int fd = connectvia(serverAddr, 8080, timeout, localIP);

    if (fd > 0) {
        printf("Connected successfully!\n");
        printf("  Connection time: %.2f seconds\n",
            (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
        printf("  Remaining timeout: %.2f seconds\n",
            (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);
        printf("  Local endpoint: %s:%d\n",
            GetSocketLocalAddr(fd).print(),
            GetSocketLocalPort(fd));

        // Verify correct interface
        if (GetSocketLocalAddr(fd) == localIP) {
            printf("  Verified: Using requested interface\n");
        }

        close(fd);
    } else {
        printf("Connection failed: %d\n", fd);
        printf("Time spent: %.2f seconds\n",
            (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
    }
}
```

Multi-Interface Connection with Shared Timeout

```
#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Try multiple interfaces within total time budget
int ConnectViaAnyInterface(IPADDR serverAddr, uint16_t port,
    uint32_t totalTimeoutTicks) {
    TickTimeout timeout(totalTimeoutTicks);
    int numInterfaces = GetNumberOfInterfaces();

    printf("Trying %d interfaces with %.1f second total budget\n\n",
        numInterfaces,
        (float)totalTimeoutTicks / TICKS_PER_SECOND);

    for (int i = 0; i < numInterfaces && !timeout.Expired(); i++) {
        IPADDR localIP = GetInterfaceIP(i);

        printf("Attempt %d via interface %d (%s)...\n",
            i + 1, i, localIP.print());
        printf("  Time remaining: %.1f seconds\n",
            (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);

        int fd = connectvia(serverAddr, port, timeout, localIP);
```

```

        if (fd > 0) {
            printf("  SUCCESS!\n");
            printf("    Total time used: %.2f seconds\n",
                (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
            printf("    Time remaining: %.2f seconds\n",
                (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);
            return fd;
        }

        printf("  Failed: %d\n", fd);
    }

    if (timeout.Expired()) {
        printf("\nTimeout budget exhausted\n");
    } else {
        printf("\nAll interfaces failed\n");
    }

    return -1;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Try all interfaces within 30 seconds
    int fd = ConnectViaAnyInterface(serverAddr, 8080, TICKS_PER_SECOND * 30);

    if (fd > 0) {
        // Use connection...
        close(fd);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Dual-Homed System with Complete Protocol Exchange

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>
#include <utils.h>

// Connect via specific interface and perform multi-step protocol
bool ProtocolViaInterface(IPADDR serverAddr, uint16_t port,
    int interfaceNum, uint32_t totalTimeout) {
    TickTimeout timeout(totalTimeout);
    IPADDR localIP = GetInterfaceIP(interfaceNum);

    printf("Starting protocol via interface %d (%s)\n",
        interfaceNum, localIP.print());
    printf("Total timeout: %.1f seconds\n\n",
        (float)totalTimeout / TICKS_PER_SECOND);

    // Step 1: Connect via specific interface
    printf("Step 1: Connecting via interface %d...\n", interfaceNum);
    int fd = connectvia(serverAddr, port, timeout, localIP);

    if (fd < 0) {
        printf("Connection failed: %d\n", fd);
        return false;
    }

    printf("Connected! Elapsed: %.2f sec, Remaining: %.2f sec\n\n",
        (float)timeout.GetElapsedTime() / TICKS_PER_SECOND,
        (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);

    // Step 2: Send initialization
    printf("Step 2: Sending initialization...\n");
    const char *init = "INIT\r\n";
    if (WriteWithTimeout(fd, init, strlen(init), timeout) < 0) {
        printf("Init send failed\n");
        close(fd);
        return false;
    }
    printf("Init sent. Remaining: %.2f sec\n\n",
        (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);

    // Step 3: Receive acknowledgment
    printf("Step 3: Receiving ACK...\n");
    char ack[32];
    int n = ReadWithTimeout(fd, ack, sizeof(ack), timeout);
    if (n <= 0) {

```

```

        printf("ACK receive failed\n");
        close(fd);
        return false;
    }
    ack[n] = '\0';
    printf("Received: %s", ack);
    printf("Remaining: %.2f sec\n\n",
        (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);

    // Step 4: Send data
    printf("Step 4: Sending data...\n");
    const char *data = "DATA payload\r\n";
    if (WriteWithTimeout(fd, data, strlen(data), timeout) < 0) {
        printf("Data send failed\n");
        close(fd);
        return false;
    }

    printf("Protocol complete!\n");
    printf("Total time: %.2f seconds\n",
        (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
    printf("Unused time: %.2f seconds\n",
        (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);

    close(fd);
    return true;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Perform protocol via interface 1 with 20 second budget
    bool success = ProtocolViaInterface(serverAddr, 8080, 1,
        TICKS_PER_SECOND * 20);

    printf("\nProtocol %s\n", success ? "SUCCEEDED" : "FAILED");

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Interface Failover with Timeout Tracking

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Try primary interface, failover to backup if needed
int ConnectWithInterfaceFailover(IPADDR serverAddr, uint16_t port,
    int primaryIF, int backupIF,
    uint32_t totalTimeout) {
    TickTimeout timeout(totalTimeout);

    printf("Connection with failover (%.1f sec total budget)\n",
        (float)totalTimeout / TICKS_PER_SECOND);

    // Try primary interface
    IPADDR primaryIP = GetInterfaceIP(primaryIF);
    printf("\nTrying primary interface %d (%s)...\n",
        primaryIF, primaryIP.print());

    int fd = connectvia(serverAddr, port, timeout, primaryIP);

    if (fd > 0) {
        printf("Connected via primary interface\n");
        printf("Connection time: %.2f seconds\n",
            (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
        return fd;
    }

    printf("Primary failed: %d (%.2f sec elapsed)\n",
        fd, (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);

    // Check if we have time for backup
    if (timeout.Expired()) {
        printf("No time remaining for backup attempt\n");
        return -1;
    }

    // Try backup interface
    IPADDR backupIP = GetInterfaceIP(backupIF);
    printf("\nFailing over to backup interface %d (%s)...\n",
        backupIF, backupIP.print());
}

```

```

printf("Time remaining: %.2f seconds\n",
      (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);

fd = connectvia(serverAddr, port, timeout, backupIP);

if (fd > 0) {
    printf("Connected via backup interface\n");
    printf("Total time: %.2f seconds\n",
          (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
    return fd;
}

printf("Backup also failed: %d\n", fd);
printf("Total time spent: %.2f seconds\n",
      (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);

return -1;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Try interface 0, failover to 1, with 20 second total budget
    int fd = ConnectWithInterfaceFailover(serverAddr, 8080, 0, 1,
                                         TICKS_PER_SECOND * 20);

    if (fd > 0) {
        printf("\nConnection established!\n");
        printf("Using interface: %d\n", TcpGetSocketInterface(fd));
        close(fd);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Round-Robin Interface Selection with Time Budget

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Distribute connections across interfaces with total time limit
void RoundRobinWithBudget(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    // 60 second total budget for all connections
    TickTimeout totalBudget(TICKS_PER_SECOND * 60);

    int successCount = 0;
    int failCount = 0;
    int connectionNum = 0;

    printf("Round-robin connections with 60 second budget\n");
    printf("Interfaces: %d\n\n", numInterfaces);

    while (!totalBudget.Expired()) {
        int ifNum = connectionNum % numInterfaces;
        IPADDR localIP = GetInterfaceIP(ifNum);

        printf("Connection %d via interface %d (%s)...",
              connectionNum + 1, ifNum, localIP.print());
        printf(" [%.1f sec left]\n",
              (float)totalBudget.GetTimeLeft() / TICKS_PER_SECOND);

        // Each attempt gets up to 5 seconds
        TickTimeout attemptTimeout(TICKS_PER_SECOND * 5);

        int fd = connectvia(serverAddr, 8080, attemptTimeout, localIP);

        if (fd > 0) {
            successCount++;
            printf(" Success (%.2f sec)\n",
                  (float)attemptTimeout.GetElapsedTime() / TICKS_PER_SECOND);
            close(fd);
        } else {
            failCount++;
            printf(" Failed: %d\n", fd);
        }
        connectionNum++;
    }
}

```

```

        connectionNum++;

        // Small pause between connections
        if (!totalBudget.Expired()) {
            OSTimeDly(TICKS_PER_SECOND);
        }
    }

    printf("\n=== Summary ===\n");
    printf("Total attempts: %d\n", connectionNum);
    printf("Successful: %d\n", successCount);
    printf("Failed: %d\n", failCount);
    printf("Time used: %.1f seconds\n",
        (float)totalBudget.GetElapsedTime() / TICKS_PER_SECOND);
}

```

Performance Comparison Across Interfaces

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void CompareInterfacePerformance(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    printf("Comparing connection performance across interfaces\n\n");

    for (int i = 0; i < numInterfaces; i++) {
        IPADDR localIP = GetInterfaceIP(i);

        printf("=== Interface %d (%s) ===\n", i, localIP.print());

        // Measure 5 connection attempts
        uint32_t totalTime = 0;
        int successes = 0;

        for (int attempt = 0; attempt < 5; attempt++) {
            TickTimeout timeout(TICKS_PER_SECOND * 10);

            int fd = connectvia(serverAddr, 8080, timeout, localIP);

            if (fd > 0) {
                uint32_t elapsed = timeout.GetElapsedTime();
                totalTime += elapsed;
                successes++;

                printf("  Attempt %d: %.3f sec\n",
                    attempt + 1,
                    (float)elapsed / TICKS_PER_SECOND);

                close(fd);
            } else {
                printf("  Attempt %d: FAILED (%d)\n", attempt + 1, fd);
            }

            OSTimeDly(TICKS_PER_SECOND / 2);
        }

        if (successes > 0) {
            float avgTime = (float)totalTime / successes / TICKS_PER_SECOND;
            printf("Average: %.3f sec (%d/%d successful)\n",
                avgTime, successes, 5);
        } else {
            printf("All attempts failed\n");
        }

        printf("\n");
        OSTimeDly(TICKS_PER_SECOND);
    }

    printf("Performance comparison complete\n");
}

```

Adaptive Timeout Based on Interface Performance

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Track performance and adjust timeouts accordingly
struct InterfaceStats {

```

```

uint32_t avgConnectionTime;
int attemptCount;
};

void AdaptiveInterfaceConnection(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    // Initialize statistics
    InterfaceStats stats[4] = {0}; // Support up to 4 interfaces

    for (int i = 0; i < numInterfaces; i++) {
        stats[i].avgConnectionTime = TICKS_PER_SECOND * 5; // Initial 5 sec
        stats[i].attemptCount = 0;
    }

    printf("Adaptive interface connection with learned timeouts\n\n");

    for (int conn = 0; conn < 10; conn++) {
        int ifNum = conn % numInterfaces;
        IPADDR localIP = GetInterfaceIP(ifNum);

        // Calculate adaptive timeout (avg + 50% margin)
        uint32_t adaptiveTimeout = stats[ifNum].avgConnectionTime * 3 / 2;

        printf("Connection %d via interface %d\n", conn + 1, ifNum);
        printf(" Adaptive timeout: %.2f sec\n",
            (float)adaptiveTimeout / TICKS_PER_SECOND);

        TickTimeout timeout(adaptiveTimeout);
        int fd = connectvia(serverAddr, 8080, timeout, localIP);

        if (fd > 0) {
            uint32_t elapsed = timeout.GetElapsedTime();

            printf(" Success in %.2f sec\n",
                (float)elapsed / TICKS_PER_SECOND);

            // Update running average
            if (stats[ifNum].attemptCount == 0) {
                stats[ifNum].avgConnectionTime = elapsed;
            } else {
                stats[ifNum].avgConnectionTime =
                    (stats[ifNum].avgConnectionTime + elapsed) / 2;
            }
            stats[ifNum].attemptCount++;

            printf(" New avg for IF%d: %.2f sec\n",
                ifNum,
                (float)stats[ifNum].avgConnectionTime / TICKS_PER_SECOND);

            close(fd);
        } else {
            printf(" Failed: %d\n", fd);
        }

        printf("\n");
        OSTimeDly(TICKS_PER_SECOND);
    }

    printf("=== Final Statistics ===\n");
    for (int i = 0; i < numInterfaces; i++) {
        printf("Interface %d: %.2f sec avg (%d attempts)\n",
            i,
            (float)stats[i].avgConnectionTime / TICKS_PER_SECOND,
            stats[i].attemptCount);
    }
}

```

Parallel Connection Attempts with First-Success

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>
#include <nbrtos.h>

// Global results from parallel attempts
struct ConnectionAttempt {
    int interfaceNum;
    int fd;
    uint32_t elapsedTime;
    bool completed;
} g_attempts[4];

// Worker task for parallel connection
void ConnectionWorker(void *pdata) {
    ConnectionAttempt *attempt = (ConnectionAttempt *)pdata;

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    IPADDR localIP = GetInterfaceIP(attempt->interfaceNum);

    TickTimeout timeout(TICKS_PER_SECOND * 10);

    printf("[Worker IF%d] Connecting via %s...\n",
        attempt->interfaceNum, localIP.print());

    attempt->fd = connectvia(serverAddr, 8080, timeout, localIP);
    attempt->elapsedTime = timeout.GetElapsedTime();
    attempt->completed = true;

    if (attempt->fd > 0) {
        printf("[Worker IF%d] SUCCESS in %.2f sec\n",
            attempt->interfaceNum,
            (float)attempt->elapsedTime / TICKS_PER_SECOND);
    } else {
        printf("[Worker IF%d] FAILED: %d\n",
            attempt->interfaceNum, attempt->fd);
    }
}

// Note: This is a conceptual example - actual implementation would
// require proper task synchronization and stack management
void ParallelConnectionExample(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int numInterfaces = GetNumberOfInterfaces();

    printf("Parallel connection attempts across %d interfaces\n\n",
        numInterfaces);

    // Initialize attempts
    for (int i = 0; i < numInterfaces; i++) {
        g_attempts[i].interfaceNum = i;
        g_attempts[i].fd = -1;
        g_attempts[i].completed = false;
    }

    // Launch workers (simplified - would need proper stack allocation)
    printf("Launching parallel connection attempts...\n");

    // Wait for first success or all complete
    uint32_t startTime = Secs;
    int firstSuccess = -1;

    while ((Secs - startTime) < 15) {
        for (int i = 0; i < numInterfaces; i++) {
            if (g_attempts[i].completed && g_attempts[i].fd > 0) {
                firstSuccess = i;
                break;
            }
        }

        if (firstSuccess >= 0) break;

        OSTimeDly(TICKS_PER_SECOND / 10);
    }

    if (firstSuccess >= 0) {
        printf("\nFirst success: Interface %d\n", firstSuccess);
        printf("Closing other connections...\n");

        for (int i = 0; i < numInterfaces; i++) {
            if (i != firstSuccess && g_attempts[i].fd > 0) {
                close(g_attempts[i].fd);
            }
        }
    }
}

```



```

    }

    close(g_attempts[firstSuccess].fd);
}
}

```

Connection Quality Monitoring

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void MonitorConnectionQuality(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    printf("Monitoring connection quality across interfaces\n\n");

    for (int cycle = 0; cycle < 5; cycle++) {
        printf("=== Cycle %d ===\n", cycle + 1);

        for (int i = 0; i < numInterfaces; i++) {
            IPADDR localIP = GetInterfaceIP(i);

            TickTimeout timeout(TICKS_PER_SECOND * 8);
            int fd = connectvia(serverAddr, 8080, timeout, localIP);

            if (fd > 0) {
                uint32_t connectTime = timeout.GetElapsedTime();

                // Measure connection quality
                const char *ping = "PING\r\n";
                TickTimeout writeTimeout(TICKS_PER_SECOND * 2);

                uint32_t writeStart = TimeTick;
                int sent = WriteWithTimeout(fd, ping, strlen(ping),
                                           writeTimeout);
                uint32_t writeTime = TimeTick - writeStart;

                printf("IF%d: Connect=%.2fms Write=%.2fms",
                       i,
                       (float)(connectTime * 1000) / TICKS_PER_SECOND,
                       (float)(writeTime * 1000) / TICKS_PER_SECOND);

                if (sent > 0) {
                    printf(" [GOOD]\n");
                } else {
                    printf(" [POOR]\n");
                }

                close(fd);
            } else {
                printf("IF%d: FAILED (%d)\n", i, fd);
            }
        }

        printf("\n");
        OSTimeDly(TICKS_PER_SECOND * 5);
    }
}

```

See also

[connectvia\(const IPADDR&, uint16_t, uint32_t, const IPADDR&\)](#), [connectvia\(const IPADDR&, uint16_t, TickTimeout&, int\)](#), [connectwlocal\(\)](#), [NoBlockConnectVia\(\)](#), [GetInterfaceIP\(\)](#), [TcpGetSocketInterface\(\)](#), [TickTimeout](#)

14.53.14.8 connectvia() [2/4]

```

int connectvia (
    const IPADDR & ipAddress,
    uint16_t remotePort,
    TickTimeout & timeout,
    int ifnum) [inline]

```

#include <tcp.h>

Establish a TCP connection using a specific local interface number with TickTimeout.

Creates a new TCP socket and connects to the remote host using the specified local network interface number, with TickTimeout object for sophisticated timeout management. This provides the most direct interface control combined with precise timeout tracking, ideal for complex multi-homed systems requiring both explicit interface selection and accurate time budget management.

This function combines the unambiguous interface control of interface number selection with the advanced timeout capabilities of TickTimeout. The interface number takes precedence over all routing decisions, and the TickTimeout object tracks elapsed time for sharing timeout budgets across sequential operations.

Parameters

<i>ipAddress</i>	IP address of the remote host. Can be IPv4 or IPv6 depending on build configuration
<i>remotePort</i>	Port number on the remote host (1-65535)
<i>timeout</i>	Reference to TickTimeout object specifying the maximum wait time. The TickTimeout tracks elapsed time and can be reused across multiple operations. The object is updated by this function to reflect remaining time after the connection attempt
<i>ifnum</i>	Local interface number (0, 1, 2, etc.) to use for this connection. This overrides all routing decisions based on destination IP addresses. The interface number directly corresponds to the physical or logical network interface. Valid range is 0 to (number of interfaces - 1). Use GetNumberOfInterfaces() to determine valid range

Returns

Positive file descriptor on success (use for read/write operations), negative [TCP Socket Status](#) error code on failure:

- TCP_ERR_TIMEOUT: Connection not established within timeout period
- TCP_ERR_NONE_AVAIL: No available sockets in system pool
- TCP_ERR_CON_RESET: Connection was reset during establishment
- TCP_ERR_CON_ABORT: Connection was aborted

Notes:

- The TickTimeout object is updated by this function to reflect the remaining time. This allows multiple operations to share a common timeout budget.
- The local port is automatically assigned from the ephemeral port range.
- Interface numbers typically start at 0 and increment sequentially. Interface 0 is usually the primary Ethernet port, but this can vary by platform and configuration.
- This combination of interface number selection and TickTimeout provides the most control and precision for multi-homed systems with complex timing requirements.
- Always call [close\(\)](#) on the returned file descriptor when finished to free system resources.
- This function blocks the calling task until the connection is established or the timeout expires. For non-blocking connection attempts, use [NoBlockConnectVia\(\)](#).

Warning

- The timeout parameter is passed by reference and will be modified by this function. If you need to preserve the original timeout value, create a copy.
- Specifying an invalid interface number (negative or $\geq$ number of interfaces) may cause unpredictable behavior or connection failure. Always validate with GetNumberOfInterfaces() first.
- The specified interface must be in an "up" state with a valid IP address assigned for the connection to succeed.

Expand for Example Usage

Examples

Basic Interface Number Selection with TickTimeout

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void ConnectViaInterfaceWithTimeout(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int interfaceNum = 1; // Use interface 1

    printf("Connecting via interface %d\n", interfaceNum);
    printf("Interface %d IP: %s\n",
        interfaceNum, GetInterfaceIP(interfaceNum).print());

    // Create TickTimeout for 15 seconds
    TickTimeout timeout(TICKS_PER_SECOND * 15);

    int fd = connectvia(serverAddr, 8080, timeout, interfaceNum);

    if (fd > 0) {
        printf("Connected successfully!\n");
        printf("  Connection time: %.2f seconds\n",
            (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
        printf("  Remaining timeout: %.2f seconds\n",
            (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);
        printf("  Interface used: %d\n", TcpGetSocketInterface(fd));
        printf("  Local endpoint: %s:%d\n",
            GetSocketLocalAddr(fd).print(),
            GetSocketLocalPort(fd));

        close(fd);
    } else {
        printf("Connection failed: %d\n", fd);
        printf("Time spent: %.2f seconds\n",
            (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
        printf("Timeout expired: %s\n", timeout.Expired() ? "yes" : "no");
    }
}

```

Sequential Interface Attempts with Shared Timeout

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Try each interface in sequence until success or timeout
int ConnectViaAnyInterface(IPADDR serverAddr, uint16_t port,
    uint32_t totalTimeoutTicks) {
    TickTimeout timeout(totalTimeoutTicks);
    int numInterfaces = GetNumberOfInterfaces();

    printf("Trying %d interfaces sequentially\n", numInterfaces);
    printf("Total timeout budget: %.1f seconds\n\n",
        (float)totalTimeoutTicks / TICKS_PER_SECOND);

    for (int i = 0; i < numInterfaces && !timeout.Expired(); i++) {
        printf("Attempt %d: Interface %d (%s)\n",
            i + 1, i, GetInterfaceIP(i).print());
        printf("  Time remaining: %.2f seconds\n",
            (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);

        int fd = connectvia(serverAddr, port, timeout, i);

        if (fd > 0) {
            printf("  SUCCESS!\n");
            printf("  Total time: %.2f seconds\n",
                (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
            printf("  Unused time: %.2f seconds\n",
                (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);
            return fd;
        }

        printf("  Failed: %d (%.2f sec used)\n",
            fd, (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
    }

    if (timeout.Expired()) {
        printf("\nTimeout budget exhausted (%.2f sec)\n",
            (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
    } else {
        printf("\nAll %d interfaces failed\n", numInterfaces);
    }

    return -1;
}

```

```

}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Try all interfaces within 30 second budget
    int fd = ConnectViaAnyInterface(serverAddr, 8080, TICKS_PER_SECOND * 30);

    if (fd > 0) {
        printf("\nConnection established!\n");
        close(fd);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Complete Protocol Sequence via Specific Interface

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>
#include <utils.h>

// Perform complete protocol handshake via specific interface
bool ProtocolViaInterface(IPADDR serverAddr, uint16_t port,
                          int interfaceNum, uint32_t totalTimeout) {
    TickTimeout timeout(totalTimeout);

    printf("Protocol sequence via interface %d\n", interfaceNum);
    printf("Interface IP: %s\n", GetInterfaceIP(interfaceNum).print());
    printf("Total timeout: %.1f seconds\n\n",
           (float)totalTimeout / TICKS_PER_SECOND);

    // Phase 1: Connect
    printf("Phase 1: Connecting...\n");
    int fd = connectvia(serverAddr, port, timeout, interfaceNum);

    if (fd < 0) {
        printf("Connection failed: %d\n", fd);
        return false;
    }

    printf("Connected! Elapsed: %.2f sec, Remaining: %.2f sec\n\n",
           (float)timeout.GetElapsedTime() / TICKS_PER_SECOND,
           (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);

    // Phase 2: Authentication
    printf("Phase 2: Authentication...\n");
    const char *auth = "AUTH user:password\r\n";
    if (WriteWithTimeout(fd, auth, strlen(auth), timeout) < 0) {
        printf("Auth send failed\n");
        close(fd);
        return false;
    }

    char authResp[64];
    int n = ReadWithTimeout(fd, authResp, sizeof(authResp), timeout);
    if (n <= 0) {
        printf("Auth response read failed\n");
        close(fd);
        return false;
    }
    authResp[n] = '\0';
    printf("Auth response: %s", authResp);
    printf("Remaining: %.2f sec\n\n",
           (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);

    // Phase 3: Command
    printf("Phase 3: Sending command...\n");
    const char *cmd = "GET STATUS\r\n";
    if (WriteWithTimeout(fd, cmd, strlen(cmd), timeout) < 0) {
        printf("Command send failed\n");
        close(fd);
        return false;
    }

    char cmdResp[256];
    n = ReadWithTimeout(fd, cmdResp, sizeof(cmdResp), timeout);
    if (n <= 0) {
        printf("Command response read failed\n");
        close(fd);
        return false;
    }
}

```

```

    }
    cmdResp[n] = '\0';
    printf("Status: %s", cmdResp);

    // Summary
    printf("\n=== Protocol Complete ===\n");
    printf("Total time: %.2f seconds\n",
        (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
    printf("Time budget remaining: %.2f seconds\n",
        (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);
    printf("Efficiency: %.1f%%\n",
        (float)timeout.GetElapsedTime() * 100.0f / totalTime);

    close(fd);
    return true;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Execute protocol via interface 0 with 30 second budget
    bool success = ProtocolViaInterface(serverAddr, 8080, 0,
        TICKS_PER_SECOND * 30);

    printf("\nProtocol %s\n", success ? "SUCCEEDED" : "FAILED");

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Priority-Based Interface Selection with Timeout

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Try interfaces in priority order with exponential backoff
int ConnectWithPriority(IPADDR serverAddr, uint16_t port,
    int priorityInterfaces[], int numPriorities,
    uint32_t totalTime) {
    TickTimeout timeout(totalTime);

    printf("Priority-based connection attempt\n");
    printf("Priorities: ");
    for (int i = 0; i < numPriorities; i++) {
        printf("%d ", priorityInterfaces[i]);
    }
    printf("\n\n");

    uint32_t attemptTimeout = TICKS_PER_SECOND * 3; // Start with 3 seconds

    for (int i = 0; i < numPriorities && !timeout.Expired(); i++) {
        int ifNum = priorityInterfaces[i];

        // Don't exceed remaining time
        uint32_t thisTimeout = (attemptTimeout < timeout.GetTimeLeft()) ?
            attemptTimeout : timeout.GetTimeLeft();

        printf("Priority %d: Interface %d (%s)\n",
            i + 1, ifNum, GetInterfaceIP(ifNum).print());
        printf("  Attempt timeout: %.1f sec\n",
            (float)thisTimeout / TICKS_PER_SECOND);
        printf("  Total remaining: %.1f sec\n",
            (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);

        TickTimeout attemptTimer(thisTimeout);
        int fd = connectvia(serverAddr, port, attemptTimer, ifNum);

        if (fd > 0) {
            printf("  SUCCESS at priority %d!\n", i + 1);
            printf("  Time used: %.2f seconds\n",
                (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
            return fd;
        }

        printf("  Failed: %d\n", fd);

        // Exponential backoff for next attempt
        attemptTimeout *= 2;
        if (attemptTimeout > TICKS_PER_SECOND * 15) {
            attemptTimeout = TICKS_PER_SECOND * 15; // Cap at 15 seconds
        }
    }
}

```

```

    printf("\nAll priority interfaces failed\n");
    return -1;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Priority order: try interface 0, then 2, then 1
    int priorities[] = {0, 2, 1};

    int fd = ConnectWithPriority(serverAddr, 8080, priorities, 3,
                                TICKS_PER_SECOND * 60);

    if (fd > 0) {
        close(fd);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Load Balancing with Time Budget

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Distribute connections across interfaces with total time limit
void LoadBalancedConnections(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    // 2 minute total budget
    TickTimeout totalBudget(TICKS_PER_SECOND * 120);

    int connectionCounts[4] = {0}; // Support up to 4 interfaces
    int totalConnections = 0;
    int successCount = 0;

    printf("Load-balanced connections with 2 minute budget\n");
    printf("Interfaces: %d\n\n", numInterfaces);

    while (!totalBudget.Expired()) {
        // Round-robin interface selection
        int ifNum = totalConnections % numInterfaces;

        printf("Connection %d via interface %d [%.1f sec left]...",
              totalConnections + 1, ifNum,
              (float)totalBudget.GetTimeLeft() / TICKS_PER_SECOND);

        // Each attempt gets up to 5 seconds
        TickTimeout attemptTimeout(TICKS_PER_SECOND * 5);

        int fd = connectvia(serverAddr, 8080, attemptTimeout, ifNum);

        if (fd > 0) {
            successCount++;
            connectionCounts[ifNum]++;
            printf(" OK (%.2f sec)\n",
                  (float)attemptTimeout.GetElapsedTime() / TICKS_PER_SECOND);

            // Do some work
            const char *msg = "PING\r\n";
            write(fd, msg, strlen(msg));

            close(fd);
        } else {
            printf(" FAILED: %d\n", fd);
        }

        totalConnections++;

        // Brief pause
        if (!totalBudget.Expired()) {
            OSTimeDly(TICKS_PER_SECOND / 2);
        }
    }

    printf("\n=== Summary ===\n");
}

```

```

printf("Total time: %.1f seconds\n",
      (float)totalBudget.GetElapsedTime() / TICKS_PER_SECOND);
printf("Total attempts: %d\n", totalConnections);
printf("Successful: %d (%.1f%%)\n",
      successCount,
      (float)successCount * 100.0f / totalConnections);

printf("\nPer-Interface Distribution:\n");
for (int i = 0; i < numInterfaces; i++) {
    printf("  Interface %d: %d connections\n", i, connectionCounts[i]);
}
}

```

Interface Performance Profiling

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

struct InterfaceProfile {
    int interfaceNum;
    uint32_t minTime;
    uint32_t maxTime;
    uint32_t totalTime;
    int successCount;
    int failCount;
};

void ProfileInterfacePerformance(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    InterfaceProfile profiles[4] = {0}; // Support up to 4 interfaces

    // Initialize profiles
    for (int i = 0; i < numInterfaces; i++) {
        profiles[i].interfaceNum = i;
        profiles[i].minTime = 0xFFFFFFFF;
        profiles[i].maxTime = 0;
        profiles[i].totalTime = 0;
        profiles[i].successCount = 0;
        profiles[i].failCount = 0;
    }

    printf("Profiling %d interfaces (10 attempts each)\n\n", numInterfaces);

    // Profile each interface with 10 attempts
    for (int i = 0; i < numInterfaces; i++) {
        printf("Profiling Interface %d (%s)\n",
            i, GetInterfaceIP(i).print());

        for (int attempt = 0; attempt < 10; attempt++) {
            TickTimeout timeout(TICKS_PER_SECOND * 10);

            int fd = connectvia(serverAddr, 8080, timeout, i);

            if (fd > 0) {
                uint32_t elapsed = timeout.GetElapsedTime();

                profiles[i].successCount++;
                profiles[i].totalTime += elapsed;

                if (elapsed < profiles[i].minTime) {
                    profiles[i].minTime = elapsed;
                }
                if (elapsed > profiles[i].maxTime) {
                    profiles[i].maxTime = elapsed;
                }

                printf("  Attempt %d: %.3f sec\n",
                    attempt + 1,
                    (float)elapsed / TICKS_PER_SECOND);

                close(fd);
            } else {
                profiles[i].failCount++;
                printf("  Attempt %d: FAILED (%d)\n", attempt + 1, fd);
            }

            OSTimeDly(TICKS_PER_SECOND / 2);
        }

        printf("\n");
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

```

    }

    // Display profile results
    printf("=== Performance Profile Results ===\n\n");

    for (int i = 0; i < numInterfaces; i++) {
        printf("Interface %d:\n", i);
        printf("  IP Address: %s\n", GetInterfaceIP(i).print());
        printf("  Success Rate: %d/%d (%.1f%%)\n",
            profiles[i].successCount,
            profiles[i].successCount + profiles[i].failCount,
            (float)profiles[i].successCount * 100.0f /
            (profiles[i].successCount + profiles[i].failCount));

        if (profiles[i].successCount > 0) {
            float avgTime = (float)profiles[i].totalTime /
                profiles[i].successCount / TICKS_PER_SECOND;
            float minTime = (float)profiles[i].minTime / TICKS_PER_SECOND;
            float maxTime = (float)profiles[i].maxTime / TICKS_PER_SECOND;

            printf("  Min Time: %.3f sec\n", minTime);
            printf("  Avg Time: %.3f sec\n", avgTime);
            printf("  Max Time: %.3f sec\n", maxTime);
            printf("  Variance: %.3f sec\n", maxTime - minTime);
        }

        printf("\n");
    }

    // Recommend best interface
    int bestInterface = 0;
    float bestScore = 0.0f;

    for (int i = 0; i < numInterfaces; i++) {
        if (profiles[i].successCount > 0) {
            float avgTime = (float)profiles[i].totalTime /
                profiles[i].successCount;
            float successRate = (float)profiles[i].successCount /
                (profiles[i].successCount + profiles[i].failCount);

            // Score based on success rate and speed (lower time is better)
            float score = successRate / avgTime;

            if (score > bestScore) {
                bestScore = score;
                bestInterface = i;
            }
        }
    }

    printf("Recommended Interface: %d (%s)\n",
        bestInterface, GetInterfaceIP(bestInterface).print());
}

```

Adaptive Retry Strategy

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Adaptively retry with learned timeout adjustments
int AdaptiveConnect(IPADDR serverAddr, uint16_t port, int interfaceNum,
    uint32_t totalTimeOut) {
    TickTimeout timeout(totalTimeOut);

    // Start with short timeout
    uint32_t attemptTimeout = TICKS_PER_SECOND * 2;
    int attempt = 0;

    printf("Adaptive connection via interface %d\n", interfaceNum);
    printf("Total budget: %.1f seconds\n\n",
        (float)totalTimeOut / TICKS_PER_SECOND);

    while (!timeout.Expired() && attempt < 10) {
        attempt++;

        uint32_t thisTimeout = (attemptTimeout < timeout.GetTimeLeft()) ?
            attemptTimeout : timeout.GetTimeLeft();

        printf("Attempt %d: timeout=%.1f sec, budget=%.1f sec\n",
            attempt,
            (float)thisTimeout / TICKS_PER_SECOND,
            (float)timeout.GetTimeLeft() / TICKS_PER_SECOND);

        TickTimeout attemptTimer(thisTimeout);
        int fd = connectvia(serverAddr, port, attemptTimer, interfaceNum);
    }
}

```



```

    if (fd > 0) {
        printf("SUCCESS on attempt %d!\n", attempt);
        printf("Total time: %.2f seconds\n",
            (float)timeout.GetElapsedTime() / TICKS_PER_SECOND);
        return fd;
    }

    uint32_t usedTime = attemptTimer.GetElapsedTime();
    bool timedOut = attemptTimer.Expired();

    printf("    Failed: %d (%s)\n",
        fd, timedOut ? "timeout" : "error");

    // Adapt timeout based on result
    if (timedOut) {
        // Connection attempt used full timeout - increase it
        attemptTimeout = (attemptTimeout * 3) / 2; // Increase 50%
        printf("    Increasing timeout to %.1f sec\n",
            (float)attemptTimeout / TICKS_PER_SECOND);
    } else {
        // Failed quickly - keep current timeout
        printf("    Failed quickly (%.2f sec), keeping timeout\n",
            (float)usedTime / TICKS_PER_SECOND);
    }

    // Cap maximum timeout
    if (attemptTimeout > TICKS_PER_SECOND * 20) {
        attemptTimeout = TICKS_PER_SECOND * 20;
    }

    // Brief pause before retry
    if (!timeout.Expired()) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

printf("All attempts exhausted\n");
return -1;
}

```

Health Monitoring with Timeout Tracking

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>
#include <nbrtos.h>

struct InterfaceHealth {
    bool operational;
    uint32_t lastSuccessTime;
    uint32_t avgConnectTime;
    int consecutiveFailures;
};

void MonitorInterfaceHealth(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR testServer = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    InterfaceHealth health[4] = {0}; // Support up to 4 interfaces

    printf("Starting interface health monitoring\n");
    printf("Monitoring %d interfaces\n\n", numInterfaces);

    // Monitor for 10 cycles
    for (int cycle = 0; cycle < 10; cycle++) {
        printf("=== Health Check Cycle %d ===\n", cycle + 1);

        for (int i = 0; i < numInterfaces; i++) {
            TickTimeout timeout(TICKS_PER_SECOND * 5);

            int fd = connectvia(testServer, 8080, timeout, i);

            if (fd > 0) {
                uint32_t connectTime = timeout.GetElapsedTime();

                health[i].operational = true;
                health[i].lastSuccessTime = Secs;
                health[i].consecutiveFailures = 0;

                // Update running average
                if (health[i].avgConnectTime == 0) {
                    health[i].avgConnectTime = connectTime;
                } else {
                    health[i].avgConnectTime =

```

```

        (health[i].avgConnectTime + connectTime) / 2;
    }

    printf("IF%d: OK (%.2f sec, avg=%.2f sec)\n",
        i,
        (float)connectTime / TICKS_PER_SECOND,
        (float)health[i].avgConnectTime / TICKS_PER_SECOND);

    close(fd);
} else {
    health[i].consecutiveFailures++;

    if (health[i].consecutiveFailures >= 3) {
        health[i].operational = false;
    }

    printf("IF%d: FAIL (error %d, %d consecutive)\n",
        i, fd, health[i].consecutiveFailures);
}

}

// Status summary
int healthyCount = 0;
for (int i = 0; i < numInterfaces; i++) {
    if (health[i].operational) healthyCount++;
}

printf("Healthy interfaces: %d/%d\n\n",
    healthyCount, numInterfaces);

OSTimeDly(TICKS_PER_SECOND * 10);
}

printf("Health monitoring complete\n");
}

```

See also

[connectvia\(const IPADDR&, uint16_t, uint32_t, int\)](#), [connectvia\(const IPADDR&, uint16_t, TickTimeout&, const IPADDR&\)](#), [connectwlocal\(\)](#), [NoBlockConnectVia\(\)](#), [GetNumberOfInterfaces\(\)](#), [TcpGetSocketInterface\(\)](#), [TickTimeout](#)

14.53.14.9 connectvia() [3/4]

```

int connectvia (
    const IPADDR & ipAddress,
    uint16_t remotePort,
    uint32_t timeout,
    const IPADDR & localIpAddress) [inline]

```

#include <tcp.h>

Establish a TCP connection using a specific local interface IP address.

Creates a new TCP socket and connects to the remote host using the specified local network interface identified by its IP address. This allows control over which interface is used when multiple network interfaces are available. The standard [connect\(\)](#) function uses the default interface selected by the routing table, while this function overrides that decision.

This is particularly useful in multi-homed systems where you need to ensure traffic uses a specific network path, such as separating control and data traffic, routing through different network segments, or implementing failover between interfaces.

Parameters

<i>ipAddress</i>	IP address of the remote host. Can be IPv4 or IPv6 depending on build configuration
<i>remotePort</i>	Port number on the remote host (1-65535)
<i>timeout</i>	Maximum connection time in system ticks: • Use <code>TICKS_PER_SECOND * n</code> for n second timeout • Example: <code>TICKS_PER_SECOND * 10</code> for 10 second timeout • Recommended: 5-30 seconds for most applications

Parameters

<i>localIpAddress</i>	IP address of the local interface to use for this connection. The system will select the network interface that has this IP address assigned. If multiple interfaces share the same IP (unusual configuration), the lower numbered interface is selected. Use IPADDR::NullIP() to fall back to automatic interface selection based on routing table (equivalent to calling connect())

Returns

Positive file descriptor on success (use for read/write operations), negative [TCP Socket Status](#) error code on failure:

- `TCP_ERR_TIMEOUT`: Connection not established within timeout period
- `TCP_ERR_NONE_AVAIL`: No available sockets in system pool
- `TCP_ERR_CON_RESET`: Connection was reset during establishment
- `TCP_ERR_CON_ABORT`: Connection was aborted

Notes:

- The local port is automatically assigned from the ephemeral port range. This function does not allow specification of the local port number.
- If the specified local IP address does not exist on any interface, the behavior depends on the network stack implementation. Generally, the connection will fail or fall back to the default interface.
- Use [GetInterfaceIP\(\)](#) to obtain the IP address of a specific interface number before calling this function.
- Always call [close\(\)](#) on the returned file descriptor when finished to free system resources.
- This function blocks the calling task until the connection is established or the timeout expires. For non-blocking connection attempts, use [NoBlockConnectVia\(\)](#).

Warning

Ensure the specified local IP address is actually assigned to an interface before calling this function. Use [GetInterfaceIP\(\)](#) to verify.

Expand for Example Usage

Examples

Basic Interface Selection

```
#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void ConnectViaInterface(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Get IP address of interface 0
    IPADDR localIP = GetInterfaceIP(0);
    printf("Using local interface: %s\n", localIP.print());

    // Connect to server using specific interface
    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int fd = connectvia(serverAddr, 8080, TICKS_PER_SECOND * 10, localIP);

    if (fd > 0) {
        printf("Connected successfully!\n");
        printf("  Local endpoint: %s:%d\n",
            GetSocketLocalAddr(fd).print(),
            GetSocketLocalPort(fd));
        printf("  Remote endpoint: %s:%d\n",
            GetSocketRemoteAddr(fd).print(),
            GetSocketRemotePort(fd));

        // Verify we used the correct interface
        if (GetSocketLocalAddr(fd) == localIP) {
            printf("  Confirmed: Using requested interface\n");
        }
    }
}
```

```

    }

    close(fd);
} else {
    printf("Connection failed: %d\n", fd);
}
}

```

Dual-Homed System - Separate Control and Data Networks

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// System with two network interfaces:
// Interface 0: 192.168.1.x (Control network)
// Interface 1: 10.0.0.x (Data network)

void DualHomedExample(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Get interface IP addresses
    IPADDR controlIP = GetInterfaceIP(0); // Control network
    IPADDR dataIP = GetInterfaceIP(1);    // Data network

    printf("Control network IP: %s\n", controlIP.print());
    printf("Data network IP:    %s\n", dataIP.print());

    // Connect to control server via control network
    IPADDR controlServer = AsciiToIp("192.168.1.50");
    printf("\nConnecting to control server via control network...\n");

    int fdControl = connectvia(controlServer, 9000,
                               TICKS_PER_SECOND * 10,
                               controlIP);

    if (fdControl > 0) {
        printf("Control connection established via %s\n",
              GetSocketLocalAddr(fdControl).print());

        // Send control commands
        const char *cmd = "STATUS\r\n";
        write(fdControl, cmd, strlen(cmd));
    }

    // Connect to data server via data network
    IPADDR dataServer = AsciiToIp("10.0.0.50");
    printf("\nConnecting to data server via data network...\n");

    int fdData = connectvia(dataServer, 8080,
                           TICKS_PER_SECOND * 10,
                           dataIP);

    if (fdData > 0) {
        printf("Data connection established via %s\n",
              GetSocketLocalAddr(fdData).print());

        // Transfer bulk data
        char buffer[1024];
        // ... data transfer ...
    }

    // Clean up
    if (fdControl > 0) close(fdControl);
    if (fdData > 0)   close(fdData);

    printf("\nDual-homed operation complete\n");
}

```

Interface Failover

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Try primary interface first, failover to backup if unavailable
int ConnectWithFailover(IPADDR serverAddr, uint16_t port) {
    // Try primary interface (interface 0)
    IPADDR primaryIP = GetInterfaceIP(0);
    printf("Attempting connection via primary interface %s...\n",
          primaryIP.print());

    int fd = connectvia(serverAddr, port, TICKS_PER_SECOND * 5, primaryIP);

    if (fd > 0) {
        printf("Connected via primary interface\n");
    }
}

```

```

        return fd;
    }

    printf("Primary interface failed: %d\n", fd);

    // Try backup interface (interface 1)
    IPADDR backupIP = GetInterfaceIP(1);
    printf("Attempting connection via backup interface %s...\n",
        backupIP.print());

    fd = connectvia(serverAddr, port, TICKS_PER_SECOND * 5, backupIP);

    if (fd > 0) {
        printf("Connected via backup interface\n");
        return fd;
    }

    printf("Backup interface also failed: %d\n", fd);
    printf("All interfaces exhausted\n");

    return fd;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int fd = ConnectWithFailover(serverAddr, 8080);

    if (fd > 0) {
        // Use connection...
        close(fd);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Round-Robin Interface Usage

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void RoundRobinConnections(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    printf("System has %d network interfaces\n", numInterfaces);

    // Display all interface IPs
    for (int i = 0; i < numInterfaces; i++) {
        IPADDR ifIP = GetInterfaceIP(i);
        printf(" Interface %d: %s\n", i, ifIP.print());
    }

    // Make multiple connections, rotating through interfaces
    printf("\nMaking 10 connections with round-robin interface selection:\n");

    for (int conn = 0; conn < 10; conn++) {
        int interfaceNum = conn % numInterfaces;
        IPADDR localIP = GetInterfaceIP(interfaceNum);

        printf("\nConnection %d using interface %d (%s)...",
            conn + 1, interfaceNum, localIP.print());

        int fd = connectvia(serverAddr, 8080, TICKS_PER_SECOND * 10, localIP);

        if (fd > 0) {
            printf(" Success\n");
            printf(" Actual local IP: %s\n",
                GetSocketLocalAddr(fd).print());
            printf(" Local port: %d\n",
                GetSocketLocalPort(fd));

            // Do work...
            OSTimeDly(TICKS_PER_SECOND);

            close(fd);
        } else {
            printf(" Failed: %d\n", fd);
        }
    }
}

```

```

    }
}

```

Verifying Interface Selection

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void VerifyInterfaceSelection(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Test each available interface
    int numInterfaces = GetNumberOfInterfaces();

    for (int i = 0; i < numInterfaces; i++) {
        IPADDR requestedIP = GetInterfaceIP(i);

        printf("\n=== Testing Interface %d ===\n", i);
        printf("Requested local IP: %s\n", requestedIP.print());

        int fd = connectvia(serverAddr, 8080,
                           TICKS_PER_SECOND * 10,
                           requestedIP);

        if (fd > 0) {
            IPADDR actualIP = GetSocketLocalAddr(fd);
            int actualInterface = TcpGetSocketInterface(fd);

            printf("Actual local IP: %s\n", actualIP.print());
            printf("Actual interface: %d\n", actualInterface);

            // Verify match
            if (actualIP == requestedIP && actualInterface == i) {
                printf("VERIFIED: Interface selection correct\n");
            } else {
                printf("WARNING: Interface mismatch!\n");
                printf("  Expected interface %d with IP %s\n",
                      i, requestedIP.print());
                printf("  Got interface %d with IP %s\n",
                      actualInterface, actualIP.print());
            }

            close(fd);
        } else {
            printf("Connection failed: %d\n", fd);
        }
    }
}

```

Comparing Automatic vs Explicit Interface Selection

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void CompareInterfaceSelection(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Test 1: Automatic interface selection (standard connect)
    printf("=== Test 1: Automatic Interface Selection ===\n");
    int fd1 = connect(serverAddr, 8080, TICKS_PER_SECOND * 10);

    if (fd1 > 0) {
        printf("Connected via automatic selection\n");
        printf("  Selected interface: %d\n", TcpGetSocketInterface(fd1));
        printf("  Local IP: %s\n", GetSocketLocalAddr(fd1).print());
        printf("  (Determined by routing table)\n");
        close(fd1);
    }

    OSTimeDly(TICKS_PER_SECOND);

    // Test 2: Explicit interface selection
    printf("\n=== Test 2: Explicit Interface Selection ===\n");
    IPADDR explicitIP = GetInterfaceIP(0);
    printf("Requesting interface with IP: %s\n", explicitIP.print());

    int fd2 = connectvia(serverAddr, 8080,
                        TICKS_PER_SECOND * 10,

```

```

        explicitIP);

    if (fd2 > 0) {
        printf("Connected via explicit selection\n");
        printf("  Requested interface: %s\n", explicitIP.print());
        printf("  Selected interface: %d\n", TcpGetSocketInterface(fd2));
        printf("  Local IP: %s\n", GetSocketLocalAddr(fd2).print());
        close(fd2);
    }

    OSTimeDly(TICKS_PER_SECOND);

    // Test 3: Using NullIP (equivalent to automatic)
    printf("\n=== Test 3: connectvia with NullIP ===\n");
    printf("Using IPADDR::NullIP() (should be like automatic)\n");

    int fd3 = connectvia(serverAddr, 8080,
                        TICKS_PER_SECOND * 10,
                        IPADDR::NullIP());

    if (fd3 > 0) {
        printf("Connected\n");
        printf("  Selected interface: %d\n", TcpGetSocketInterface(fd3));
        printf("  Local IP: %s\n", GetSocketLocalAddr(fd3).print());
        close(fd3);
    }

    printf("\nComparison complete\n");
}

```

Multi-Interface Network Monitor

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>
#include <nbrtos.h>

// Monitor connectivity to a server via all interfaces
void MonitorInterfaceConnectivity(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    uint16_t port = 8080;
    int numInterfaces = GetNumberOfInterfaces();

    printf("Monitoring connectivity to %s:%d via %d interfaces\n\n",
           serverAddr.print(), port, numInterfaces);

    // Monitor loop
    for (int cycle = 0; cycle < 5; cycle++) {
        printf("=== Check Cycle %d ===\n", cycle + 1);

        for (int i = 0; i < numInterfaces; i++) {
            IPADDR localIP = GetInterfaceIP(i);

            printf("Interface %d (%s): ", i, localIP.print());

            uint32_t startTime = Secs;
            int fd = connectvia(serverAddr, port, TICKS_PER_SECOND * 5, localIP);
            uint32_t elapsed = Secs - startTime;

            if (fd > 0) {
                printf("OK (%lu sec)\n", elapsed);
                close(fd);
            } else {
                printf("FAILED (error %d after %lu sec)\n", fd, elapsed);
            }
        }

        printf("\n");
        OSTimeDly(TICKS_PER_SECOND * 10); // Wait 10 seconds between checks
    }

    printf("Monitoring complete\n");
}

```

Source Address Selection for Multi-Subnet Access

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Connect to servers on different subnets using appropriate interfaces
void MultiSubnetAccess(void *pd) {
    init();

```

```

    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Interface 0: 192.168.1.x/24
    // Interface 1: 10.0.0.x/8

    IPADDR interface0_IP = GetInterfaceIP(0);
    IPADDR interface1_IP = GetInterfaceIP(1);

    printf("Interface 0 IP: %s\n", interface0_IP.print());
    printf("Interface 1 IP: %s\n", interface1_IP.print());

    // Server on subnet 192.168.1.x - use interface 0
    printf("\nConnecting to server on 192.168.1.x subnet...\n");
    IPADDR server1 = AsciiToIp("192.168.1.50");
    int fd1 = connectvia(server1, 8080, TICKS_PER_SECOND * 10, interface0_IP);

    if (fd1 > 0) {
        printf("Connected to %s via %s\n",
            server1.print(),
            GetSocketLocalAddr(fd1).print());

        // Communicate with server1...
        close(fd1);
    }

    // Server on subnet 10.x.x.x - use interface 1
    printf("\nConnecting to server on 10.x.x.x subnet...\n");
    IPADDR server2 = AsciiToIp("10.0.1.50");
    int fd2 = connectvia(server2, 8080, TICKS_PER_SECOND * 10, interface1_IP);

    if (fd2 > 0) {
        printf("Connected to %s via %s\n",
            server2.print(),
            GetSocketLocalAddr(fd2).print());

        // Communicate with server2...
        close(fd2);
    }

    printf("\nMulti-subnet access complete\n");
}

```

See also

[connect\(\)](#), [connectvia\(const IPADDR&, uint16_t, uint32_t, int\)](#), [connectwlocal\(\)](#), [NoBlockConnectVia\(\)](#), [GetInterfaceIP\(\)](#), [TcpGetSocketInterface\(\)](#)

14.53.14.10 connectvia() [4/4]

```

int connectvia (
    const IPADDR & ipAddress,
    uint16_t remotePort,
    uint32_t timeout,
    int ifnum) [inline]

```

#include <tcp.h>

Establish a TCP connection using a specific local interface number.

Creates a new TCP socket and connects to the remote host using the specified local network interface number. This allows direct control over which interface is used, completely overriding the routing table. The interface number takes precedence over any IP-based routing decisions, providing the most explicit control over interface selection. This is the most direct way to control interface selection in multi-homed systems. Unlike selecting by IP address, specifying an interface number eliminates any ambiguity about which interface will be used, even if multiple interfaces share similar IP configurations.

Parameters

<i>ipAddress</i>	IP address of the remote host. Can be IPv4 or IPv6 depending on build configuration
<i>remotePort</i>	Port number on the remote host (1-65535)

Parameters

<i>timeout</i>	Maximum connection time in system ticks: • Use <code>TICKS_PER_SECOND * n</code> for n second timeout • Example: <code>TICKS_PER_SECOND * 10</code> for 10 second timeout • Recommended: 5-30 seconds for most applications
<i>ifnum</i>	Local interface number (0, 1, 2, etc.) to use for this connection. This overrides all routing decisions based on destination IP addresses. The interface number directly corresponds to the physical or logical network interface. Valid range is 0 to (number of interfaces - 1). Use <code>GetNumberOfInterfaces()</code> to determine valid range

Returns

Positive file descriptor on success (use for read/write operations), negative [TCP Socket Status](#) error code on failure:

- `TCP_ERR_TIMEOUT`: Connection not established within timeout period
- `TCP_ERR_NONE_AVAIL`: No available sockets in system pool
- `TCP_ERR_CON_RESET`: Connection was reset during establishment
- `TCP_ERR_CON_ABORT`: Connection was aborted

Notes:

- The local port is automatically assigned from the ephemeral port range. This function does not allow specification of the local port number.
- Interface numbers typically start at 0 and increment sequentially. Interface 0 is usually the primary Ethernet port, but this can vary by platform and configuration.
- If you specify an interface number that doesn't exist, the behavior is undefined. Always verify the interface count with `GetNumberOfInterfaces()` before using this function.
- This method of interface selection is preferred over IP-based selection when you need guaranteed, unambiguous interface control.
- Always call [close\(\)](#) on the returned file descriptor when finished to free system resources.
- This function blocks the calling task until the connection is established or the timeout expires. For non-blocking connection attempts, use [NoBlockConnectVia\(\)](#).

Warning

- Specifying an invalid interface number (negative or $\geq$ number of interfaces) may cause unpredictable behavior or connection failure.
- The specified interface must be in an "up" state with a valid IP address assigned for the connection to succeed.

Expand for Example Usage

Examples

Basic Interface Number Selection

```
#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void ConnectViaInterfaceNumber(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Connect using interface 1
    int interfaceNum = 1;

    printf("Connecting via interface %d\n", interfaceNum);
    printf("Interface %d IP: %s\n",
        interfaceNum, GetInterfaceIP(interfaceNum).print());

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int fd = connectvia(serverAddr, 8080, TICKS_PER_SECOND * 10, interfaceNum);

    if (fd > 0) {
        printf("Connected successfully!\n");
        printf(" Interface used: %d\n", TcpGetSocketInterface(fd));
        printf("  Local address: %s:%d\n",
            GetSocketLocalAddr(fd).print(),
            GetSocketLocalPort(fd));
        printf(" Remote address: %s:%d\n",
            GetSocketRemoteAddr(fd).print(),
            GetSocketRemotePort(fd));

        // Verify correct interface was used
        if (TcpGetSocketInterface(fd) == interfaceNum) {
            printf(" Verified: Correct interface selected\n");
        }

        close(fd);
    } else {
        printf("Connection failed: %d\n", fd);
    }
}
```

Iterating Through All Interfaces

```
#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void TestAllInterfaces(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    printf("System has %d network interfaces\n\n", numInterfaces);

    // Try connecting via each interface
    for (int i = 0; i < numInterfaces; i++) {
        IPADDR ifAddr = GetInterfaceIP(i);

        printf("Testing Interface %d (%s)...", i, ifAddr.print());

        uint32_t startTime = Secs;
        int fd = connectvia(serverAddr, 8080, TICKS_PER_SECOND * 5, i);
        uint32_t elapsed = Secs - startTime;

        if (fd > 0) {
            printf(" SUCCESS (%lu sec)\n", elapsed);
            printf(" Confirmed interface: %d\n", TcpGetSocketInterface(fd));
            printf(" Local endpoint: %s:%d\n",
                GetSocketLocalAddr(fd).print(),
                GetSocketLocalPort(fd));
            close(fd);
        } else {
            printf(" FAILED (error %d after %lu sec)\n", fd, elapsed);
        }

        printf("\n");
    }

    printf("Interface testing complete\n");
}
```

Primary and Backup Interface with Automatic Failover

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Connect with automatic failover from primary to backup interface
int ConnectWithInterfaceFailover(IPADDR serverAddr, uint16_t port,
                                int primaryInterface, int backupInterface) {
    // Try primary interface first
    printf("Attempting connection via primary interface %d (%s)...\n",
           primaryInterface,
           GetInterfaceIP(primaryInterface).print());

    int fd = connectvia(serverAddr, port, TICKS_PER_SECOND * 5, primaryInterface);

    if (fd > 0) {
        printf("Connected via primary interface\n");
        return fd;
    }

    printf("Primary interface failed: %d\n", fd);
    printf("Failing over to backup interface %d (%s)...\n",
           backupInterface,
           GetInterfaceIP(backupInterface).print());

    fd = connectvia(serverAddr, port, TICKS_PER_SECOND * 5, backupInterface);

    if (fd > 0) {
        printf("Connected via backup interface\n");
        return fd;
    }

    printf("Backup interface also failed: %d\n", fd);
    printf("All interfaces exhausted\n");

    return fd;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Try interface 0 first, failover to interface 1
    int fd = ConnectWithInterfaceFailover(serverAddr, 8080, 0, 1);

    if (fd > 0) {
        printf("Connection established via interface %d\n",
               TcpGetSocketInterface(fd));

        // Use connection...
        close(fd);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Interface Validation Before Connection

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Safely connect via interface with validation
int SafeConnectViaInterface(IPADDR serverAddr, uint16_t port, int ifnum) {
    // Validate interface number
    int maxInterfaces = GetNumberOfInterfaces();

    if (ifnum < 0 || ifnum >= maxInterfaces) {
        printf("ERROR: Invalid interface number %d\n", ifnum);
        printf("Valid range is 0 to %d\n", maxInterfaces - 1);
        return -1;
    }

    // Get interface information
    IPADDR ifAddr = GetInterfaceIP(ifnum);

    // Check if interface has valid IP
    if (ifAddr == IPADDR::NullIP()) {
        printf("ERROR: Interface %d has no IP address assigned\n", ifnum);
        return -1;
    }
}

```

```

// Check if interface is up (platform-specific, may need additional checks)
printf("Interface %d status:\n", ifnum);
printf(" IP Address: %s\n", ifAddr.print());

// Proceed with connection
printf("Connecting via interface %d...\n", ifnum);
int fd = connectvia(serverAddr, port, TICKS_PER_SECOND * 10, ifnum);

if (fd > 0) {
    printf("Connection successful\n");
} else {
    printf("Connection failed: %d\n", fd);
}

return fd;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    int fd = SafeConnectViaInterface(serverAddr, 8080, 1);

    if (fd > 0) {
        close(fd);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Load Distribution Across Interfaces

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Distribute connections evenly across available interfaces
void LoadBalancedConnections(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    printf("Load balancing across %d interfaces\n\n", numInterfaces);

    // Connection counters for each interface
    int connectionCounts[4] = {0}; // Support up to 4 interfaces
    int totalConnections = 20;
    int successfulConnections = 0;

    for (int i = 0; i < totalConnections; i++) {
        // Round-robin interface selection
        int interfaceNum = i % numInterfaces;

        printf("Connection %d via interface %d...", i + 1, interfaceNum);

        int fd = connectvia(serverAddr, 8080, TICKS_PER_SECOND * 10, interfaceNum);

        if (fd > 0) {
            printf(" OK\n");
            connectionCounts[interfaceNum]++;
            successfulConnections++;

            // Simulate some work
            const char *msg = "PING\r\n";
            write(fd, msg, strlen(msg));

            close(fd);
        } else {
            printf(" FAILED: %d\n", fd);
        }

        OSTimeDly(TICKS_PER_SECOND / 2); // Small delay between connections
    }

    // Display statistics
    printf("\n=== Connection Statistics ===\n");
    printf("Total attempts: %d\n", totalConnections);
    printf("Successful: %d\n", successfulConnections);
    printf("Failed: %d\n\n", totalConnections - successfulConnections);

    printf("Per-Interface Distribution:\n");
}

```

```

    for (int i = 0; i < numInterfaces; i++) {
        printf("    Interface %d: %d connections (%.1f%%)\n",
            i,
            connectionCounts[i],
            (connectionCounts[i] * 100.0) / successfulConnections);
    }
}

```

Interface-Specific Connection Pools

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

#define MAX_CONNECTIONS_PER_INTERFACE 5

// Manage separate connection pools for each interface
struct InterfacePool {
    int interfaceNum;
    int connections[MAX_CONNECTIONS_PER_INTERFACE];
    int activeCount;
};

void InitPool(InterfacePool *pool, int ifnum) {
    pool->interfaceNum = ifnum;
    pool->activeCount = 0;
    for (int i = 0; i < MAX_CONNECTIONS_PER_INTERFACE; i++) {
        pool->connections[i] = -1;
    }
}

int AddConnection(InterfacePool *pool, IPADDR serverAddr, uint16_t port) {
    if (pool->activeCount >= MAX_CONNECTIONS_PER_INTERFACE) {
        printf("Interface %d pool full\n", pool->interfaceNum);
        return -1;
    }

    int fd = connectvia(serverAddr, port,
        TICKS_PER_SECOND * 10,
        pool->interfaceNum);

    if (fd > 0) {
        pool->connections[pool->activeCount++] = fd;
        printf("Added connection to interface %d pool (now %d active)\n",
            pool->interfaceNum, pool->activeCount);
    }

    return fd;
}

void CloseAllConnections(InterfacePool *pool) {
    printf("Closing all connections on interface %d...\n", pool->interfaceNum);

    for (int i = 0; i < pool->activeCount; i++) {
        if (pool->connections[i] > 0) {
            close(pool->connections[i]);
            pool->connections[i] = -1;
        }
    }

    pool->activeCount = 0;
    printf("All connections closed on interface %d\n", pool->interfaceNum);
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Create pools for two interfaces
    InterfacePool pool0, pool1;
    InitPool(&pool0, 0);
    InitPool(&pool1, 1);

    // Add connections to each pool
    printf("Building connection pools...\n");

    for (int i = 0; i < 3; i++) {
        AddConnection(&pool0, serverAddr, 8080);
        AddConnection(&pool1, serverAddr, 8080);
    }

    printf("\nPools established\n");
    printf("Interface 0: %d connections\n", pool0.activeCount);
    printf("Interface 1: %d connections\n", pool1.activeCount);
}

```

```

// Use connections...
OSTimeDly(TICKS_PER_SECOND * 5);

// Clean up
CloseAllConnections(&pool0);
CloseAllConnections(&pool1);

while (1) {
    OSTimeDly(TICKS_PER_SECOND);
}
}

```

Comparing Interface Selection Methods

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void CompareSelectionMethods(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    IPADDR interface1IP = GetInterfaceIP(1);

    printf("Comparing interface selection methods for interface 1\n");
    printf("Interface 1 IP: %s\n\n", interface1IP.print());

    // Method 1: By interface number (this function)
    printf("Method 1: Selection by interface number\n");
    uint32_t start1 = TimeTick;
    int fd1 = connectvia(serverAddr, 8080, TICKS_PER_SECOND * 10, 1);
    uint32_t time1 = TimeTick - start1;

    if (fd1 > 0) {
        printf(" Success in %lu ticks\n", time1);
        printf(" Interface: %d\n", TcpGetSocketInterface(fd1));
        printf(" Local IP: %s\n", GetSocketLocalAddr(fd1).print());
        close(fd1);
    } else {
        printf(" Failed: %d\n", fd1);
    }

    OSTimeDly(TICKS_PER_SECOND);

    // Method 2: By IP address (overloaded version)
    printf("\nMethod 2: Selection by IP address\n");
    uint32_t start2 = TimeTick;
    int fd2 = connectvia(serverAddr, 8080, TICKS_PER_SECOND * 10, interface1IP);
    uint32_t time2 = TimeTick - start2;

    if (fd2 > 0) {
        printf(" Success in %lu ticks\n", time2);
        printf(" Interface: %d\n", TcpGetSocketInterface(fd2));
        printf(" Local IP: %s\n", GetSocketLocalAddr(fd2).print());
        close(fd2);
    } else {
        printf(" Failed: %d\n", fd2);
    }

    // Analysis
    printf("\n=== Analysis ===\n");
    printf("Interface number method is more direct and unambiguous\n");
    printf("IP address method requires IP lookup step\n");
    printf("Both methods should select the same interface\n");

    if (fd1 > 0 && fd2 > 0) {
        printf("\nTiming comparison:\n");
        printf(" By number: %lu ticks\n", time1);
        printf(" By IP: %lu ticks\n", time2);
        printf(" Difference: %ld ticks\n", (long)(time2 - time1));
    }
}

```

Interface Health Monitoring

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>
#include <nbrtos.h>

// Monitor health of each interface by attempting periodic connections
void MonitorInterfaceHealth(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);
}

```

```

IPADDR testServer = AsciiToIp("192.168.1.100");
int numInterfaces = GetNumberOfInterfaces();

// Health tracking
struct InterfaceHealth {
    int successCount;
    int failureCount;
    uint32_t lastSuccessTime;
    uint32_t lastFailureTime;
} health[4] = {0}; // Support up to 4 interfaces

printf("Starting interface health monitoring\n");
printf("Test server: %s\n", testServer.print());
printf("Interfaces: %d\n\n", numInterfaces);

for (int cycle = 0; cycle < 10; cycle++) {
    printf("=== Health Check Cycle %d ===\n", cycle + 1);

    for (int i = 0; i < numInterfaces; i++) {
        int fd = connectvia(testServer, 8080, TICKS_PER_SECOND * 3, i);

        if (fd > 0) {
            health[i].successCount++;
            health[i].lastSuccessTime = Secs;
            printf("Interface %d: HEALTHY\n", i);
            close(fd);
        } else {
            health[i].failureCount++;
            health[i].lastFailureTime = Secs;
            printf("Interface %d: UNHEALTHY (error %d)\n", i, fd);
        }
    }

    // Display statistics
    printf("\nCurrent Statistics:\n");
    for (int i = 0; i < numInterfaces; i++) {
        int total = health[i].successCount + health[i].failureCount;
        float successRate = (total > 0) ?
            (health[i].successCount * 100.0f) / total : 0.0f;

        printf("  Interface %d: %.1f%% success rate (%d/%d)\n",
            i, successRate, health[i].successCount, total);
    }

    printf("\n");
    OSTimeDly(TICKS_PER_SECOND * 5);
}

printf("Health monitoring complete\n");
}

```

Dynamic Interface Selection Based on Policy

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

enum ConnectionPolicy {
    POLICY_PRIMARY_ONLY,
    POLICY_ROUND_ROBIN,
    POLICY_LEAST_LOADED,
    POLICY_RANDOM
};

int connectionCounts[4] = {0}; // Track connections per interface
int currentRRInterface = 0; // Round-robin counter

int SelectInterface(ConnectionPolicy policy, int numInterfaces) {
    switch (policy) {
        case POLICY_PRIMARY_ONLY:
            return 0; // Always use interface 0

        case POLICY_ROUND_ROBIN:
            {
                int selected = currentRRInterface;
                currentRRInterface = (currentRRInterface + 1) % numInterfaces;
                return selected;
            }

        case POLICY_LEAST_LOADED:
            {
                int minCount = connectionCounts[0];
                int minIndex = 0;

                for (int i = 1; i < numInterfaces; i++) {
                    if (connectionCounts[i] < minCount) {
                        minCount = connectionCounts[i];
                    }
                }
                return minIndex;
            }
    }
}

```

```

        minIndex = i;
    }
    }
    return minIndex;
}

case POLICY_RANDOM:
    return rand() % numInterfaces;

default:
    return 0;
}
}

void PolicyBasedConnections(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    ConnectionPolicy policy = POLICY_ROUND_ROBIN;

    printf("Using policy: ROUND_ROBIN\n");
    printf("Making 10 connections...\n\n");

    for (int i = 0; i < 10; i++) {
        int selectedIF = SelectInterface(policy, numInterfaces);

        printf("Connection %d: Selected interface %d\n", i + 1, selectedIF);

        int fd = connectvia(serverAddr, 8080, TICKS_PER_SECOND * 10, selectedIF);

        if (fd > 0) {
            connectionCounts[selectedIF]++;
            printf(" Success via interface %d\n", selectedIF);
            close(fd);
        } else {
            printf(" Failed: %d\n", fd);
        }
    }

    printf("\n=== Final Distribution ===\n");
    for (int i = 0; i < numInterfaces; i++) {
        printf("Interface %d: %d connections\n", i, connectionCounts[i]);
    }
}

```

See also

[connect\(\)](#), [connectvia\(const IPADDR&, uint16_t, uint32_t, const IPADDR&\)](#), [connectwlocal\(\)](#), [NoBlockConnectVia\(\)](#), [GetNumberOfInterfaces\(\)](#), [GetInterfaceIP\(\)](#), [TcpGetSocketInterface\(\)](#)

14.53.14.11 connectwlocal() [1/2]

```

int connectwlocal (
    const IPADDR & ipAddress,
    uint16_t localPort,
    uint16_t remotePort,
    TickTimeout timeout,
    const IPADDR & localIpAddress = IPADDR::NullIP(),
    int intf = -1) [inline]

#include <tcp.h>

```

Establish a TCP connection with full control over local parameters and TickTimeout.

Creates a new TCP socket with complete control over both local and remote connection parameters, using a Tick↔Timeout object for sophisticated timeout management. This function allows specification of the local port, interface IP address, or interface number, combined with precise timeout tracking for complex connection sequences.

This is the most advanced connection function, providing maximum control over all connection parameters plus the ability to track elapsed time and share timeout budgets. Most applications should use simpler connection functions unless they have specific requirements for local port control, explicit interface selection, and advanced timeout management.

Warning

Specifying a local port number is not recommended and should be avoided except in specialized cases. Using a fixed local port can cause connection failures when the remote host has a half-open socket remaining from a previous connection.

Parameters

<i>ipAddress</i>	IP address of the remote host. Can be IPv4 or IPv6 depending on build configuration
<i>localPort</i>	Local port number to bind (1-65535, or 0 for automatic assignment). Strongly recommend using 0 unless you have a specific protocol requirement (e.g., FTP active mode). Non-zero values can cause TCP_ERR_NOCON or connection failures when reconnecting if the previous connection is in TIME_WAIT state on the remote side
<i>remotePort</i>	Port number on the remote host (1-65535)
<i>timeout</i>	TickTimeout object specifying the maximum wait time. Unlike the reference version, this is passed by value, so the original timeout object is not modified. The function creates an internal copy for timeout tracking
<i>localIpAddress</i>	IP address of the local interface to use. Selects the network interface with this IP address. Use <code>IPADDR::NullIP()</code> if you want to specify interface by number instead (intf parameter). Default: <code>IPADDR::NullIP()</code> (automatic selection or use intf)
<i>intf</i>	Local interface number (0, 1, 2, etc.). Use -1 if you want to specify interface by IP address instead (localIpAddress parameter). Interface number takes precedence over IP-based routing. Default: -1 (use IP-based routing or localIpAddress)

Returns

Positive file descriptor on success (use for read/write operations), negative [TCP Socket Status](#) error code on failure:

- TCP_ERR_TIMEOUT: Connection not established within timeout period
- TCP_ERR_NONE_AVAIL: No available sockets in system pool
- TCP_ERR_NOCON: Local port is already in use or address binding failed
- TCP_ERR_CON_RESET: Connection was reset during establishment
- TCP_ERR_CON_ABORT: Connection was aborted

Notes:

- Unlike the reference version (`TickTimeout&`), this function accepts `TickTimeout` by value, so the caller's timeout object is not modified. This is useful when you want to preserve the original timeout state.
- Parameter Priority:
 - If `intf >= 0`: Uses the specified interface number, ignores `localIpAddress`
 - If `intf == -1` and `localIpAddress != NullIP()`: Uses interface with that IP
 - If `intf == -1` and `localIpAddress == NullIP()`: Uses routing table
- Local Port Behavior:
 - `localPort == 0`: System assigns available ephemeral port (recommended)
 - `localPort != 0`: Attempts to bind to specific port (can fail if in use)
- Always call [close\(\)](#) on the returned file descriptor when finished to free system resources and release the local port.
- This function blocks the calling task until the connection is established or the timeout expires. For non-blocking connection attempts, use [NoBlockConnectwlocal\(\)](#).

Warning

The combination of local port, interface IP, and interface number must be logically consistent. Specifying conflicting parameters will result in unpredictable behavior.

Expand for Example Usage**Examples****Basic Usage with Preserved Timeout Object**

```
#include <tcp.h>
#include <init.h>

void ConnectWithPreservedTimeout(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Create timeout that we want to preserve
    TickTimeout timeout(TICKS_PER_SECOND * 15);

    printf("Original timeout: %lu ticks\n", timeout.GetTimeLeft());

    // Connect using value semantics (timeout is copied, not modified)
    int fd = connectwlocal(serverAddr,
                          0, // Auto-assign local port
                          8080, // Remote port
                          timeout, // Passed by value
                          IPADDR::NullIP(),
                          -1);

    // Original timeout object is unchanged
    printf("After connect, original timeout still: %lu ticks\n",
          timeout.GetTimeLeft());

    if (fd > 0) {
        printf("Connected successfully!\n");
        printf("Can reuse original timeout for other operations\n");
        close(fd);
    }
}
```

Multiple Connections with Same Timeout Template

```
#include <tcp.h>
#include <init.h>

// Make multiple connections using the same timeout template
void MultipleConnectionsSameTimeout(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR servers[] = {
        AsciiToIp("192.168.1.10"),
        AsciiToIp("192.168.1.20"),
        AsciiToIp("192.168.1.30")
    };

    // Create timeout template
    TickTimeout timeoutTemplate(TICKS_PER_SECOND * 10);

    printf("Connecting to 3 servers, each with fresh 10 sec timeout\n\n");

    for (int i = 0; i < 3; i++) {
        printf("Server %d (%s)...", i + 1, servers[i].print());

        // Each connection gets a fresh copy of the timeout
        int fd = connectwlocal(servers[i],
                              0,
                              8080,
                              timeoutTemplate, // Copy for each connection
                              IPADDR::NullIP(),
                              -1);

        if (fd > 0) {
            printf(" Connected\n");
            close(fd);
        } else {
            printf(" Failed: %d\n", fd);
        }
    }

    // Verify template is unchanged
    printf(" Template timeout still: %lu ticks\n",
          timeoutTemplate.GetTimeLeft());
}
```

```
    printf("\nAll connections attempted with independent timeouts\n");
}
```

Interface Selection by IP with TickTimeout

```
#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void ConnectViaSpecificInterface(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    IPADDR localIP = GetInterfaceIP(1); // Use interface 1

    printf("Connecting via interface %s\n", localIP.print());

    TickTimeout timeout(TICKS_PER_SECOND * 15);

    int fd = connectwlocal(serverAddr,
                           0, // Auto-assign port
                           8080, // Remote port
                           timeout, // By value
                           localIP, // Specific interface IP
                           -1); // Not using interface number

    if (fd > 0) {
        printf("Connected successfully!\n");
        printf("  Local: %s:%d\n",
               GetSocketLocalAddr(fd).print(),
               GetSocketLocalPort(fd));
        printf("  Remote: %s:%d\n",
               GetSocketRemoteAddr(fd).print(),
               GetSocketRemotePort(fd));

        // Verify correct interface used
        if (GetSocketLocalAddr(fd) == localIP) {
            printf("  Verified: Using requested interface\n");
        }

        close(fd);
    } else {
        printf("Connection failed: %d\n", fd);
    }
}
```

Interface Selection by Number with TickTimeout

```
#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void ConnectViaInterfaceNumber(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int interfaceNum = 0;

    printf("Connecting via interface %d (%s)\n",
           interfaceNum, GetInterfaceIP(interfaceNum).print());

    TickTimeout timeout(TICKS_PER_SECOND * 15);

    int fd = connectwlocal(serverAddr,
                           0, // Auto-assign port
                           8080, // Remote port
                           timeout, // By value
                           IPADDR::NullIP(), // Not using IP selection
                           interfaceNum); // Specific interface number

    if (fd > 0) {
        printf("Connected successfully!\n");
        printf("  Interface used: %d\n", TcpGetSocketInterface(fd));
        printf("  Local IP: %s\n", GetSocketLocalAddr(fd).print());
        close(fd);
    } else {
        printf("Connection failed: %d\n", fd);
    }
}
```

FTP Active Mode with Fixed Port

```
#include <tcp.h>
#include <init.h>

// FTP active mode requires fixed local port for data connection
void FtpActiveModeConnection(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR ftpServer = AsciiToIp("192.168.1.10");

    // Step 1: Connect to FTP control port
    printf("Connecting to FTP control port...\n");
    TickTimeout controlTimeout(TICKS_PER_SECOND * 10);

    int fdControl = connectwlocal(ftpServer,
                                0, // Auto-assign
                                21, // FTP control
                                controlTimeout,
                                IPADDR::NullIP(),
                                -1);

    if (fdControl < 0) {
        printf("Control connection failed: %d\n", fdControl);
        return;
    }

    printf("FTP control connected\n");

    // Step 2: For FTP active mode, we need a listening socket
    // This is one of the few legitimate uses of fixed local port
    uint16_t dataPort = 2000;
    int fdDataListen = listen(INADDR_ANY, dataPort, 1);

    if (fdDataListen < 0) {
        printf("Data listen socket failed: %d\n", fdDataListen);
        close(fdControl);
        return;
    }

    printf("Data listen socket on port %d\n", dataPort);

    // Step 3: Tell server our IP and port
    IPADDR localIP = GetSocketLocalAddr(fdControl);
    char portCmd[128];
    uint8_t *ip = (uint8_t *)&localIP;
    snprintf(portCmd, sizeof(portCmd),
             "PORT %d,%d,%d,%d,%d,%d\r\n",
             ip[0], ip[1], ip[2], ip[3],
             dataPort >> 8, dataPort & 0xFF);

    write(fdControl, portCmd, strlen(portCmd));
    printf("Sent PORT command\n");

    // Step 4: Accept data connection from server
    printf("Waiting for server data connection...\n");
    IPADDR serverDataAddr;
    uint16_t serverDataPort;

    int fdData = accept(fdDataListen, &serverDataAddr, &serverDataPort,
                       TICKS_PER_SECOND * 30);

    if (fdData > 0) {
        printf("FTP data connection accepted from %s:%d\n",
              serverDataAddr.print(), serverDataPort);
        close(fdData);
    }

    close(fdDataListen);
    close(fdControl);

    printf("FTP active mode example complete\n");
}
```

Connection Pool with Per-Interface Timeout

```
#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

#define CONNECTIONS_PER_INTERFACE 3

struct ConnectionPool {
    int interfaceNum;
    int connections[CONNECTIONS_PER_INTERFACE];
    int activeCount;
```

```

};

void CreateInterfacePool(ConnectionPool *pool, int ifNum,
                        IPADDR serverAddr, uint16_t port) {
    pool->interfaceNum = ifNum;
    pool->activeCount = 0;

    IPADDR localIP = GetInterfaceIP(ifNum);
    TickTimeout timeoutTemplate(TICKS_PER_SECOND * 10);

    printf("Creating pool for interface %d (%s)\n",
           ifNum, localIP.print());

    for (int i = 0; i < CONNECTIONS_PER_INTERFACE; i++) {
        printf(" Connection %d...", i + 1);

        // Each connection gets fresh timeout (passed by value)
        int fd = connectwlocal(serverAddr,
                               0, // Auto port
                               port,
                               timeoutTemplate, // Copy
                               localIP, // Via this interface
                               -1);

        if (fd > 0) {
            pool->connections[pool->activeCount++] = fd;
            printf(" OK (port %d)\n", GetSocketLocalPort(fd));
        } else {
            pool->connections[i] = -1;
            printf(" Failed: %d\n", fd);
        }
    }

    printf("Pool created: %d/%d connections\n\n",
           pool->activeCount, CONNECTIONS_PER_INTERFACE);
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    ConnectionPool pools[4]; // Support up to 4 interfaces

    // Create pool for each interface
    for (int i = 0; i < numInterfaces; i++) {
        CreateInterfacePool(&pools[i], i, serverAddr, 8080);
    }

    // Use connections...
    OSTimeDly(TICKS_PER_SECOND * 5);

    // Clean up
    for (int i = 0; i < numInterfaces; i++) {
        for (int j = 0; j < pools[i].activeCount; j++) {
            close(pools[i].connections[j]);
        }
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Parallel Attempts Across Interfaces

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Try all interfaces in parallel with same timeout
void ParallelInterfaceAttempts(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = GetNumberOfInterfaces();

    // Create shared timeout template
    TickTimeout timeoutTemplate(TICKS_PER_SECOND * 8);

    printf("Attempting parallel connections via all interfaces\n");
    printf("Each attempt gets 8 second timeout\n\n");

    int fds[4] = {-1, -1, -1, -1}; // Store results

```

```

uint32_t startTime = Secs;

// Launch all attempts (in real implementation, would use tasks)
for (int i = 0; i < numInterfaces; i++) {
    IPADDR localIP = GetInterfaceIP(i);

    printf("Interface %d (%s): Connecting...\n",
           i, localIP.print());

    // Each gets independent timeout copy
    fds[i] = connectwlocal(serverAddr,
                           0,
                           8080,
                           timeoutTemplate, // Fresh copy
                           localIP,
                           -1);

    if (fds[i] > 0) {
        printf(" SUCCESS\n");
    } else {
        printf(" Failed: %d\n", fds[i]);
    }
}

uint32_t elapsed = Secs - startTime;

// Summary
printf("\n=== Results ===\n");
printf("Total time: %lu seconds\n", elapsed);

int successCount = 0;
for (int i = 0; i < numInterfaces; i++) {
    if (fds[i] > 0) {
        successCount++;
        printf("Interface %d: Connected (fd=%d)\n", i, fds[i]);
        close(fds[i]);
    } else {
        printf("Interface %d: Failed\n", i);
    }
}

printf("\nSuccess rate: %d/%d\n", successCount, numInterfaces);
}

```

Testing All Parameter Combinations

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

void TestAllParameterCombinations(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    IPADDR interface0IP = GetInterfaceIP(0);

    TickTimeout timeoutTemplate(TICKS_PER_SECOND * 10);

    printf("Testing connectwlocal parameter combinations\n\n");

    // Test 1: Auto port, auto interface
    printf("Test 1: Auto port, auto interface\n");
    int fd1 = connectwlocal(serverAddr, 0, 8080,
                           timeoutTemplate,
                           IPADDR::NullIP(), -1);

    if (fd1 > 0) {
        printf(" Result: %s:%d\n",
               GetSocketLocalAddr(fd1).print(),
               GetSocketLocalPort(fd1));
        close(fd1);
    }
    OSTimeDly(TICKS_PER_SECOND);

    // Test 2: Auto port, interface by IP
    printf("\nTest 2: Auto port, interface by IP (%s)\n",
           interface0IP.print());
    int fd2 = connectwlocal(serverAddr, 0, 8080,
                           timeoutTemplate,
                           interface0IP, -1);

    if (fd2 > 0) {
        printf(" Result: %s:%d\n",
               GetSocketLocalAddr(fd2).print(),
               GetSocketLocalPort(fd2));
        close(fd2);
    }
    OSTimeDly(TICKS_PER_SECOND);
}

```

```

// Test 3: Auto port, interface by number
printf("\nTest 3: Auto port, interface by number (0)\n");
int fd3 = connectwlocal(serverAddr, 0, 8080,
                       timeoutTemplate,
                       IPADDR::NullIP(), 0);

if (fd3 > 0) {
    printf("    Result: %s:%d\n",
           GetSocketLocalAddr(fd3).print(),
           GetSocketLocalPort(fd3));
    close(fd3);
}
OSTimeDly(TICKS_PER_SECOND);

// Test 4: Fixed port (NOT RECOMMENDED except for special cases)
printf("\nTest 4: Fixed port 7000 (NOT RECOMMENDED)\n");
int fd4 = connectwlocal(serverAddr, 7000, 8080,
                       timeoutTemplate,
                       IPADDR::NullIP(), -1);

if (fd4 > 0) {
    printf("    Result: %s:%d\n",
           GetSocketLocalAddr(fd4).print(),
           GetSocketLocalPort(fd4));
    close(fd4);
} else {
    printf("    Failed: %d (may be in use)\n", fd4);
}

printf("\nTesting complete\n");
}

```

Timeout Template Library Pattern

```

#include <tcp.h>
#include <init.h>

// Create library of reusable timeout templates
namespace TimeoutTemplates {
    TickTimeout Quick(TICKS_PER_SECOND * 3);           // 3 seconds
    TickTimeout Normal(TICKS_PER_SECOND * 10);         // 10 seconds
    TickTimeout Patient(TICKS_PER_SECOND * 30);        // 30 seconds
    TickTimeout VeryPatient(TICKS_PER_SECOND * 60);    // 60 seconds
};

void UseTimeoutTemplates(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    // Quick attempt for high-priority connection
    printf("Quick attempt (3 sec timeout)...\n");
    int fd1 = connectwlocal(serverAddr, 0, 8080,
                           TimeoutTemplates::Quick,
                           IPADDR::NullIP(), -1);

    if (fd1 < 0) {
        printf("Quick failed, trying normal timeout...\n");

        // Fall back to normal timeout
        int fd2 = connectwlocal(serverAddr, 0, 8080,
                                TimeoutTemplates::Normal,
                                IPADDR::NullIP(), -1);

        if (fd2 < 0) {
            printf("Normal failed, using patient timeout...\n");

            // Last resort with patient timeout
            int fd3 = connectwlocal(serverAddr, 0, 8080,
                                    TimeoutTemplates::Patient,
                                    IPADDR::NullIP(), -1);

            if (fd3 > 0) {
                printf("Connected with patient timeout\n");
                close(fd3);
            }
        } else {
            printf("Connected with normal timeout\n");
            close(fd2);
        }
    } else {
        printf("Connected with quick timeout\n");
        close(fd1);
    }

    // Template objects are preserved for reuse
    printf("\nTimeout templates ready for next use\n");
}

```

}

See also

[connectwlocal\(const IPADDR&, uint16_t, uint16_t, uint32_t, const IPADDR&, int\)](#), [connect\(\)](#), [connectvia\(\)](#), [NoBlockConnectwlocal\(\)](#), [TickTimeout](#)

14.53.14.12 connectwlocal() [2/2]

```
int connectwlocal (
    const IPADDR & ipAddress,
    uint16_t localPort,
    uint16_t remotePort,
    uint32_t timeout,
    const IPADDR & localIpAddress = IPADDR::NullIP(),
    int intf = -1) [inline]
#include <tcp.h>
```

Establish a TCP connection with full control over local parameters.

Creates a new TCP socket with complete control over both local and remote connection parameters. This function allows specification of the local port, interface IP address, or interface number. This is an advanced function intended for specialized use cases where standard [connect\(\)](#) or [connectvia\(\)](#) are insufficient.

Most applications should use [connect\(\)](#) or [connectvia\(\)](#) instead. This function is provided for protocols or scenarios that require specific local port numbers or explicit control over both the interface and local port simultaneously.

Warning

Specifying a local port number is not recommended and should be avoided except in specialized cases. Using a fixed local port can cause connection failures when the remote host has a half-open socket remaining from a previous connection. This is a common problem in client applications that reconnect frequently.

The combination of local port, interface IP, and interface number must be logically consistent. Specifying conflicting parameters will result in unpredictable behavior.

Parameters

<i>ipAddress</i>	IP address of the remote host. Can be IPv4 or IPv6 depending on build configuration
<i>localPort</i>	Local port number to bind (1-65535, or 0 for automatic assignment). Strongly recommend using 0 unless you have a specific protocol requirement. Non-zero values can cause TCP_ERR_NOCON or connection failures when reconnecting if the previous connection is in TIME_WAIT state on the remote side
<i>remotePort</i>	Port number on the remote host (1-65535)
<i>timeout</i>	Maximum connection time in system ticks: • Use TICKS_PER_SECOND * n for n second timeout • Example: TICKS_PER_SECOND * 10 for 10 second timeout • Recommended: 5-30 seconds for most applications
<i>localIpAddress</i>	IP address of the local interface to use. Selects the network interface with this IP address. Use IPADDR::NullIP() if you want to specify interface by number instead (intf parameter). Default: IPADDR::NullIP() (automatic selection)
<i>intf</i>	Local interface number (0, 1, 2, etc.). Use -1 if you want to specify interface by IP address instead (localIpAddress parameter). Interface number takes precedence over IP-based routing. Default: -1 (use IP-based routing or localIpAddress)

Returns

Positive file descriptor on success (use for read/write operations), negative [TCP Socket Status](#) error code on failure:

- TCP_ERR_TIMEOUT: Connection not established within timeout period
- TCP_ERR_NONE_AVAIL: No available sockets in system pool
- TCP_ERR_NOCON: Local port is already in use or address binding failed
- TCP_ERR_CON_RESET: Connection was reset during establishment
- TCP_ERR_CON_ABORT: Connection was aborted

Notes:

Parameter Priority:

- If intf $\geq$ 0: Uses the specified interface number, ignores localIpAddress
- If intf == -1 and localIpAddress != NullIP(): Uses interface with that IP
- If intf == -1 and localIpAddress == NullIP(): Uses routing table

Local Port Behavior:

- localPort == 0: System assigns available ephemeral port (recommended)
- localPort != 0: Attempts to bind to specific port (can fail if in use)

Common Use Cases:

- FTP active mode (requires specific local port for data connection)
- Protocol testing and debugging (need predictable source port)
- Load balancing across multiple interfaces
- Binding to specific interface when routing is ambiguous

Always call [close\(\)](#) on the returned file descriptor when finished to free system resources and release the local port.
Expand for Example Usage

Examples**Basic Usage - Interface Selection by IP**

```
#include <tcp.h>
#include <init.h>

void ConnectViaSpecificInterface(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    IPADDR localAddr = GetInterfaceIP(0); // Use interface 0's IP

    printf("Connecting to %s via local interface %s\n",
           serverAddr.print(), localAddr.print());

    // Connect using specific local interface, automatic port
    int fd = connectwlocal(serverAddr,
                           0, // Auto-assign local port
                           8080, // Remote port
                           TICKS_PER_SECOND * 10,
                           localAddr, // Specific interface IP
                           -1); // Not using interface number

    if (fd > 0) {
        printf("Connected!\n");
        printf("  Local: %s:%d\n",
               GetSocketLocalAddr(fd).print(),
               GetSocketLocalPort(fd));
        printf("  Remote: %s:%d\n",
               GetSocketRemoteAddr(fd).print(),
               GetSocketRemotePort(fd));

        close(fd);
    } else {
        printf("Connection failed: %d\n", fd);
    }
}
```

Interface Selection by Number

```
#include <tcp.h>
#include <init.h>

void ConnectViaInterfaceNumber(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("10.0.1.50");
    int interfaceNum = 1; // Use interface 1

    printf("Connecting via interface %d\n", interfaceNum);
    printf("Interface %d IP: %s\n",
           interfaceNum, GetInterfaceIP(interfaceNum).print());

    // Connect using interface number
    int fd = connectwlocal(serverAddr,
                           0, // Auto-assign local port
                           9000, // Remote port
                           TICKS_PER_SECOND * 10,
                           IPADDR::NullIP(), // Not using IP selection
                           interfaceNum); // Use interface 1

    if (fd > 0) {
        printf("Connected via interface %d\n", interfaceNum);
        printf("Actual local IP used: %s\n",
               GetSocketLocalAddr(fd).print());
        close(fd);
    } else {
        printf("Connection failed: %d\n", fd);
    }
}
```

Specifying Local Port (Not Recommended)

```
#include <tcp.h>
#include <init.h>

// Example showing why fixed local ports are problematic
void DemonstrateLocalPortProblem(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    uint16_t fixedLocalPort = 5000;

    printf("Attempting first connection with local port %d\n", fixedLocalPort);

    // First connection with fixed local port
    int fd1 = connectwlocal(serverAddr,
                           fixedLocalPort, // Fixed local port
                           8080,
                           TICKS_PER_SECOND * 10,
                           IPADDR::NullIP(),
                           -1);

    if (fd1 > 0) {
        printf("First connection succeeded\n");
        printf("  Local: %s:%d\n",
               GetSocketLocalAddr(fd1).print(),
               GetSocketLocalPort(fd1));

        // Simulate some work
        OSTimeDly(TICKS_PER_SECOND * 2);
        close(fd1);
        printf("First connection closed\n");
    }

    // Try to reconnect immediately with same local port
    printf("\nAttempting second connection with same local port %d\n",
           fixedLocalPort);

    int fd2 = connectwlocal(serverAddr,
                           fixedLocalPort, // Same fixed local port
                           8080,
                           TICKS_PER_SECOND * 10,
                           IPADDR::NullIP(),
                           -1);

    if (fd2 > 0) {
        printf("Second connection succeeded (lucky!)\n");
        close(fd2);
    } else {
        printf("Second connection FAILED: %d\n", fd2);
        printf("This is why fixed local ports are problematic!\n");
        printf("The local port may still be in TIME_WAIT state.\n");
    }
}
```

```

    }

    // Show the correct approach
    printf("\nCorrect approach - using automatic port assignment:\n");

    int fd3 = connectwlocal(serverAddr,
                           0, // Auto-assign port (RECOMMENDED)
                           8080,
                           TICKS_PER_SECOND * 10,
                           IPADDR::NullIP(),
                           -1);

    if (fd3 > 0) {
        printf("Connection with auto-assigned port succeeded\n");
        printf("  Assigned local port: %d\n", GetSocketLocalPort(fd3));
        close(fd3);
    }
}

```

Multi-Interface Load Balancing

```

#include <tcp.h>
#include <init.h>

// Distribute connections across multiple interfaces
void LoadBalancedConnections(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    int numInterfaces = 2; // Assume 2 network interfaces
    int connectionCount = 0;

    printf("Load balancing connections across %d interfaces\n", numInterfaces);

    for (int i = 0; i < 10; i++) { // Make 10 connections
        int interfaceNum = connectionCount % numInterfaces;

        printf("\nConnection %d using interface %d\n", i + 1, interfaceNum);

        int fd = connectwlocal(serverAddr,
                               0, // Auto port
                               8080,
                               TICKS_PER_SECOND * 10,
                               IPADDR::NullIP(),
                               interfaceNum);

        if (fd > 0) {
            printf("  Success - Local IP: %s, Port: %d\n",
                  GetSocketLocalAddr(fd).print(),
                  GetSocketLocalPort(fd));

            connectionCount++;

            // Do work...
            const char *msg = "Hello\r\n";
            write(fd, msg, strlen(msg));

            close(fd);
        } else {
            printf("  Failed: %d\n", fd);
        }

        OSTimeDly(TICKS_PER_SECOND);
    }

    printf("\nTotal successful connections: %d\n", connectionCount);
}

```

FTP Active Mode (Legitimate Fixed Port Use Case)

```

#include <tcp.h>
#include <init.h>

// FTP active mode requires server to connect back to specific client port
void FtpActiveMode(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR ftpServer = AsciiToIp("192.168.1.10");

    // Step 1: Connect to FTP control port
    int fdControl = connect(ftpServer, 21, TICKS_PER_SECOND * 10);
    if (fdControl < 0) {
        printf("FTP control connection failed: %d\n", fdControl);
        return;
    }
}

```

```

    }

    printf("FTP control connection established\n");

    // Step 2: Create listening socket for data connection
    uint16_t dataPort = 2000; // Our data port
    int fdDataListen = listen(INADDR_ANY, dataPort, 1);

    if (fdDataListen < 0) {
        printf("Failed to create data listen socket: %d\n", fdDataListen);
        close(fdControl);
        return;
    }

    // Step 3: Send PORT command to FTP server
    IPADDR localIP = GetSocketLocalAddr(fdControl);
    char portCmd[128];

    // Format: PORT h1,h2,h3,h4,p1,p2 where h=IP octets, p=port bytes
    uint8_t *ip = (uint8_t *)&localIP;
    snprintf(portCmd, sizeof(portCmd),
             "PORT %d,%d,%d,%d,%d,%d\r\n",
             ip[0], ip[1], ip[2], ip[3],
             dataPort >> 8, dataPort & 0xFF);

    write(fdControl, portCmd, strlen(portCmd));
    printf("Sent: %s", portCmd);

    // Step 4: Read FTP response
    char response[256];
    int n = read(fdControl, response, sizeof(response) - 1);
    if (n > 0) {
        response[n] = '\0';
        printf("FTP response: %s", response);
    }

    // Step 5: Accept data connection from server
    printf("Waiting for FTP server to connect to port %d...\n", dataPort);
    IPADDR serverDataAddr;
    uint16_t serverDataPort;
    int fdData = accept(fdDataListen, &serverDataAddr, &serverDataPort,
                       TICKS_PER_SECOND * 30);

    if (fdData > 0) {
        printf("FTP data connection accepted from %s:%d\n",
              serverDataAddr.print(), serverDataPort);

        // Transfer data...
        close(fdData);
    }

    close(fdDataListen);
    close(fdControl);

    printf("FTP active mode example complete\n");
}

```

Comparing Interface Selection Methods

```

#include <tcp.h>
#include <init.h>

void CompareInterfaceSelection(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");

    printf("Comparing three interface selection methods:\n\n");

    // Method 1: Automatic (standard connect)
    printf("Method 1: Automatic interface selection\n");
    int fd1 = connect(serverAddr, 8080, TICKS_PER_SECOND * 10);
    if (fd1 > 0) {
        printf(" Selected interface: %s\n",
              GetSocketLocalAddr(fd1).print());
        printf(" Interface number: %d\n",
              TcpGetSocketInterface(fd1));
        close(fd1);
    }

    OSTimeDly(TICKS_PER_SECOND);

    // Method 2: By IP address
    printf("\nMethod 2: Selection by IP address\n");
    IPADDR interface0IP = GetInterfaceIP(0);
    printf(" Requesting interface with IP: %s\n", interface0IP.print());
}

```

```

int fd2 = connectwlocal(serverAddr,
                        0,
                        8080,
                        TICKS_PER_SECOND * 10,
                        interface0IP, // Select by IP
                        -1);

if (fd2 > 0) {
    printf(" Selected interface: %s\n",
           GetSocketLocalAddr(fd2).print());
    printf(" Interface number: %d\n",
           TcpGetSocketInterface(fd2));
    close(fd2);
}

OSTimeDly(TICKS_PER_SECOND);

// Method 3: By interface number
printf("\nMethod 3: Selection by interface number\n");
int interfaceNum = 1;
printf(" Requesting interface number: %d\n", interfaceNum);

int fd3 = connectwlocal(serverAddr,
                        0,
                        8080,
                        TICKS_PER_SECOND * 10,
                        IPADDR::NullIP(),
                        interfaceNum); // Select by number

if (fd3 > 0) {
    printf(" Selected interface: %s\n",
           GetSocketLocalAddr(fd3).print());
    printf(" Interface number: %d\n",
           TcpGetSocketInterface(fd3));
    close(fd3);
}

printf("\nComparison complete\n");
}

```

Error Handling for Local Port Conflicts

```

#include <tcp.h>
#include <init.h>

void HandleLocalPortConflicts(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    uint16_t desiredPort = 6000;

    printf("Attempting to use local port %d\n", desiredPort);

    int fd = connectwlocal(serverAddr,
                           desiredPort,
                           8080,
                           TICKS_PER_SECOND * 10,
                           IPADDR::NullIP(),
                           -1);

    if (fd > 0) {
        printf("Successfully bound to port %d\n",
               GetSocketLocalPort(fd));
        close(fd);
    } else if (fd == TCP_ERR_NOCON) {
        printf("Port %d binding failed (TCP_ERR_NOCON)\n", desiredPort);
        printf("Possible causes:\n");
        printf(" - Port already in use by another socket\n");
        printf(" - Port in TIME_WAIT state from previous connection\n");
        printf(" - Insufficient permissions (ports < 1024)\n");
        printf("\nRetrying with automatic port assignment...\n");

        // Retry with automatic port
        fd = connectwlocal(serverAddr,
                           0, // Let system choose
                           8080,
                           TICKS_PER_SECOND * 10,
                           IPADDR::NullIP(),
                           -1);

        if (fd > 0) {
            printf("Success with auto-assigned port %d\n",
                   GetSocketLocalPort(fd));
            close(fd);
        }
    }
}

```

```

    }
} else {
    printf("Connection failed: %d\n", fd);
}
}

```

Testing All Parameter Combinations

```

#include <tcp.h>
#include <init.h>

void TestParameterCombinations(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    IPADDR serverAddr = AsciiToIp("192.168.1.100");
    IPADDR localIP = GetInterfaceIP(0);

    printf("Testing connectwlocal parameter combinations:\n\n");

    // Test 1: Auto port, no interface specification
    printf("Test 1: Auto port, auto interface\n");
    int fd1 = connectwlocal(serverAddr, 0, 8080,
                           TICKS_PER_SECOND * 10,
                           IPADDR::NullIP(), -1);

    if (fd1 > 0) {
        printf("    Result: %s:%d\n",
               GetSocketLocalAddr(fd1).print(),
               GetSocketLocalPort(fd1));
        close(fd1);
    } else {
        printf("    Failed: %d\n", fd1);
    }

    OSTimeDly(TICKS_PER_SECOND);

    // Test 2: Auto port, specific interface by IP
    printf("\nTest 2: Auto port, interface by IP (%s)\n",
           localIP.print());
    int fd2 = connectwlocal(serverAddr, 0, 8080,
                           TICKS_PER_SECOND * 10,
                           localIP, -1);

    if (fd2 > 0) {
        printf("    Result: %s:%d\n",
               GetSocketLocalAddr(fd2).print(),
               GetSocketLocalPort(fd2));
        close(fd2);
    } else {
        printf("    Failed: %d\n", fd2);
    }

    OSTimeDly(TICKS_PER_SECOND);

    // Test 3: Auto port, specific interface by number
    printf("\nTest 3: Auto port, interface by number (0)\n");
    int fd3 = connectwlocal(serverAddr, 0, 8080,
                           TICKS_PER_SECOND * 10,
                           IPADDR::NullIP(), 0);

    if (fd3 > 0) {
        printf("    Result: %s:%d\n",
               GetSocketLocalAddr(fd3).print(),
               GetSocketLocalPort(fd3));
        close(fd3);
    } else {
        printf("    Failed: %d\n", fd3);
    }

    OSTimeDly(TICKS_PER_SECOND);

    // Test 4: Fixed port, auto interface (NOT RECOMMENDED)
    printf("\nTest 4: Fixed port 7000, auto interface (NOT RECOMMENDED)\n");
    int fd4 = connectwlocal(serverAddr, 7000, 8080,
                           TICKS_PER_SECOND * 10,
                           IPADDR::NullIP(), -1);

    if (fd4 > 0) {
        printf("    Result: %s:%d\n",
               GetSocketLocalAddr(fd4).print(),
               GetSocketLocalPort(fd4));
        close(fd4);
    } else {
        printf("    Failed: %d (expected on retry)\n", fd4);
    }

    printf("\nTesting complete\n");
}

```

See also

[connect\(\)](#), [connectvia\(\)](#), [NoBlockConnectwlocal\(\)](#), [listen\(\)](#), [listenvia\(\)](#)

14.53.14.13 DumpTcpDebug()

```
void DumpTcpDebug ()
#include <tcp.h>
```

Display current TCP debug information to the console.

Outputs detailed TCP debug information to the console, including active connections, socket states, buffer usage, and other TCP stack diagnostics. The information displayed depends on which debug flags have been enabled via [EnableTcpDebug\(\)](#). This function is useful for troubleshooting network issues and monitoring TCP connection states.

See also

[EnableTcpDebug\(\)](#)

Expand for Example Usage

Examples

Basic Usage - Dumping TCP Debug Info

```
// Enable TCP debugging
EnableTcpDebug(0xFFFF); // Enable all debug flags

// Perform some TCP operations
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

// Dump current TCP state
DumpTcpDebug();

close(fd);
```

Periodic Monitoring - Debug Output Every Second

```
void MonitoringTask(void *pd) {
    EnableTcpDebug(0x0001); // Enable basic debug info

    while (1) {
        printf("\n=== TCP Debug Info ===\n");
        DumpTcpDebug();
        printf("=====\n\n");

        OSTimeDly(TICKS_PER_SECOND);
    }
}

void UserMain(void *pd) {
    init();
    OSTaskCreate(MonitoringTask, NULL, &MonitorStack[USER_TASK_STK_SIZE],
                MonitorStack, MAIN_PRIO - 1);
}
```

Troubleshooting Connection Issues

```
int fd = socket(AF_INET, SOCK_STREAM, 0);

printf("Before connect:\n");
DumpTcpDebug();

int result = connect(fd, (struct sockaddr*)&addr, sizeof(addr));

if (result < 0) {
    printf("Connect failed! Debug info:\n");
    DumpTcpDebug(); // See what went wrong
} else {
    printf("After successful connect:\n");
    DumpTcpDebug(); // Verify connection state
}
```

Debug Snapshot on Error

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

char buffer[1024];
int n = read(fd, buffer, sizeof(buffer));

if (n < 0) {
    printf("Read error occurred! TCP state:\n");
    DumpTcpDebug(); // Capture state at error time

    printf("Error code: %d\n", n);
    abortsocket(fd);
}
```

14.53.14.14 EnableTcpDebug()

```
void EnableTcpDebug (
    uint16_t dbFlags)
```

```
#include <tcp.h>
```

Enable TCP debug output with specified flags.

Configures which types of TCP debug information should be tracked and displayed when [DumpTcpDebug\(\)](#) is called. The debug flags control the level of detail and which aspects of TCP operations are monitored. Use bitwise OR to combine multiple flags. Set to 0 to disable all TCP debugging.

Parameters

<i>dbFlags</i>	Debug flags to enable (bitwise OR of debug flag values) • 0x0000 = Disable all debugging • 0xFFFF = Enable all debugging • Individual flags control specific debug features (see TCP debug documentation)

See also

[DumpTcpDebug\(\)](#)

Expand for Example Usage

Examples

Basic Usage - Enable All Debug Output

```
void UserMain(void *pd) {
    init();

    // Enable all TCP debugging
    EnableTcpDebug(0xFFFF);

    // TCP operations will now generate debug output
    int fd = socket(AF_INET, SOCK_STREAM, 0);
    connect(fd, (struct sockaddr*)&addr, sizeof(addr));

    DumpTcpDebug(); // Display debug information
}
```

Selective Debug Output - Specific Flags

```
// Enable only specific debug features
#define TCP_DBG_CONNECTIONS 0x0001
#define TCP_DBG_BUFFERS 0x0002
#define TCP_DBG_RETRANSMIT 0x0004

// Enable connection and buffer debugging only
EnableTcpDebug(TCP_DBG_CONNECTIONS | TCP_DBG_BUFFERS);

DumpTcpDebug(); // Will show only connection and buffer info
```


Toggle Debug Mode

```
bool debugEnabled = false;

void ToggleDebugMode() {
    if (debugEnabled) {
        printf("Disabling TCP debug\n");
        EnableTcpDebug(0x0000); // Disable all debugging
        debugEnabled = false;
    } else {
        printf("Enabling TCP debug\n");
        EnableTcpDebug(0xFFFF); // Enable all debugging
        debugEnabled = true;
    }
}

void UserMain(void *pd) {
    init();

    // Start with debugging disabled
    EnableTcpDebug(0x0000);

    // Can be toggled during runtime via command or button press
    while (1) {
        if (CheckDebugButton()) {
            ToggleDebugMode();
        }
        OSTimeDly(TICKS_PER_SECOND / 10);
    }
}
```

Production vs Development Build

```
void UserMain(void *pd) {
    init();

#ifdef DEBUG_BUILD
    // Enable full debugging in development builds
    EnableTcpDebug(0xFFFF);
    printf("TCP debugging enabled (development build)\n");
#else
    // Disable debugging in production builds
    EnableTcpDebug(0x0000);
#endif

    StartHttp();

    while (1) {
#ifdef DEBUG_BUILD
        DumpTcpDebug(); // Periodic debug output in dev builds
#endif
        OSTimeDly(TICKS_PER_SECOND * 10);
    }
}
```

Conditional Debug Based on Connection Count

```
void MonitorConnections() {
    static int lastConnectionCount = 0;
    int currentConnections = GetActiveConnectionCount();

    if (currentConnections > 10 && lastConnectionCount <= 10) {
        // High connection count, enable detailed debugging
        printf("High connection count detected, enabling TCP debug\n");
        EnableTcpDebug(0xFFFF);
    } else if (currentConnections <= 10 && lastConnectionCount > 10) {
        // Connection count normalized, reduce debug output
        printf("Connection count normalized, reducing TCP debug\n");
        EnableTcpDebug(0x0001); // Basic info only
    }

    lastConnectionCount = currentConnections;
}
```

Debug On Demand via Network Command

```
void HandleDebugCommand(int clientFd, const char *cmd) {
    if (strcmp(cmd, "DEBUG ON") == 0) {
        EnableTcpDebug(0xFFFF);
        write(clientFd, "TCP debug enabled\n", 18);
        DumpTcpDebug();
    } else if (strcmp(cmd, "DEBUG OFF") == 0) {
        EnableTcpDebug(0x0000);
        write(clientFd, "TCP debug disabled\n", 19);
    } else if (strcmp(cmd, "DEBUG STATUS") == 0) {

```

```

        DumpTcpDebug();
    }
}

```

14.53.14.15 getsocketerror()

```

int getsocketerror (
    int fd)
#include <tcp.h>

```

Get the error status of a socket.

Retrieves the current error state of a socket file descriptor. This function is useful for diagnosing connection failures, especially with non-blocking sockets where errors may not be immediately apparent from the return value of [connect\(\)](#) or other operations.

Parameters

<i>fd</i>	File descriptor of the socket to query

Returns

0 if no error, positive error code if an error occurred, negative value on invalid fd

Notes:

- This function is particularly useful after a non-blocking connect operation to determine why a connection failed to establish.
- Common error codes include:
 - 0: No error (success)
 - ECONNREFUSED: Connection refused by remote host
 - ETIMEDOUT: Connection attempt timed out
 - ENETUNREACH: Network is unreachable
 - EHOSTUNREACH: Host is unreachable
 - ECONNRESET: Connection reset by peer
- After calling [getsocketerror\(\)](#), the error status is typically cleared.

See also

[TcpGetSocketState\(\)](#), [NoBlockConnect\(\)](#), [NoBlockConnectVia\(\)](#), [connect\(\)](#)

Expand for Example Usage

Examples

Basic Error Checking After Non-Blocking Connect

```

#include <tcp.h>
#include <nbrtos.h>
#include <errno.h>

void CheckConnectionError(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    // Initiate non-blocking connection
    int fd = NoBlockConnect(serverIp, serverPort);

    if (fd < 0)
    {
        printf("Failed to create socket: %d\r\n", fd);
        return;
    }

    printf("Connecting to %I:%d...\r\n", serverIp, serverPort);
}

```

```

// Wait for connection to complete or fail
uint32_t startTime = Secs;
bool connected = false;

while ((Secs - startTime) < 10)
{
    TcpState state = TcpGetSocketState(fd);

    if (state == TCP_STATE_ESTABLISHED)
    {
        connected = true;
        printf("Connection established successfully\r\n");
        break;
    }
    else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
    {
        // Connection failed - check error code
        int error = getsocketerror(fd);

        if (error == 0)
        {
            printf("Connection closed without specific error\r\n");
        }
        else
        {
            printf("Connection failed with error: %d\r\n", error);

            // Interpret common errors
            switch (error)
            {
                case ECONNREFUSED:
                    printf(" Connection refused - server not listening\r\n");
                    break;
                case ETIMEDOUT:
                    printf(" Connection timed out - no response from host\r\n");
                    break;
                case ENETUNREACH:
                    printf(" Network unreachable - routing problem\r\n");
                    break;
                case EHOSTUNREACH:
                    printf(" Host unreachable - host may be down\r\n");
                    break;
                default:
                    printf(" Unknown error code\r\n");
                    break;
            }
        }
        break;
    }

    OSTimeDly(TICKS_PER_SECOND / 10);
}

if (!connected && (Secs - startTime) >= 10)
{
    printf("Connection timeout\r\n");
    int error = getsocketerror(fd);
    if (error != 0)
    {
        printf("Socket error at timeout: %d\r\n", error);
    }
}

close(fd);
}

```

Enhanced Connection with Detailed Error Reporting

```

#include <tcp.h>
#include <nbrtos.h>
#include <errno.h>

const char* GetErrorDescription(int error)
{
    switch (error)
    {
        case 0:
            return "No error";
        case ECONNREFUSED:
            return "Connection refused";
        case ETIMEDOUT:
            return "Connection timed out";
        case ENETUNREACH:
            return "Network unreachable";
        case EHOSTUNREACH:
            return "Host unreachable";
        case ECONNRESET:
            return "Connection reset";
        case ECONNABORTED:
            return "Connection aborted";
        case ENETDOWN:
            return "Network is down";
        case EHOSTDOWN:
            return "Host is down";
        case EADDRINUSE:
            return "Address already in use";
        case EADDRNOTAVAIL:
            return "Address not available";
    }
}

```

```

        default:
            return "Unknown error";
    }
}

int ConnectWithDetailedErrors(const IPADDR &serverIp, uint16_t serverPort,
                             uint32_t timeoutSecs)
{
    int fd = NoBlockConnect(serverIp, serverPort);

    if (fd < 0)
    {
        printf("Socket creation failed: %d\r\n", fd);
        return fd;
    }

    printf("Connecting to %I:%d (timeout: %lu seconds)...\r\n",
           serverIp, serverPort, timeoutSecs);

    uint32_t startTime = Secs;

    while ((Secs - startTime) < timeoutSecs)
    {
        TcpState state = TcpGetSocketState(fd);

        if (state == TCP_STATE_ESTABLISHED)
        {
            printf("Connection established after %lu seconds\r\n",
                   Secs - startTime);
            return fd;
        }
        else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
        {
            int error = getsocketerror(fd);
            printf("Connection failed: %s (code: %d)\r\n",
                   GetErrorDescription(error), error);
            close(fd);
            return -error; // Return negative error code
        }

        OSTimeDly(TICKS_PER_SECOND / 10);
    }

    // Timeout occurred
    int error = getsocketerror(fd);
    printf("Connection timeout after %lu seconds\r\n", timeoutSecs);
    if (error != 0)
    {
        printf("Final socket error: %s (code: %d)\r\n",
               GetErrorDescription(error), error);
    }

    close(fd);
    return -ETIMEDOUT;
}

void DetailedConnectionExample(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = ConnectWithDetailedErrors(serverIp, serverPort, 10);

    if (fd > 0)
    {
        // Connection successful - use socket
        const char *message = "Hello, Server!\r\n";
        write(fd, message, strlen(message));
        close(fd);
    }
    else
    {
        printf("Failed to connect (error: %d)\r\n", -fd);
    }
}

```

Monitoring Multiple Connections for Errors

```

#include <tcp.h>
#include <nbrtos.h>
#include <errno.h>

#define MAX_CONNECTIONS 5

struct ConnectionStatus
{
    int fd;
    IPADDR serverIp;

```

```

    uint16_t serverPort;
    bool active;
    int errorCode;
};

void MonitorMultipleConnections(void *pd)
{
    ConnectionStatus connections[MAX_CONNECTIONS];

    // Initialize connections
    const char *servers[] = {
        "192.168.1.100",
        "192.168.1.101",
        "192.168.1.102",
        "192.168.1.103",
        "192.168.1.104"
    };

    printf("Initiating %d connections...\r\n", MAX_CONNECTIONS);

    for (int i = 0; i < MAX_CONNECTIONS; i++)
    {
        connections[i].serverIp = AsciiToIp(servers[i]);
        connections[i].serverPort = 8080;
        connections[i].active = false;
        connections[i].errorCode = 0;

        connections[i].fd = NoBlockConnect(connections[i].serverIp,
                                           connections[i].serverPort);

        if (connections[i].fd < 0)
        {
            printf(" Connection %d: Failed to create socket (%d)\r\n",
                   i, connections[i].fd);
        }
        else
        {
            printf(" Connection %d: Initiated to %I:%d\r\n",
                   i, connections[i].serverIp, connections[i].serverPort);
        }
    }

    // Monitor connections
    uint32_t startTime = Secs;
    int activeCount = 0;

    while ((Secs - startTime) < 15)
    {
        bool allResolved = true;

        for (int i = 0; i < MAX_CONNECTIONS; i++)
        {
            if (connections[i].fd < 0)
                continue; // Already failed at creation

            if (connections[i].active)
                continue; // Already established

            allResolved = false;

            TcpState state = TcpGetSocketState(connections[i].fd);

            if (state == TCP_STATE_ESTABLISHED)
            {
                connections[i].active = true;
                activeCount++;
                printf(" Connection %d: Established\r\n", i);
            }
            else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
            {
                connections[i].errorCode = getsocketerror(connections[i].fd);
                printf(" Connection %d: Failed (error: %d)\r\n",
                       i, connections[i].errorCode);
                close(connections[i].fd);
                connections[i].fd = -1;
            }
        }

        if (allResolved)
        {
            printf("All connections resolved\r\n");
            break;
        }

        OSTimeDly(TICKS_PER_SECOND / 5);
    }
}

```

```

// Report final status
printf("\r\nFinal Connection Status:\r\n");
printf(" Active: %d/%d\r\n", activeCount, MAX_CONNECTIONS);

for (int i = 0; i < MAX_CONNECTIONS; i++)
{
    if (connections[i].fd < 0 && connections[i].errorCode == 0)
    {
        printf(" Connection %d: Socket creation failed\r\n", i);
    }
    else if (connections[i].active)
    {
        printf(" Connection %d: Active\r\n", i);
    }
    else
    {
        printf(" Connection %d: Failed (error: %d)\r\n",
            i, connections[i].errorCode);
    }
}

// Clean up
for (int i = 0; i < MAX_CONNECTIONS; i++)
{
    if (connections[i].fd > 0)
    {
        close(connections[i].fd);
    }
}
}

```

Using select() with Error Detection

```

#include <tcp.h>
#include <nbrtos.h>
#include <errno.h>

void SelectBasedConnectionWithErrors(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = NoBlockConnect(serverIp, serverPort);

    if (fd < 0)
    {
        printf("Socket creation failed: %d\r\n", fd);
        return;
    }

    printf("Connecting to %I:%d using select()...\r\n", serverIp, serverPort);

    // Use select to wait for connection completion
    fd_set write_fds;
    fd_set error_fds;

    FD_ZERO(&write_fds);
    FD_ZERO(&error_fds);
    FD_SET(fd, &write_fds);
    FD_SET(fd, &error_fds);

    int result = select(fd + 1, NULL, &write_fds, &error_fds,
        TICKS_PER_SECOND * 10);

    if (result < 0)
    {
        printf("select() error: %d\r\n", result);
        close(fd);
        return;
    }
    else if (result == 0)
    {
        printf("Connection timeout\r\n");
        int error = getsocketerror(fd);
        if (error != 0)
        {
            printf("Socket error at timeout: %d\r\n", error);
        }
        close(fd);
        return;
    }

    // Check for errors first
    if (FD_ISSET(fd, &error_fds))
    {
        int error = getsocketerror(fd);
        printf("Socket error detected by select(): %d\r\n", error);
    }
}

```

```

        close(fd);
        return;
    }

    // Check if socket is writable (connected)
    if (FD_ISSET(fd, &write_fds))
    {
        // Double-check with getsockopterror() - socket may be writable but failed
        int error = getsockopterror(fd);

        if (error != 0)
        {
            printf("Socket writable but has error: %d\r\n", error);
            close(fd);
            return;
        }

        // Verify state is established
        TcpState state = TcpGetSocketState(fd);

        if (state == TCP_STATE_ESTABLISHED)
        {
            printf("Connection established successfully\r\n");

            const char *message = "Hello via select()!\r\n";
            write(fd, message, strlen(message));
        }
        else
        {
            printf("Socket writable but not established (state: %d)\r\n", state);
        }
    }

    close(fd);
}

```

14.53.14.16 GetSocketLocalAddr()

```

IPADDR GetSocketLocalAddr (
    int fd) [inline]
#include <tcp.h>

```

Get the local interface IP address for a socket connection.

Retrieves the IP address of the local network interface used for the specified socket connection. This function works for both client connections and server sockets that have accepted a connection.

Parameters

<i>fd</i>	Socket file descriptor

Returns

IP address of the local interface, or [IPADDR::NullIP\(\)](#) if socket is not bound

Notes: For listening sockets, this returns the address the socket is bound to. If bound to `INADDR_ANY`, it returns `INADDR_ANY` rather than a specific interface IP.

- This function is useful for determining which network interface is being used when you have multiple interfaces (e.g., Ethernet and WiFi).
- This function supports both IPv4 and IPv6 addresses through the `IPADDR` class.

See also

[GetSocketRemoteAddr\(\)](#), [GetSocketRemotePort\(\)](#), [GetIPAddress\(\)](#)

Expand for Example Usage

Examples

Basic Usage - Determine Connection Interface

```
#include <tcp.h>
#include <netinterface.h>
#include <nbrtos.h>

void DisplayConnectionDetails(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    // Connect to server
    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Get local and remote addresses
    IPADDR localAddr = GetSocketLocalAddr(fd);
    IPADDR remoteAddr = GetSocketRemoteAddr(fd);
    uint16_t remotePort = GetSocketRemotePort(fd);

    printf("Connection established:\r\n");
    printf("  Local address: %I\r\n", localAddr);
    printf("  Remote address: %I:%d\r\n", remoteAddr, remotePort);

    // Determine which interface was used
    for (int i = 0; i < GetNumInterfaces(); i++)
    {
        if (localAddr == GetIPAddress(i))
        {
            printf("  Using interface %d\r\n", i);
            break;
        }
    }

    close(fd);
}
```

14.53.14.17 GetSocketLocalPort()

```
uint16_t GetSocketLocalPort (
    int fd)
```

```
#include <tcp.h>
```

Get the local port number for a socket connection.

Retrieves the local port number assigned to the specified socket connection.

Parameters

<i>fd</i>	Socket file descriptor

Returns

Local port number

See also

[GetSocketRemotePort\(\)](#)

14.53.14.18 GetSocketRemoteAddr()

```
IPADDR GetSocketRemoteAddr (
    int fd) [inline]
```

```
#include <tcp.h>
```

Get the remote host IP address for a socket connection.

Retrieves the IP address of the remote host connected to the specified socket. This function works for both established connections and listening sockets that have accepted a client connection.

Parameters

<i>fd</i>	Socket file descriptor

Returns

IP address of the remote host, or `IPADDR::NullIP()` if socket is not connected

Notes:

- For listening sockets that have not accepted a connection, this function returns `IPADDR::NullIP()`. -note This function supports both IPv4 and IPv6 addresses through the `IPADDR` class.

See also

[GetSocketLocalAddr\(\)](#), [GetSocketRemotePort\(\)](#), [accept\(\)](#)

Expand for Example Usage

Examples

Basic Usage - Get Remote Address After Connection

```
#include <tcp.h>
#include <nbrtos.h>

void DisplayConnectionInfo(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    // Connect to server
    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Get remote address (should match serverIp)
    IPADDR remoteAddr = GetSocketRemoteAddr(fd);

    printf("Connected to remote host: %I\r\n", remoteAddr);
    printf("Remote port: %d\r\n", serverPort);

    // Send data
    const char *message = "Hello, Server!\r\n";
    write(fd, message, strlen(message));

    close(fd);
}
```

14.53.14.19 GetSocketRemotePort()

```
uint16_t GetSocketRemotePort (
    int fd)
```

```
#include <tcp.h>
```

Get the remote host port number for a socket connection.

Retrieves the port number being used by the remote host in the specified socket connection. This function works for both client connections and server sockets that have accepted a connection.

Parameters

<i>fd</i>	Socket file descriptor

Returns

Port number of the remote host, or 0 if socket is not connected

Notes:

- For listening sockets that have not accepted a connection, this function returns 0.
- The remote port is typically an ephemeral port (1024-65535) for client connections, or a well-known/registered port for server connections.

See also

[GetSocketLocalPort\(\)](#), [GetSocketRemoteAddr\(\)](#), [GetSocketLocalAddr\(\)](#), [accept\(\)](#)

Expand for Example Usage**Examples****Basic Usage - Display Remote Connection Details**

```
#include <tcp.h>
#include <nbrtos.h>

void DisplayRemoteConnectionInfo(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    // Connect to server
    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Get remote connection details
    IPADDR remoteAddr = GetSocketRemoteAddr(fd);
    uint16_t remotePort = GetSocketRemotePort(fd);

    printf("Connected to remote host:\r\n");
    printf("  IP address: %I\r\n", remoteAddr);
    printf("  Port:      %d\r\n", remotePort);

    // Verify it matches what we connected to
    if (remoteAddr == serverIp && remotePort == serverPort)
    {
        printf("  Connection verified!\r\n");
    }

    close(fd);
}
```

14.53.14.20 getsockopt()

```
int getsockopt (
    int fd)
#include <tcp.h>
```

Query current TCP socket options.

Retrieves the currently enabled option flags for the specified socket. The return value is a bitmask where each set bit represents an enabled option.

Parameters

<i>fd</i>	Socket file descriptor

Returns

Bitmask of all currently enabled options ([TCP Socket Options](#)), or negative value on error

Notes:

- The returned bitmask can be tested for specific options using bitwise AND: `if (getsockopt(fd) & SO_NODELAY) { ... }`
- A return value of 0 indicates no special options are enabled (all default behavior).
- This function is useful for checking socket configuration before making changes or for debugging connection behavior.

See also

[setsockopt\(\)](#), [clrsockopt\(\)](#), [SetSocketUnackBuffers\(\)](#), [SetSocketRxBuffers\(\)](#), [SetSocketTxBuffers\(\)](#)

Expand for Example Usage**Examples****Check Specific Option Status**

```
#include <tcp.h>
#include <nbrtos.h>

void CheckNagleStatus(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Check if Nagle is disabled
    int options = getsockopt(fd);

    if (options < 0)
    {
        printf("Failed to get socket options: %d\r\n", options);
    }
    else
    {
        if (options & SO_NODELAY)
        {
            printf("Nagle algorithm is DISABLED (SO_NODELAY set)\r\n");
            printf("Small packets will be sent immediately\r\n");
        }
        else
        {
            printf("Nagle algorithm is ENABLED (default)\r\n");
            printf("Small packets will be buffered for efficiency\r\n");
        }
    }

    close(fd);
}
```

Display All Active Options

```
#include <tcp.h>
#include <nbrtos.h>

void DisplaySocketOptions(int fd)
{
    int options = getsockopt(fd);

    if (options < 0)
    {
        printf("Error reading socket options: %d\r\n", options);
        return;
    }

    printf("Socket Options (0x%X):\r\n", options);

    if (options == 0)
```

```

    {
        printf(" No special options set (all defaults)\r\n");
    }
    else
    {
        if (options & SO_NODELAY)
            printf(" SO_NODELAY - Nagle algorithm disabled\r\n");

        if (options & SO_PUSH)
            printf(" SO_PUSH - TCP PUSH flag enabled\r\n");

        if (options & SO_KEEPALIVE)
            printf(" SO_KEEPALIVE - Keepalive probes enabled\r\n");

        // Check for any other unknown flags
        int knownFlags = SO_NODELAY | SO_PUSH | SO_KEEPALIVE;
        int unknownFlags = options & ~knownFlags;

        if (unknownFlags)
        {
            printf(" Unknown flags: 0x%X\r\n", unknownFlags);
        }
    }
}

void OptionInspectionExample(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    printf("Initial socket configuration:\r\n");
    DisplaySocketOptions(fd);

    printf("\r\nEnabling SO_NODELAY and SO_PUSH...\r\n");
    setsockopt(fd, SO_NODELAY | SO_PUSH);

    printf("\r\nUpdated socket configuration:\r\n");
    DisplaySocketOptions(fd);

    close(fd);
}

```

Conditional Configuration

```

#include <tcp.h>
#include <nbrtos.h>

void ConditionallyEnableOption(int fd, int option, bool enable)
{
    int currentOptions = getsockopt(fd);

    if (currentOptions < 0)
    {
        printf("Failed to get current options\r\n");
        return;
    }

    bool isCurrentlySet = (currentOptions & option) != 0;

    if (enable && !isCurrentlySet)
    {
        printf("Enabling option 0x%X\r\n", option);
        setsockopt(fd, option);
    }
    else if (!enable && isCurrentlySet)
    {
        printf("Disabling option 0x%X\r\n", option);
        clrsocketoption(fd, option);
    }
    else
    {
        printf("Option 0x%X already in desired state\r\n", option);
    }
}

void SmartOptionManagement(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;
}

```

```

int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

if (fd < 0)
{
    printf("Connection failed: %d\r\n", fd);
    return;
}

// Only enable SO_NODELAY if not already set
ConditionallyEnableOption(fd, SO_NODELAY, true);

// Try to enable again (should be no-op)
ConditionallyEnableOption(fd, SO_NODELAY, true);

// Disable it
ConditionallyEnableOption(fd, SO_NODELAY, false);

close(fd);
}

```

Verify Configuration After Setting

```

#include <tcp.h>
#include <nbrtos.h>

bool VerifySocketOptions(int fd, int expectedOptions)
{
    int actualOptions = getsockopt(fd);

    if (actualOptions < 0)
    {
        printf("Error reading socket options: %d\r\n", actualOptions);
        return false;
    }

    if ((actualOptions & expectedOptions) == expectedOptions)
    {
        printf("Socket options verified: 0x%X\r\n", actualOptions);
        return true;
    }
    else
    {
        printf("Socket options mismatch!\r\n");
        printf("  Expected: 0x%X\r\n", expectedOptions);
        printf("  Actual:   0x%X\r\n", actualOptions);
        printf("  Missing:  0x%X\r\n", expectedOptions & ~actualOptions);
        return false;
    }
}

void ConfigureAndVerify(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Configure socket
    int desiredOptions = SO_NODELAY | SO_PUSH;
    int result = setsockopt(fd, desiredOptions);

    if (result >= 0)
    {
        printf("Options set, verifying...\r\n");

        if (VerifySocketOptions(fd, desiredOptions))
        {
            printf("Configuration successful!\r\n");
        }
        else
        {
            printf("Configuration verification failed!\r\n");
        }
    }
    else
    {
        printf("Failed to set options: %d\r\n", result);
    }

    close(fd);
}

```

```
}
```

Connection Diagnostics

```
#include <tcp.h>
#include <nbrtos.h>

void DiagnoseConnection(int fd)
{
    printf("\r\n=== Connection Diagnostics ===\r\n");

    // Socket options
    int options = getsockopt(fd);
    printf("Socket Options: 0x%X\r\n", options);

    if (options >= 0)
    {
        printf(" Nagle:      %s\r\n",
               (options & SO_NODELAY) ? "Disabled" : "Enabled");
        printf(" Push:      %s\r\n",
               (options & SO_PUSH) ? "Enabled" : "Disabled");
        printf(" Keepalive: %s\r\n",
               (options & SO_KEEPAIVE) ? "Enabled" : "Disabled");
    }

    // Connection state
    TcpState state = TcpGetSocketState(fd);
    printf("TCP State: %d\r\n", state);

    // Retransmissions
    int rtxCount = GetTcpRtxCount(fd);
    printf("Retransmissions: %d\r\n", rtxCount);

    // Idle time
    uint32_t idleSeconds = TcpGetLastRxInterval(fd) / TICKS_PER_SECOND;
    printf("Idle Time: %lu seconds\r\n", idleSeconds);

    printf("=====\r\n\r\n");
}

void ConnectionDiagnosticsTask(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Initial diagnostics
    DiagnoseConnection(fd);

    // Configure socket
    setsockopt(fd, SO_NODELAY);

    // Send some data
    const char *msg = "Test message\r\n";
    write(fd, msg, strlen(msg));

    OSTimeDly(TICKS_PER_SECOND * 2);

    // Check diagnostics after activity
    DiagnoseConnection(fd);

    close(fd);
}
```

14.53.14.21 GetTcpRtxCount()

```
int GetTcpRtxCount (
    int fd)
#include <tcp.h>
```

Get the number of TCP retransmissions on a socket.

Returns the count of how many times TCP has retransmitted data on this connection. Retransmissions occur when acknowledgments are not received from the remote host within the expected timeout period. A high retransmission count may indicate network congestion, packet loss, or a slow/unstable connection.

Parameters

<i>fd</i>	Socket file descriptor

Returns

Number of retransmissions that have occurred on this connection, or -1 if socket is invalid

Notes:

- This counter is maintained for the lifetime of the connection and is reset when the socket is closed or reused.
- A small number of retransmissions is normal in most networks. However, a rapidly increasing count or very high values may indicate network problems.
- Each retransmitted segment increments this counter, so the same data being retransmitted multiple times will increment the counter each time.

See also

[TcpGetSocketState\(\)](#), [getsocketerror\(\)](#)

Expand for Example Usage

Examples

Basic Retransmission Monitoring

```
#include <tcp.h>
#include <nbrtos.h>

void MonitorRetransmissions(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    printf("Connected - monitoring retransmissions\r\n");

    int lastRtxCount = 0;

    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND * 5);

        int currentRtxCount = GetTcpRtxCount(fd);

        if (currentRtxCount > lastRtxCount)
        {
            int newRtx = currentRtxCount - lastRtxCount;
            printf("Retransmissions detected: %d (total: %d)\r\n",
                newRtx, currentRtxCount);
            lastRtxCount = currentRtxCount;
        }

        // Send some data periodically
        const char *message = "Test data\r\n";
        write(fd, message, strlen(message));

        // Check connection state
        TcpState state = TcpGetSocketState(fd);
        if (state != TCP_STATE_ESTABLISHED)
        {
            printf("Connection closed (state: %d)\r\n", state);
            break;
        }
    }

    int finalRtxCount = GetTcpRtxCount(fd);
    printf("Connection closed - total retransmissions: %d\r\n", finalRtxCount);

    close(fd);
}
```

Connection Quality Assessment

```

#include <tcp.h>
#include <nbrtos.h>

typedef enum
{
    QUALITY_EXCELLENT, // 0 retransmissions
    QUALITY_GOOD,      // 1-5 retransmissions
    QUALITY_FAIR,       // 6-20 retransmissions
    QUALITY_POOR        // 20+ retransmissions
} ConnectionQuality;

ConnectionQuality AssessConnectionQuality(int fd)
{
    int rtxCount = GetTcpRtxCount (fd);

    if (rtxCOUNT == 0)
        return QUALITY_EXCELLENT;
    else if (rtxCOUNT <= 5)
        return QUALITY_GOOD;
    else if (rtxCOUNT <= 20)
        return QUALITY_FAIR;
    else
        return QUALITY_POOR;
}

const char* QualityToString(ConnectionQuality quality)
{
    switch (quality)
    {
        case QUALITY_EXCELLENT: return "Excellent";
        case QUALITY_GOOD:      return "Good";
        case QUALITY_FAIR:       return "Fair";
        case QUALITY_POOR:       return "Poor";
        default:                 return "Unknown";
    }
}

void QualityMonitorTask(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    printf("Connected - monitoring connection quality\r\n");

    ConnectionQuality lastQuality = QUALITY_EXCELLENT;

    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND * 10);

        ConnectionQuality currentQuality = AssessConnectionQuality(fd);
        int rtxCount = GetTcpRtxCount (fd);

        if (currentQuality != lastQuality)
        {
            printf("Connection quality changed: %s -> %s (RTX: %d)\r\n",
                QualityToString(lastQuality),
                QualityToString(currentQuality),
                rtxCount);

            lastQuality = currentQuality;

            // Take action based on quality
            if (currentQuality == QUALITY_POOR)
            {
                printf("WARNING: Poor connection quality detected!\r\n");
                // Could implement: reduce data rate, trigger reconnection, etc.
            }
        }

        // Check connection state
        TcpState state = TcpGetSocketState(fd);
        if (state != TCP_STATE_ESTABLISHED)
        {
            break;
        }
    }
}

```



```

    printf("Final connection quality: %s (RTX: %d)\r\n",
           QualityToString(AssessConnectionQuality(fd)),
           GetTcpRtxCount(fd));

    close(fd);
}

```

Server-Side Connection Diagnostics

```

#include <tcp.h>
#include <nbrtos.h>

void DiagnosticServerTask(void *pd)
{
    const uint16_t SERVER_PORT = 8080;

    int listenFd = listen(INADDR_ANY, SERVER_PORT, 5);

    if (listenFd < 0)
    {
        printf("Failed to create listening socket: %d\r\n", listenFd);
        return;
    }

    printf("Diagnostic server listening on port %d\r\n", SERVER_PORT);

    while (1)
    {
        IPADDR clientAddr;
        uint16_t clientPort;

        int clientFd = accept(listenFd, &clientAddr, &clientPort, 0);

        if (clientFd > 0)
        {
            printf("\r\nClient connected: %I:%d\r\n", clientAddr, clientPort);

            int initialRtx = GetTcpRtxCount(clientFd);
            uint32_t startTime = Secs;
            uint32_t bytesTransferred = 0;

            // Handle client for up to 60 seconds
            while ((Secs - startTime) < 60)
            {
                char buffer[256];
                int bytesRead = read(clientFd, buffer, sizeof(buffer) - 1,
                                     TICKS_PER_SECOND * 5);

                if (bytesRead > 0)
                {
                    bytesTransferred += bytesRead;

                    // Echo back
                    int bytesWritten = write(clientFd, buffer, bytesRead);
                    if (bytesWritten > 0)
                    {
                        bytesTransferred += bytesWritten;
                    }
                }

                // Check connection state
                TcpState state = TcpGetSocketState(clientFd);
                if (state != TCP_STATE_ESTABLISHED)
                {
                    break;
                }
            }

            // Display connection statistics
            int finalRtx = GetTcpRtxCount(clientFd);
            uint32_t duration = Secs - startTime;

            printf("\r\nConnection Statistics for %I:\r\n", clientAddr);
            printf("  Duration: %lu seconds\r\n", duration);
            printf("  Bytes transferred: %lu\r\n", bytesTransferred);
            printf("  Retransmissions: %d\r\n", finalRtx);

            if (finalRtx > 0)
            {
                float rtxRate = (float)finalRtx / duration;
                printf("  RTX rate: %.2f per second\r\n", rtxRate);
            }

            close(clientFd);
        }
    }
}

```

```

        close(listenFd);
    }

```

Adaptive Behavior Based on Retransmissions

```

#include <tcp.h>
#include <nbrtos.h>

#define RTX_THRESHOLD_REDUCE_RATE 10
#define RTX_THRESHOLD_RECONNECT 50

void AdaptiveDataTransfer(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    printf("Connected - starting adaptive data transfer\r\n");

    uint32_t sendInterval = 1; // Start with 1 second interval
    int lastRtxCheck = 0;

    while (1)
    {
        // Send data at current rate
        char message[64];
        snprintf(message, sizeof(message), "Data packet (interval: %lu sec)\r\n",
            sendInterval);
        write(fd, message, strlen(message));

        OSTimeDly(sendInterval * TICKS_PER_SECOND);

        // Check retransmissions
        int currentRtx = GetTcpRtxCount(fd);
        int newRtx = currentRtx - lastRtxCheck;

        printf("RTX count: %d (new: %d)\r\n", currentRtx, newRtx);

        if (currentRtx >= RTX_THRESHOLD_RECONNECT)
        {
            printf("RTX threshold exceeded - reconnecting\r\n");
            close(fd);

            OSTimeDly(TICKS_PER_SECOND * 5);

            fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);
            if (fd < 0)
            {
                printf("Reconnection failed\r\n");
                break;
            }

            sendInterval = 1;
            lastRtxCheck = 0;
            printf("Reconnected - reset to 1 second interval\r\n");
        }
        else if (currentRtx >= RTX_THRESHOLD_REDUCE_RATE && sendInterval < 10)
        {
            sendInterval *= 2;
            if (sendInterval > 10) sendInterval = 10;
            printf("Reducing data rate - new interval: %lu seconds\r\n",
                sendInterval);
        }
        else if (currentRtx == lastRtxCheck && sendInterval > 1)
        {
            // No new retransmissions - can increase rate
            sendInterval = (sendInterval > 1) ? sendInterval / 2 : 1;
            printf("Increasing data rate - new interval: %lu seconds\r\n",
                sendInterval);
        }

        lastRtxCheck = currentRtx;

        // Check connection state
        TcpState state = TcpGetSocketState(fd);
        if (state != TCP_STATE_ESTABLISHED)
        {
            printf("Connection closed (state: %d)\r\n", state);
            break;
        }
    }
}

```

```
    }  
}  
  
if (fd > 0) close(fd);  
}
```

14.53.14.22 listen()

```
int listen (  
    const IPADDR & addr,  
    uint16_t port,  
    uint8_t maxpend = 5) [inline]  
#include <tcp.h>
```

Create a listening socket to accept incoming TCP connections.

Creates a socket that listens for incoming connection requests on the specified port. This is the server-side counterpart to [connect\(\)](#), enabling your application to accept connections from remote clients. Incoming connections are placed in a pending queue until accepted with [accept\(\)](#). The maxpend parameter determines how many connections can wait in the queue before new connection attempts are rejected by the TCP stack.

The listening socket itself does not handle data transfer - it only accepts new connections. Each accepted connection returns a new socket file descriptor for communicating with that specific client. The listening socket remains open and continues accepting new connections.

Parameters

<i>addr</i>	IP address to bind to: • Use INADDR_ANY (or IPADDR::AnyIP()) to accept connections on all network interfaces. This is the most common usage for servers • Use a specific IP address to accept connections only on that interface • Example: INADDR_ANY for multi-homed systems
<i>port</i>	Port number to listen on (1-65535). Common server ports include: • 80 for HTTP • 443 for HTTPS • 23 for Telnet • Custom ports typically use values above 1024 Ports below 1024 are traditionally reserved for well-known services
<i>maxpend</i>	Maximum number of pending connections in the accept queue (default: 5). This is the maximum number of clients that can be waiting in the queue simultaneously. When this limit is reached, additional connection attempts will be refused by the TCP stack. Typical values are 5-20. Higher values consume more system resources but allow for burst connection handling

Returns

Positive file descriptor on success (use with [accept\(\)](#) to get client connections), negative [TCP Socket Status](#) error code on failure:

- TCP_ERR_NONE_AVAIL: No available sockets in system pool
- TCP_ERR_NOCON: Port already in use or binding failed

Notes:

- The listening socket uses one socket from the system pool. Each accepted connection uses an additional socket. Plan your maxpend value and system socket count accordingly.

- Always call `close()` on the listening socket when your server is shutting down to free the socket resource and release the port.
- The listening socket does not automatically close when the application exits. For clean shutdown, explicitly close all listening sockets.
- If you need to listen on a specific interface in a multi-homed system, use `listenvia()` instead for explicit interface control.
- Port numbers below 1024 may require special privileges on some systems, though this is typically not an issue on embedded NetBurner devices.

Warning

- If the port is already in use (by another listening socket or active connection), `listen()` will fail with `TCP_ERR_NOCON`. Ensure the port is available before calling.
- The `maxpend` parameter limits simultaneous pending connections, not total accepted connections. Once a connection is accepted with `accept()`, it no longer occupies a slot in the pending queue.

Expand for Example Usage

Examples

Basic Echo Server

```
#include <tcp.h>
#include <init.h>
#include <nbrtos.h>

void EchoServer(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Create listening socket on port 7 (echo port)
    int fdListen = listen(INADDR_ANY, 7, 5);

    if (fdListen < 0) {
        printf("Failed to create listening socket: %d\n", fdListen);
        return;
    }

    printf("Echo server listening on port 7\n");
    printf("Waiting for connections...\n\n");

    while (1) {
        // Accept incoming connection
        IPADDR clientAddr;
        uint16_t clientPort;
        int fdClient = accept(fdListen, &clientAddr, &clientPort, 0);

        if (fdClient > 0) {
            printf("Client connected: %s:%d\n",
                clientAddr.print(), clientPort);

            // Echo data back to client
            char buffer[512];
            while (1) {
                int n = read(fdClient, buffer, sizeof(buffer));
                if (n <= 0) break; // Connection closed

                writeall(fdClient, buffer, n); // Echo back
            }

            close(fdClient);
            printf("Client disconnected\n\n");
        }
    }

    close(fdListen);
}
```

HTTP Server Example

```
#include <tcp.h>
#include <init.h>
#include <nbrtos.h>

void SimpleHttpServer(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);
```

```

// Create listening socket on port 80 (HTTP)
int fdListen = listen(INADDR_ANY, 80, 10);

if (fdListen < 0) {
    printf("Failed to create HTTP listener: %d\n", fdListen);
    return;
}

printf("HTTP server listening on port 80\n");

while (1) {
    IPADDR clientAddr;
    uint16_t clientPort;
    int fdClient = accept(fdListen, &clientAddr, &clientPort, 0);

    if (fdClient > 0) {
        printf("HTTP request from %s:%d\n",
            clientAddr.print(), clientPort);

        // Read HTTP request (simplified)
        char request[1024];
        int n = read(fdClient, request, sizeof(request) - 1);

        if (n > 0) {
            request[n] = '\0';
            printf("Request: %s\n", request);

            // Send HTTP response
            const char *response =
                "HTTP/1.1 200 OK\r\n"
                "Content-Type: text/html\r\n"
                "Connection: close\r\n"
                "\r\n"
                "<html><body><h1>Hello from NetBurner!</h1></body></html>";

            writeall(fdClient, response, strlen(response));
        }

        close(fdClient);
    }
}

```

Server with Error Handling

```

#include <tcp.h>
#include <init.h>

int CreateServer(uint16_t port, uint8_t backlog) {
    int fdListen = listen(INADDR_ANY, port, backlog);

    if (fdListen < 0) {
        // Handle specific error cases
        switch (fdListen) {
            case TCP_ERR_NONE_AVAIL:
                printf("ERROR: No sockets available\n");
                printf("Too many open connections or sockets not freed\n");
                break;

            case TCP_ERR_NOCON:
                printf("ERROR: Cannot bind to port %d\n", port);
                printf("Port may already be in use\n");
                break;

            default:
                printf("ERROR: Failed to create listener: %d\n", fdListen);
                break;
        }

        return -1;
    }

    printf("Server listening on port %d\n", port);
    printf(" Backlog: %d connections\n", backlog);
    printf(" Accepting on all interfaces\n");

    return fdListen;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int fdListen = CreateServer(8080, 10);

    if (fdListen > 0) {
        // Accept connections...
    }
}

```

```

        close(fdListen);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Multi-Port Server

```

#include <tcp.h>
#include <init.h>
#include <nbrtos.h>

void MultiPortServer(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Listen on multiple ports
    int fdTelnet = listen(INADDR_ANY, 23, 3); // Telnet
    int fdHttp = listen(INADDR_ANY, 80, 10); // HTTP
    int fdCustom = listen(INADDR_ANY, 9000, 5); // Custom

    if (fdTelnet < 0 || fdHttp < 0 || fdCustom < 0) {
        printf("Failed to create one or more listening sockets\n");
        return;
    }

    printf("Multi-port server running:\n");
    printf("  Telnet: port 23\n");
    printf("  HTTP:   port 80\n");
    printf("  Custom: port 9000\n");

    while (1) {
        // Check for connections on any port
        // In a real implementation, you'd use select() or tasks

        // Check telnet port
        IPADDR addr;
        uint16_t port;
        int fd = accept(fdTelnet, &addr, &port, TICKS_PER_SECOND / 10);
        if (fd > 0) {
            printf("Telnet connection from %s:%d\n", addr.print(), port);
            // Handle telnet connection...
            close(fd);
        }

        // Check HTTP port
        fd = accept(fdHttp, &addr, &port, TICKS_PER_SECOND / 10);
        if (fd > 0) {
            printf("HTTP connection from %s:%d\n", addr.print(), port);
            // Handle HTTP connection...
            close(fd);
        }

        // Check custom port
        fd = accept(fdCustom, &addr, &port, TICKS_PER_SECOND / 10);
        if (fd > 0) {
            printf("Custom connection from %s:%d\n", addr.print(), port);
            // Handle custom connection...
            close(fd);
        }
    }
}

```

Connection Limit Testing

```

#include <tcp.h>
#include <init.h>

void TestConnectionLimits(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Test with small backlog
    uint8_t maxPending = 3;
    int fdListen = listen(INADDR_ANY, 9000, maxPending);

    if (fdListen < 0) {
        printf("Failed to create listener: %d\n", fdListen);
        return;
    }

    printf("Testing connection limits\n");
    printf("Max pending connections: %d\n", maxPending);
    printf("Waiting for connections...\n");
}

```

```

int acceptedCount = 0;

while (acceptedCount < 10) {
    IPADDR addr;
    uint16_t port;

    // Try to accept with short timeout
    int fd = accept(fdListen, &addr, &port, TICKS_PER_SECOND * 2);

    if (fd > 0) {
        acceptedCount++;
        printf("Accepted connection %d from %s:%d\n",
            acceptedCount, addr.print(), port);

        // Don't close immediately - let them queue up
        OSTimeDly(TICKS_PER_SECOND * 5);

        close(fd);
        printf("  Closed connection %d\n", acceptedCount);
    } else if (fd == TCP_ERR_TIMEOUT) {
        printf("No pending connections (timeout)\n");
    } else {
        printf("Accept error: %d\n", fd);
    }
}

close(fdListen);
printf("\nTest complete: %d connections accepted\n", acceptedCount);
}

```

Graceful Server Shutdown

```

#include <tcp.h>
#include <init.h>
#include <nbrtos.h>

volatile bool g_serverRunning = true;

void GracefulServer(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int fdListen = listen(INADDR_ANY, 8080, 5);

    if (fdListen < 0) {
        printf("Failed to create listener: %d\n", fdListen);
        return;
    }

    printf("Server started on port 8080\n");
    printf("Set g_serverRunning = false to shutdown\n\n");

    while (g_serverRunning) {
        // Accept with short timeout to check shutdown flag frequently
        IPADDR addr;
        uint16_t port;
        int fd = accept(fdListen, &addr, &port, TICKS_PER_SECOND);

        if (fd > 0) {
            printf("Client connected: %s:%d\n", addr.print(), port);

            // Check shutdown flag before processing
            if (!g_serverRunning) {
                const char *msg = "Server shutting down\r\n";
                write(fd, msg, strlen(msg));
                close(fd);
                break;
            }

            // Handle client...
            close(fd);
        }
    }

    close(fdListen);
    printf("Server shutdown complete\n");
}

```

Connection Statistics Monitoring

```

#include <tcp.h>
#include <init.h>
#include <nbrtos.h>

struct ServerStats {

```

```

    int totalConnections;
    int activeConnections;
    int rejectedConnections;
    uint32_t bytesReceived;
    uint32_t bytesSent;
} g_stats = {0};

void MonitoredServer(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int fdListen = listen(INADDR_ANY, 8080, 5);

    if (fdListen < 0) {
        printf("Failed to create listener: %d\n", fdListen);
        return;
    }

    printf("Monitored server on port 8080\n\n");

    uint32_t lastStatsTime = Secs;

    while (1) {
        IPADDR addr;
        uint16_t port;
        int fd = accept(fdListen, &addr, &port, TICKS_PER_SECOND);

        if (fd > 0) {
            g_stats.totalConnections++;
            g_stats.activeConnections++;

            printf("Connection %d from %s:%d\n",
                g_stats.totalConnections, addr.print(), port);

            // Handle connection
            char buffer[256];
            int n = read(fd, buffer, sizeof(buffer));
            if (n > 0) {
                g_stats.bytesReceived += n;

                // Echo response
                writeall(fd, buffer, n);
                g_stats.bytesSent += n;
            }

            close(fd);
            g_stats.activeConnections--;
        } else if (fd < 0 && fd != TCP_ERR_TIMEOUT) {
            g_stats.rejectedConnections++;
        }

        // Print statistics every 10 seconds
        if ((Secs - lastStatsTime) >= 10) {
            printf("\n=== Statistics ===\n");
            printf("Total connections: %d\n", g_stats.totalConnections);
            printf("Active connections: %d\n", g_stats.activeConnections);
            printf("Rejected connections: %d\n", g_stats.rejectedConnections);
            printf("Bytes received: %lu\n", g_stats.bytesReceived);
            printf("Bytes sent: %lu\n", g_stats.bytesSent);
            lastStatsTime = Secs;
        }
    }
}

```

Testing Port Availability

```

#include <tcp.h>
#include <init.h>

bool IsPortAvailable(uint16_t port) {
    int fd = listen(INADDR_ANY, port, 1);

    if (fd > 0) {
        close(fd);
        return true;
    }

    return false;
}

uint16_t FindAvailablePort(uint16_t startPort, uint16_t endPort) {
    printf("Searching for available port in range %d-%d\n",
        startPort, endPort);

    for (uint16_t port = startPort; port <= endPort; port++) {
        if (IsPortAvailable(port)) {

```



```

        printf("Found available port: %d\n", port);
        return port;
    }
}

printf("No available ports in range\n");
return 0;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Test specific port
    if (IsPortAvailable(8080)) {
        printf("Port 8080 is available\n");
    } else {
        printf("Port 8080 is in use\n");
    }

    // Find any available port in range
    uint16_t port = FindAvailablePort(9000, 9100);

    if (port > 0) {
        int fdListen = listen(INADDR_ANY, port, 5);
        if (fdListen > 0) {
            printf("Server listening on port %d\n", port);
            close(fdListen);
        }
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Dynamic Backlog Adjustment

```

#include <tcp.h>
#include <init.h>
#include <nbrtos.h>

void AdaptiveBacklogServer(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Start with moderate backlog
    uint8_t currentBacklog = 5;
    int fdListen = listen(INADDR_ANY, 8080, currentBacklog);

    if (fdListen < 0) {
        printf("Failed to create listener: %d\n", fdListen);
        return;
    }

    printf("Server started with backlog=%d\n", currentBacklog);

    int recentConnections = 0;
    uint32_t lastAdjustTime = Secs;

    while (1) {
        IPADDR addr;
        uint16_t port;
        int fd = accept(fdListen, &addr, &port, TICKS_PER_SECOND);

        if (fd > 0) {
            recentConnections++;

            // Handle connection...
            close(fd);
        }

        // Adjust backlog every 30 seconds based on load
        if ((Secs - lastAdjustTime) >= 30) {
            uint8_t newBacklog = currentBacklog;

            if (recentConnections > 20 && currentBacklog < 20) {
                // High load - increase backlog
                newBacklog = currentBacklog + 5;
                printf("High load (%d conn/30sec) - increasing backlog\n",
                    recentConnections);
            } else if (recentConnections < 5 && currentBacklog > 5) {
                // Low load - decrease backlog
                newBacklog = currentBacklog - 2;
                printf("Low load (%d conn/30sec) - decreasing backlog\n",
                    recentConnections);
            }

            currentBacklog = newBacklog;
            lastAdjustTime = Secs;
        }
    }
}

```

```

        if (newBacklog != currentBacklog) {
            // Recreate listener with new backlog
            close(fdListen);

            fdListen = listen(INADDR_ANY, 8080, newBacklog);

            if (fdListen > 0) {
                currentBacklog = newBacklog;
                printf("Backlog adjusted to %d\n", currentBacklog);
            }
        }

        recentConnections = 0;
        lastAdjustTime = Secs;
    }
}
}

```

See also

[accept\(\)](#), [listenvia\(\)](#), [close\(\)](#), [connect\(\)](#), [INADDR_ANY](#)

14.53.14.23 listenvia() [1/2]

```

int listenvia (
    const IPADDR & addr,
    uint16_t port,
    const IPADDR & localIpAddress,
    uint8_t maxpend = 5) [inline]
#include <tcp.h>

```

Create a listening socket bound to a specific interface IP address.

Creates a socket that listens for incoming connection requests on the specified port using the network interface with the given IP address. If multiple interfaces share the same IP address, the lower numbered interface is selected. Incoming connections are placed in a pending queue until accepted with [accept\(\)](#).

Parameters

<i>addr</i>	IP address to bind to (use INADDR_ANY to accept connections on any interface)
<i>port</i>	Port number to listen on
<i>localIpAddress</i>	IP address of the local interface to use. Selects the interface with this IP address (lower numbered interface if multiple match)
<i>maxpend</i>	Maximum number of pending connections in the accept queue (default: 5)

Returns

Positive file descriptor on success, negative [TCP Socket Status](#) error code on failure

Note

This function is useful when you have multiple network interfaces and need to ensure the listening socket binds to a specific interface rather than all interfaces.

Warning

If multiple interfaces have the same IP address, the behavior is deterministic but may not match expectations - the lower numbered interface will be selected.

See also

[accept\(\)](#), [listen\(\)](#), [close\(\)](#), [CoreListen\(\)](#)

Expand for Example Usage

Examples

Basic Usage - Listen on Specific Interface

```

#include <tcp.h>
#include <nbrtos.h>

void ServerTask(void *pd)
{
    // Get the IP address of interface 0
    IPADDR localIp = GetIPAddress(0);

    // Create a listening socket on port 8080 bound to interface 0
    int listenFd = listenvia(INADDR_ANY, 8080, localIp);

    if (listenFd < 0)
    {
        printf("Failed to create listening socket: %d\r\n", listenFd);
        return;
    }

    printf("Listening on %I:8080 via interface IP %I\r\n", INADDR_ANY, localIp);

    while (1)
    {
        IPADDR clientAddr;
        uint16_t clientPort;

        // Accept incoming connections
        int clientFd = accept(listenFd, &clientAddr, &clientPort, 0);

        if (clientFd > 0)
        {
            printf("Client connected from %I:%d\r\n", clientAddr, clientPort);

            // Handle client connection
            // ... your code here ...

            close(clientFd);
        }

        close(listenFd);
    }
}

```

Multi-Interface Server - Listen on Both Ethernet and WiFi

```

#include <tcp.h>
#include <nbrtos.h>

void MultiInterfaceServer(void *pd)
{
    const uint16_t SERVER_PORT = 8080;

    // Get IP addresses for Ethernet (interface 0) and WiFi (interface 1)
    IPADDR ethernetIp = GetIPAddress(0);
    IPADDR wifiIp = GetIPAddress(1);

    // Create listening socket on Ethernet interface
    int ethernetFd = listenvia(INADDR_ANY, SERVER_PORT, ethernetIp, 10);
    if (ethernetFd < 0)
    {
        printf("Failed to bind to Ethernet interface: %d\r\n", ethernetFd);
        return;
    }

    // Create listening socket on WiFi interface
    int wifiFd = listenvia(INADDR_ANY, SERVER_PORT, wifiIp, 10);
    if (wifiFd < 0)
    {
        printf("Failed to bind to WiFi interface: %d\r\n", wifiFd);
        close(ethernetFd);
        return;
    }

    printf("Server listening on:\r\n");
    printf(" Ethernet: %I:%d\r\n", ethernetIp, SERVER_PORT);
    printf(" WiFi:      %I:%d\r\n", wifiIp, SERVER_PORT);

    // Use select() to monitor both sockets
    fd_set read_fds;

    while (1)
    {
        FD_ZERO(&read_fds);
        FD_SET(ethernetFd, &read_fds);
        FD_SET(wifiFd, &read_fds);

        int maxFd = (ethernetFd > wifiFd) ? ethernetFd : wifiFd;
    }
}

```

```

    if (select(maxFd + 1, &read_fds, NULL, NULL, TICKS_PER_SECOND * 5) > 0)
    {
        IPADDR clientAddr;
        uint16_t clientPort;

        if (FD_ISSET(ethernetFd, &read_fds))
        {
            int clientFd = accept(ethernetFd, &clientAddr, &clientPort, 0);
            if (clientFd > 0)
            {
                printf("Client connected via Ethernet from %I:%d\r\n",
                    clientAddr, clientPort);
                // Handle client...
                close(clientFd);
            }
        }

        if (FD_ISSET(wifiFd, &read_fds))
        {
            int clientFd = accept(wifiFd, &clientAddr, &clientPort, 0);
            if (clientFd > 0)
            {
                printf("Client connected via WiFi from %I:%d\r\n",
                    clientAddr, clientPort);
                // Handle client...
                close(clientFd);
            }
        }
    }

    close(ethernetFd);
    close(wifiFd);
}

```

Error Handling and Interface Validation

```

#include <tcp.h>
#include <netinterface.h>

int CreateInterfaceListener(int interfaceNum, uint16_t port)
{
    // Validate interface exists and is active
    if (interfaceNum < 0 || interfaceNum >= GetNumInterfaces())
    {
        printf("Invalid interface number: %d\r\n", interfaceNum);
        return -1;
    }

    // Get interface IP address
    IPADDR localIp = GetIPAddress(interfaceNum);

    if (localIp == IPADDR_ANY)
    {
        printf("Interface %d has no IP address assigned\r\n", interfaceNum);
        return -1;
    }

    // Create listening socket with custom queue size
    int listenFd = listenvia(INADDR_ANY, port, localIp, 20);

    if (listenFd < 0)
    {
        // Handle specific error codes
        switch (listenFd)
        {
            case TCP_ERR_NOSOCKETS:
                printf("No socket resources available\r\n");
                break;
            case TCP_ERR_PORT_IN_USE:
                printf("Port %d already in use on interface\r\n", port);
                break;
            default:
                printf("Failed to create listener: error %d\r\n", listenFd);
                break;
        }
        return listenFd;
    }

    printf("Successfully listening on %I:%d (interface %d)\r\n",
        localIp, port, interfaceNum);

    return listenFd;
}

```

14.53.14.24 listenvia() [2/2]

```
int listenvia (
    const IPADDR & addr,
    uint16_t port,
    int ifn,
    uint8_t maxpend = 5) [inline]
#include <tcp.h>
```

Create a listening socket on a specific network interface.

Creates a socket that listens for incoming connection requests on the specified port and network interface. This provides direct control over which interface accepts connections when multiple interfaces are available, useful in multi-homed systems where you need to segregate traffic by interface or ensure connections arrive on a specific network path.

This function is the interface-specific version of [listen\(\)](#). While [listen\(\)](#) accepts connections on all interfaces (when using INADDR_ANY) or relies on IP routing, this function gives explicit control by interface number. Incoming connections are placed in a pending queue until accepted with [accept\(\)](#).

Parameters

<i>addr</i>	IP address to bind to: • Use INADDR_ANY (or IPADDR::AnyIP()) to accept connections on any IP address assigned to the specified interface • Use a specific IP address to accept only on that address (the IP must be assigned to the specified interface) • INADDR_ANY is the most common usage for interface-specific servers
<i>port</i>	Port number to listen on (1-65535). The port must not already be in use on any interface
<i>ifn</i>	Network interface number (0, 1, 2, etc.) to bind to. This forces the listening socket to accept connections only through this specific interface, regardless of routing table or IP address configuration. Valid range is 0 to (number of interfaces - 1). Use GetNumberOfInterfaces() to determine valid range
<i>maxpend</i>	Maximum number of pending connections in the accept queue (default: 5). This is the maximum number of clients that can be waiting simultaneously. When this limit is reached, additional connection attempts will be refused

Returns

Positive file descriptor on success (use with [accept\(\)](#) to get client connections), negative [TCP Socket Status](#) error code on failure:

- TCP_ERR_NONE_AVAIL: No available sockets in system pool
- TCP_ERR_NOCON: Port already in use or binding failed

Notes

- The listening socket uses one socket from the system pool. Each accepted connection uses an additional socket.
- Interface numbers typically start at 0 and increment sequentially. Interface 0 is usually the primary Ethernet port, but this can vary by platform.
- If you specify an invalid interface number (negative or $\geq$ number of interfaces), the behavior is undefined. Always validate with GetNumberOfInterfaces() first.
- The specified interface must be in an "up" state with a valid IP address assigned for connections to be accepted successfully.
- Always call [close\(\)](#) on the listening socket when shutting down to free the socket resource and release the port.

- This function provides the most direct interface control for server applications. For interface selection by IP address, see the other [listenvia\(\)](#) overload.

Warning

Specifying an invalid interface number may cause the listen to fail or bind to an unexpected interface. Always validate interface numbers before use.

The port is global across all interfaces - you cannot listen on the same port on multiple interfaces simultaneously. Each port can only have one listener.

Expand for Example Usage

Examples

Basic Interface-Specific Server

```
#include <tcp.h>
#include <init.h>
#include <networkinterface.h>
#include <nbrtos.h>

void InterfaceSpecificServer(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int interfaceNum = 1; // Listen on interface 1

    printf("Creating server on interface %d\n", interfaceNum);
    printf("Interface IP: %s\n", GetInterfaceIP(interfaceNum).print());

    // Listen on interface 1, port 8080
    int fdListen = listenvia(INADDR_ANY, 8080, interfaceNum, 5);

    if (fdListen < 0) {
        printf("Failed to create listener: %d\n", fdListen);
        return;
    }

    printf("Server listening on interface %d, port 8080\n", interfaceNum);
    printf("Waiting for connections...\n\n");

    while (1) {
        IPADDR clientAddr;
        uint16_t clientPort;
        int fdClient = accept(fdListen, &clientAddr, &clientPort, 0);

        if (fdClient > 0) {
            printf("Client connected: %s:%d\n",
                clientAddr.print(), clientPort);

            // Verify connection came through correct interface
            int connInterface = TcpGetSocketInterface(fdClient);
            printf(" Connection via interface: %d\n", connInterface);

            if (connInterface == interfaceNum) {
                printf(" Verified: Correct interface\n");
            }

            // Handle client...
            const char *msg = "Hello from interface-specific server\r\n";
            write(fdClient, msg, strlen(msg));

            close(fdClient);
        }
    }

    close(fdListen);
}
```

Dual-Homed Server - Separate Control and Data Networks

```
#include <tcp.h>
#include <init.h>
#include <networkinterface.h>
#include <nbrtos.h>

// Interface 0: Control network (192.168.1.x)
// Interface 1: Data network (10.0.0.x)

void DualHomedServer(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    printf("Dual-homed server starting...\n");
```

```

printf("Control network (IF0): %s\n", GetInterfaceIP(0).print());
printf("Data network (IF1): %s\n\n", GetInterfaceIP(1).print());

// Create control server on interface 0, port 9000
int fdControl = listenvia(INADDR_ANY, 9000, 0, 5);
if (fdControl < 0) {
    printf("Failed to create control listener: %d\n", fdControl);
    return;
}
printf("Control server: interface 0, port 9000\n");

// Create data server on interface 1, port 8080
int fdData = listenvia(INADDR_ANY, 8080, 1, 10);
if (fdData < 0) {
    printf("Failed to create data listener: %d\n", fdData);
    close(fdControl);
    return;
}
printf("Data server: interface 1, port 8080\n\n");

while (1) {
    // Check for control connections (short timeout)
    IPADDR addr;
    uint16_t port;

    int fd = accept(fdControl, &addr, &port, TICKS_PER_SECOND / 10);
    if (fd > 0) {
        printf("Control connection from %s:%d\n", addr.print(), port);

        // Handle control commands
        char cmd[128];
        int n = read(fd, cmd, sizeof(cmd) - 1);
        if (n > 0) {
            cmd[n] = '\0';
            printf("Command: %s", cmd);

            const char *resp = "OK\r\n";
            write(fd, resp, strlen(resp));
        }

        close(fd);
    }

    // Check for data connections (short timeout)
    fd = accept(fdData, &addr, &port, TICKS_PER_SECOND / 10);
    if (fd > 0) {
        printf("Data connection from %s:%d\n", addr.print(), port);

        // Handle bulk data transfer
        char buffer[1024];
        int n = read(fd, buffer, sizeof(buffer));
        if (n > 0) {
            printf("Received %d bytes of data\n", n);
        }

        close(fd);
    }
}
}

```

Server on Each Interface

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>
#include <nbrtos.h>

#define MAX_INTERFACES 4

struct InterfaceServer {
    int interfaceNum;
    int fdListen;
    int connectionsAccepted;
};

void MultiInterfaceServer(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int numInterfaces = GetNumberOfInterfaces();
    InterfaceServer servers[MAX_INTERFACES];

    printf("Creating server on each of %d interfaces\n\n", numInterfaces);

    // Create listening socket on each interface
    for (int i = 0; i < numInterfaces; i++) {
        servers[i].interfaceNum = i;
    }
}

```

```

servers[i].connectionsAccepted = 0;

IPADDR ifAddr = GetInterfaceIP(i);
printf("Interface %d (%s): ", i, ifAddr.print());

// Each interface listens on the same port (8080)
// This works because we're specifying different interfaces
servers[i].fdListen = listenvia(INADDR_ANY, 8080, i, 5);

if (servers[i].fdListen > 0) {
    printf("Listening\n");
} else {
    printf("Failed: %d\n", servers[i].fdListen);
}
}

printf("\nServers ready. All listening on port 8080\n\n");

// Accept connections from all interfaces
while (1) {
    for (int i = 0; i < numInterfaces; i++) {
        if (servers[i].fdListen < 0) continue;

        IPADDR addr;
        uint16_t port;

        // Check each interface with short timeout
        int fd = accept(servers[i].fdListen, &addr, &port,
            TICKS_PER_SECOND / 20);

        if (fd > 0) {
            servers[i].connectionsAccepted++;

            printf("IF%d: Connection from %s:%d (%d)\n",
                i, addr.print(), port,
                servers[i].connectionsAccepted);

            // Send interface-specific response
            char response[128];
            snprintf(response, sizeof(response),
                "Connected to interface %d\r\n", i);
            write(fd, response, strlen(response));

            close(fd);
        }
    }
}
}

```

Interface Validation and Error Handling

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

int CreateInterfaceServer(int interfaceNum, uint16_t port, uint8_t backlog) {
    // Validate interface number
    int maxInterfaces = GetNumberOfInterfaces();

    if (interfaceNum < 0 || interfaceNum >= maxInterfaces) {
        printf("ERROR: Invalid interface number %d\n", interfaceNum);
        printf("Valid range: 0 to %d\n", maxInterfaces - 1);
        return -1;
    }

    // Get interface information
    IPADDR ifAddr = GetInterfaceIP(interfaceNum);

    if (ifAddr == IPADDR::NullIP()) {
        printf("ERROR: Interface %d has no IP address\n", interfaceNum);
        return -1;
    }

    printf("Creating server:\n");
    printf("  Interface: %d\n", interfaceNum);
    printf("  IP Address: %s\n", ifAddr.print());
    printf("  Port: %d\n", port);
    printf("  Backlog: %d\n", backlog);

    // Create listening socket
    int fdListen = listenvia(INADDR_ANY, port, interfaceNum, backlog);

    if (fdListen < 0) {
        printf("ERROR: Failed to create listener: %d\n", fdListen);

        switch (fdListen) {
            case TCP_ERR_NONE_AVAIL:

```



```

        printf(" No sockets available\n");
        break;
    case TCP_ERR_NOCON:
        printf(" Port %d may already be in use\n", port);
        break;
    default:
        printf(" Unknown error\n");
        break;
    }

    return -1;
}

printf("Server created successfully (fd=%d)\n", fdListen);
return fdListen;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int fdListen = CreateInterfaceServer(0, 8080, 10);

    if (fdListen > 0) {
        // Use server...
        close(fdListen);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Interface Failover Server

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>
#include <nbrtos.h>

void FailoverServer(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int primaryIF = 0;
    int backupIF = 1;
    int activeIF = primaryIF;
    int fdListen = -1;

    printf("Failover server starting\n");
    printf("Primary interface: %d (%s)\n",
        primaryIF, GetInterfaceIP(primaryIF).print());
    printf("Backup interface:  %d (%s)\n\n",
        backupIF, GetInterfaceIP(backupIF).print());

    // Try to start on primary interface
    fdListen = listenvia(INADDR_ANY, 8080, primaryIF, 5);

    if (fdListen < 0) {
        printf("Primary interface failed, using backup\n");
        activeIF = backupIF;
        fdListen = listenvia(INADDR_ANY, 8080, backupIF, 5);
    } else {
        printf("Running on primary interface\n");
    }

    if (fdListen < 0) {
        printf("Both interfaces failed: %d\n", fdListen);
        return;
    }

    uint32_t lastHealthCheck = Secs;

    while (1) {
        // Accept connections
        IPADDR addr;
        uint16_t port;
        int fd = accept(fdListen, &addr, &port, TICKS_PER_SECOND);

        if (fd > 0) {
            printf("Connection on IF%d from %s:%d\n",
                activeIF, addr.print(), port);

            // Handle connection...
            close(fd);
        }
    }
}

```

```

// Periodic health check
if ((Secs - lastHealthCheck) >= 30) {
    // Check if we should switch interfaces
    if (activeIF == backupIF) {
        // Currently on backup, try to restore primary
        int testFd = listenvia(INADDR_ANY, 8081, primaryIF, 1);
        if (testFd > 0) {
            printf("\nPrimary interface recovered, switching back\n");
            close(testFd);
            close(fdListen);

            fdListen = listenvia(INADDR_ANY, 8080, primaryIF, 5);
            if (fdListen > 0) {
                activeIF = primaryIF;
                printf("Restored to primary interface\n\n");
            }
        }
    }

    lastHealthCheck = Secs;
}
}
}

```

Per-Interface Connection Limiting

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>
#include <nbrtos.h>

#define MAX_CONNECTIONS_PER_IF 10

struct InterfaceLimits {
    int interfaceNum;
    int fdListen;
    int activeConnections;
    int maxConnections;
    int rejectedCount;
};

void LimitedInterfaceServer(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int numInterfaces = GetNumberOfInterfaces();
    InterfaceLimits limits[4]; // Support up to 4 interfaces

    printf("Creating rate-limited servers\n");
    printf("Max connections per interface: %d\n\n", MAX_CONNECTIONS_PER_IF);

    // Initialize each interface
    for (int i = 0; i < numInterfaces; i++) {
        limits[i].interfaceNum = i;
        limits[i].activeConnections = 0;
        limits[i].maxConnections = MAX_CONNECTIONS_PER_IF;
        limits[i].rejectedCount = 0;

        limits[i].fdListen = listenvia(INADDR_ANY, 8080, i, 5);

        if (limits[i].fdListen > 0) {
            printf("IF%d (%s): Ready\n",
                i, GetInterfaceIP(i).print());
        }
    }

    printf("\nServers running\n\n");

    while (1) {
        for (int i = 0; i < numInterfaces; i++) {
            if (limits[i].fdListen < 0) continue;

            // Check if we can accept more connections
            if (limits[i].activeConnections >= limits[i].maxConnections) {
                // At limit - just peek to reject
                IPADDR addr;
                uint16_t port;
                int fd = accept(limits[i].fdListen, &addr, &port, 0);

                if (fd > 0) {
                    limits[i].rejectedCount++;
                    const char *msg = "Server busy\r\n";
                    write(fd, msg, strlen(msg));
                    close(fd);

                    printf("IF%d: Rejected connection (at limit %d/%d)\n",
                        i, limits[i].activeConnections,

```

```

        limits[i].maxConnections);
    }
    continue;
}

// Accept new connection
IPADDR addr;
uint16_t port;
int fd = accept(limits[i].fdListen, &addr, &port,
               TICKS_PER_SECOND / 20);

if (fd > 0) {
    limits[i].activeConnections++;

    printf("IF%d: Accepted %s:%d (%d/%d active)\n",
           i, addr.print(), port,
           limits[i].activeConnections,
           limits[i].maxConnections);

    // Handle connection (simplified - normally would track)
    OSTimeDly(TICKS_PER_SECOND * 2);

    close(fd);
    limits[i].activeConnections--;
}
}
}
}

```

Interface Statistics Monitoring

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>
#include <nbrtos.h>

struct InterfaceStats {
    int interfaceNum;
    int fdListen;
    uint32_t totalConnections;
    uint32_t bytesReceived;
    uint32_t bytesSent;
};

void MonitoredInterfaceServer(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int numInterfaces = GetNumberOfInterfaces();
    InterfaceStats stats[4]; // Support up to 4 interfaces

    // Initialize
    for (int i = 0; i < numInterfaces; i++) {
        stats[i].interfaceNum = i;
        stats[i].totalConnections = 0;
        stats[i].bytesReceived = 0;
        stats[i].bytesSent = 0;

        stats[i].fdListen = listenvia(INADDR_ANY, 8080, i, 5);
    }

    printf("Monitored interface servers running\n\n");

    uint32_t lastStatsTime = Secs;

    while (1) {
        for (int i = 0; i < numInterfaces; i++) {
            if (stats[i].fdListen < 0) continue;

            IPADDR addr;
            uint16_t port;
            int fd = accept(stats[i].fdListen, &addr, &port,
                           TICKS_PER_SECOND / 20);

            if (fd > 0) {
                stats[i].totalConnections++;

                // Handle connection
                char buffer[512];
                int n = read(fd, buffer, sizeof(buffer));
                if (n > 0) {
                    stats[i].bytesReceived += n;

                    // Echo back
                    writeall(fd, buffer, n);
                    stats[i].bytesSent += n;
                }
            }
        }
    }
}

```

```

        close(fd);
    }
}

// Print statistics every 30 seconds
if ((Secs - lastStatsTime) >= 30) {
    printf("\n=== Interface Statistics ===\n");

    for (int i = 0; i < numInterfaces; i++) {
        if (stats[i].fdListen < 0) continue;

        printf("Interface %d (%s):\n",
            i, GetInterfaceIP(i).print());
        printf("  Connections: %lu\n", stats[i].totalConnections);
        printf("  RX: %lu bytes\n", stats[i].bytesReceived);
        printf("  TX: %lu bytes\n", stats[i].bytesSent);
    }

    printf("\n");
    lastStatsTime = Secs;
}
}
}

```

Dynamic Interface Server Allocation

```

#include <tcp.h>
#include <init.h>
#include <networkinterface.h>

// Automatically find best interface and create server
int CreateServerOnBestInterface(uint16_t port) {
    int numInterfaces = GetNumberOfInterfaces();

    printf("Finding best interface for server...\n");

    // Try each interface in order
    for (int i = 0; i < numInterfaces; i++) {
        IPADDR ifAddr = GetInterfaceIP(i);

        printf("Trying interface %d (%s)...", i, ifAddr.print());

        // Check if interface has valid IP
        if (ifAddr == IPADDR::NullIP()) {
            printf(" No IP\n");
            continue;
        }

        // Try to create server
        int fd = listenvia(INADDR_ANY, port, i, 5);

        if (fd > 0) {
            printf(" Success!\n");
            printf("Server created on interface %d, port %d\n", i, port);
            return fd;
        }

        printf(" Failed: %d\n", fd);
    }

    printf("No suitable interface found\n");
    return -1;
}

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int fdListen = CreateServerOnBestInterface(8080);

    if (fdListen > 0) {
        // Use server...
        close(fdListen);
    }

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

See also

[listen\(\)](#), [listenvia\(const IPADDR&, uint16_t, const IPADDR&, uint8_t\)](#), [accept\(\)](#), [close\(\)](#), [GetNumberOfInterfaces\(\)](#), [GetInterfaceIP\(\)](#), [TcpGetSocketInterface\(\)](#)

14.53.14.25 NoBlockConnect()

```
int NoBlockConnect (
    const IPADDR & ipAddress,
    uint16_t remotePort) [inline]
#include <tcp.h>
```

Initiate a non-blocking TCP connection.

Creates a TCP socket and initiates a connection to the remote host, returning immediately without waiting for the connection to complete. The application must poll the socket state using [TcpGetSocketState\(\)](#) to determine when the connection is established (TCP_STATE_ESTABLISHED). The connection status can also be checked using [writeavail\(\)](#).

Parameters

<i>ipAddress</i>	IP address of the remote host
<i>remotePort</i>	Port number on the remote host

Returns

Positive file descriptor on success, negative [TCP Socket Status](#) error code on failure

Notes:

- This function returns immediately and does not block waiting for the connection to establish. The socket will be in a connecting state, and the application must poll its state to determine when it becomes ready for data transfer.
- The system will automatically select the local interface and port for the connection. Use [NoBlockConnectVia\(\)](#) if you need control over which interface is used.

Warning

Always call [close\(\)](#) to free system resources, even if the socket enters the TCP_STATE_CLOSING state. Failure to close will cause a resource leak.

See also

[NoBlockConnectVia\(\)](#), [TcpGetSocketState\(\)](#), [writeavail\(\)](#), [connect\(\)](#), [connectvia\(\)](#), [connectwlocal\(\)](#), [close\(\)](#)

Expand for Example Usage**Examples****Basic Non-Blocking Connection with State Polling**

```
#include <tcp.h>
#include <nbrtos.h>

void NonBlockingClientTask(void *pd)
{
    // Target server details
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    // Initiate non-blocking connection
    int fd = NoBlockConnect(serverIp, serverPort);

    if (fd < 0)
    {
        printf("Failed to initiate connection: %d\r\n", fd);
        return;
    }

    printf("Connection initiated to %I:%d (fd=%d)\r\n", serverIp, serverPort, fd);

    // Poll for connection establishment (timeout after 10 seconds)
    uint32_t startTime = Secs;
    bool connected = false;

    while ((Secs - startTime) < 10)
```

```

{
    TcpState state = TcpGetSocketState(fd);

    if (state == TCP_STATE_ESTABLISHED)
    {
        connected = true;
        printf("Connection established!\r\n");
        break;
    }
    else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
    {
        printf("Connection failed (state: %d)\r\n", state);
        break;
    }

    OSTimeDly(TICKS_PER_SECOND / 10); // Poll every 100ms
}

if (!connected)
{
    printf("Connection timeout\r\n");
    close(fd);
    return;
}

// Connection successful - send data
const char *message = "Hello, Server!\r\n";
write(fd, message, strlen(message));

// Receive response
char buffer[256];
int bytesRead = read(fd, buffer, sizeof(buffer) - 1, TICKS_PER_SECOND * 5);

if (bytesRead > 0)
{
    buffer[bytesRead] = '\0';
    printf("Received: %s\r\n", buffer);
}

close(fd);
}

```

Using writeavail() to Check Connection Status

```

#include <tcp.h>
#include <nbrtos.h>

void CheckConnectionWithWriteavail(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 80;

    int fd = NoBlockConnect(serverIp, serverPort);

    if (fd < 0)
    {
        printf("Connection initiation failed: %d\r\n", fd);
        return;
    }

    printf("Connecting to %I:%d...\r\n", serverIp, serverPort);

    // Wait for connection using writeavail()
    // writeavail() returns > 0 when socket is ready to write (connected)
    uint32_t startTime = Secs;
    bool connected = false;

    while ((Secs - startTime) < 10)
    {
        int available = writeavail(fd);

        if (available > 0)
        {
            // Socket is ready for writing - connection established
            connected = true;
            printf("Connection established (write buffer: %d bytes)\r\n", available);
            break;
        }
        else if (available < 0)
        {
            // Negative value indicates error or closed socket
            printf("Connection failed: %d\r\n", available);
            break;
        }

        OSTimeDly(TICKS_PER_SECOND / 10);
    }
}

```

```

    if (!connected)
    {
        printf("Connection timeout\r\n");
    }

    close(fd);
}

```

Multiple Simultaneous Non-Blocking Connections

```

#include <tcp.h>
#include <nbrtos.h>

#define MAX_CONNECTIONS 5

struct Connection
{
    int fd;
    IPADDR serverIp;
    uint16_t serverPort;
    uint32_t startTime;
    bool established;
};

void MultipleConnectionsTask(void *pd)
{
    Connection connections[MAX_CONNECTIONS];

    // Initialize connections to different servers
    const char *servers[] = {
        "192.168.1.100",
        "192.168.1.101",
        "192.168.1.102",
        "192.168.1.103",
        "192.168.1.104"
    };

    // Initiate all connections simultaneously
    for (int i = 0; i < MAX_CONNECTIONS; i++)
    {
        connections[i].serverIp = AsciiToIp(servers[i]);
        connections[i].serverPort = 8080;
        connections[i].startTime = Secs;
        connections[i].established = false;

        connections[i].fd = NoBlockConnect(connections[i].serverIp,
                                           connections[i].serverPort);

        if (connections[i].fd < 0)
        {
            printf("Failed to initiate connection %d: %d\r\n",
                   i, connections[i].fd);
        }
        else
        {
            printf("Initiated connection %d to %I:%d (fd=%d)\r\n",
                   i, connections[i].serverIp, connections[i].serverPort,
                   connections[i].fd);
        }
    }

    // Poll all connections until established or timeout
    int establishedCount = 0;
    const uint32_t timeout = 10; // 10 seconds

    while (establishedCount < MAX_CONNECTIONS && (Secs - connections[0].startTime) < timeout)
    {
        for (int i = 0; i < MAX_CONNECTIONS; i++)
        {
            if (connections[i].fd < 0 || connections[i].established)
                continue;

            TcpState state = TcpGetSocketState(connections[i].fd);

            if (state == TCP_STATE_ESTABLISHED)
            {
                connections[i].established = true;
                establishedCount++;
                printf("Connection %d established!\r\n", i);

                // Send initial message
                char msg[64];
                snprintf(msg, sizeof(msg), "Hello from client %d\r\n", i);
                write(connections[i].fd, msg, strlen(msg));
            }
            else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)

```

```

        {
            printf("Connection %d failed\r\n", i);
            close(connections[i].fd);
            connections[i].fd = -1;
            establishedCount++; // Count as "done" even if failed
        }
    }

    OSTimeDly(TICKS_PER_SECOND / 20); // Poll every 50ms
}

printf("Connection phase complete\r\n");

// Clean up all connections
for (int i = 0; i < MAX_CONNECTIONS; i++)
{
    if (connections[i].fd > 0)
    {
        close(connections[i].fd);
    }
}
}

```

Non-Blocking Connection with select() for Event-Driven Design

```

#include <tcp.h>
#include <nbrtos.h>

void EventDrivenConnectionTask(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = NoBlockConnect(serverIp, serverPort);

    if (fd < 0)
    {
        printf("Failed to initiate connection: %d\r\n", fd);
        return;
    }

    printf("Connecting to %I:%d...\r\n", serverIp, serverPort);

    // Use select() to wait for socket to become writable (connected)
    fd_set write_fds;
    fd_set error_fds;

    FD_ZERO(&write_fds);
    FD_ZERO(&error_fds);
    FD_SET(fd, &write_fds);
    FD_SET(fd, &error_fds);

    // Wait up to 10 seconds for connection
    int result = select(fd + 1, NULL, &write_fds, &error_fds, TICKS_PER_SECOND * 10);

    if (result < 0)
    {
        printf("select() error: %d\r\n", result);
        close(fd);
        return;
    }
    else if (result == 0)
    {
        printf("Connection timeout\r\n");
        close(fd);
        return;
    }

    if (FD_ISSET(fd, &error_fds))
    {
        printf("Connection error\r\n");
        close(fd);
        return;
    }

    if (FD_ISSET(fd, &write_fds))
    {
        // Verify connection is truly established
        TcpState state = TcpGetSocketState(fd);

        if (state == TCP_STATE_ESTABLISHED)
        {
            printf("Connection established via select()!\r\n");

            // Connection ready - proceed with communication
            const char *request = "GET / HTTP/1.1\r\nHost: server\r\n\r\n";
            write(fd, request, strlen(request));
        }
    }
}

```



```

        // Read response...
    }
    else
    {
        printf("Socket writable but not established (state: %d)\r\n", state);
    }
}

close(fd);
}

```

14.53.14.26 NoBlockConnectVia() [1/2]

```

int NoBlockConnectVia (
    const IPADDR & ipAddress,
    uint16_t remotePort,
    const IPADDR & interfaceIpAddress = IPADDR::NullIP()) [inline]
#include <tcp.h>

```

Initiate a non-blocking TCP connection via a specific interface.

Creates a TCP socket and initiates a connection to the remote host using the specified local interface IP address. The function returns immediately without waiting for the connection to complete. The application must poll the socket state using [TcpGetSocketState\(\)](#) to determine when the connection is established (TCP_STATE_ESTABLISHED).

Warning

Always call [close\(\)](#) to free system resources, even if the socket enters the TCP_STATE_CLOSING state. Failure to close will cause a resource leak.

Parameters

<i>ipAddress</i>	IP address of the remote host
<i>remotePort</i>	Port number on the remote host
<i>interfaceIpAddress</i>	IP address of the local interface to use (default: IPADDR::NullIP() for automatic selection based on routing)

Returns

Positive file descriptor on success, negative [TCP Socket Status](#) error code on failure

See also

[NoBlockConnect\(\)](#), [TcpGetSocketState\(\)](#), [writeavail\(\)](#), [connect\(\)](#), [connectvia\(\)](#), [connectwlocal\(\)](#)

14.53.14.27 NoBlockConnectVia() [2/2]

```

int NoBlockConnectVia (
    const IPADDR & ipAddress,
    uint16_t remotePort,
    int ifnum) [inline]
#include <tcp.h>

```

Initiate a non-blocking TCP connection via a specific interface number.

Creates a TCP socket and initiates a connection to the remote host using the specified local interface number. The function returns immediately without waiting for the connection to complete. The interface number overrides routing decisions based on destination IP address. The application must poll the socket state using [TcpGetSocketState\(\)](#) to determine when the connection is established (TCP_STATE_ESTABLISHED).

Parameters

<i>ipAddress</i>	IP address of the remote host
<i>remotePort</i>	Port number on the remote host
<i>ifnum</i>	Network interface number to use. Overrides all routing decisions

Returns

Positive file descriptor on success, negative [TCP Socket Status](#) error code on failure

Notes:

- This function is useful when you have multiple network interfaces (e.g., Ethernet and WiFi) and need to force the connection through a specific interface regardless of routing tables.
- Interface numbers are zero-based. Use [GetNumInterfaces\(\)](#) to determine the total number of available interfaces and [GetIPAddress\(ifnum\)](#) to verify the interface has a valid IP.

Warning

- Always call [close\(\)](#) to free system resources, even if the socket enters the TCP_STATE_CLOSING state. Failure to close will cause a resource leak.
- If the specified interface is not active or does not have a valid IP address, the connection will fail. Always validate the interface before use.

See also

[NoBlockConnect\(\)](#), [TcpGetSocketState\(\)](#), [writeavail\(\)](#), [connect\(\)](#), [connectvia\(\)](#), [connectwlocal\(\)](#), [GetNumInterfaces\(\)](#), [GetIPAddress\(\)](#), [close\(\)](#)

Expand for Example Usage**Examples****Basic Non-Blocking Connection via Specific Interface**

```
#include <tcp.h>
#include <netinterface.h>
#include <nbrtos.h>

void ConnectViaEthernetTask(void *pd)
{
    // Force connection through interface 0 (typically Ethernet)
    const int ETHERNET_IF = 0;

    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    // Verify interface is valid and has an IP address
    if (ETHERNET_IF >= GetNumInterfaces())
    {
        printf("Interface %d does not exist\r\n", ETHERNET_IF);
        return;
    }

    IPADDR localIp = GetIPAddress(ETHERNET_IF);
    if (localIp == IPADDR_ANY)
    {
        printf("Interface %d has no IP address\r\n", ETHERNET_IF);
        return;
    }

    printf("Connecting to %I:%d via interface %d (%I)\r\n",
        serverIp, serverPort, ETHERNET_IF, localIp);

    // Initiate non-blocking connection via specified interface
    int fd = NoBlockConnectVia(serverIp, serverPort, ETHERNET_IF);

    if (fd < 0)
    {
        printf("Failed to initiate connection: %d\r\n", fd);
        return;
    }

    // Poll for connection establishment
    uint32_t startTime = Secs;
    bool connected = false;

    while ((Secs - startTime) < 10)
    {
        TcpState state = TcpGetSocketState(fd);

        if (state == TCP_STATE_ESTABLISHED)
        {

```

```

        connected = true;
        printf("Connection established via interface %d!\r\n", ETHERNET_IF);
        break;
    }
    else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
    {
        printf("Connection failed (state: %d)\r\n", state);
        break;
    }

    OSTimeDly(TICKS_PER_SECOND / 10);
}

if (!connected)
{
    printf("Connection timeout\r\n");
    close(fd);
    return;
}

// Send data and receive response
const char *message = "Hello via Ethernet!\r\n";
write(fd, message, strlen(message));

close(fd);
}

```

Multi-Interface Redundancy - Try Primary, Fallback to Secondary

```

#include <tcp.h>
#include <netinterface.h>
#include <nbrtos.h>

int ConnectWithFailover(const IPADDR &serverIp, uint16_t serverPort,
                        int primaryIf, int secondaryIf)
{
    int fd = -1;

    // Try primary interface first
    if (primaryIf < GetNumInterfaces() && GetIPAddress(primaryIf) != IPADDR_ANY)
    {
        printf("Attempting connection via primary interface %d\r\n", primaryIf);
        fd = NoBlockConnectVia(serverIp, serverPort, primaryIf);

        if (fd > 0)
        {
            // Wait up to 5 seconds for primary connection
            uint32_t startTime = Secs;
            while ((Secs - startTime) < 5)
            {
                TcpState state = TcpGetSocketState(fd);

                if (state == TCP_STATE_ESTABLISHED)
                {
                    printf("Connected via primary interface %d\r\n", primaryIf);
                    return fd;
                }
                else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
                {
                    printf("Primary interface connection failed\r\n");
                    close(fd);
                    fd = -1;
                    break;
                }

                OSTimeDly(TICKS_PER_SECOND / 10);
            }

            if (fd > 0)
            {
                printf("Primary interface connection timeout\r\n");
                close(fd);
                fd = -1;
            }
        }
    }

    // Primary failed, try secondary interface
    if (secondaryIf < GetNumInterfaces() && GetIPAddress(secondaryIf) != IPADDR_ANY)
    {
        printf("Attempting connection via secondary interface %d\r\n", secondaryIf);
        fd = NoBlockConnectVia(serverIp, serverPort, secondaryIf);

        if (fd > 0)
        {
            uint32_t startTime = Secs;
            while ((Secs - startTime) < 5)

```

```

    {
        TcpState state = TcpGetSocketState(fd);

        if (state == TCP_STATE_ESTABLISHED)
        {
            printf("Connected via secondary interface %d\r\n", secondaryIf);
            return fd;
        }
        else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
        {
            printf("Secondary interface connection failed\r\n");
            close(fd);
            return -1;
        }

        OSTimeDly(TICKS_PER_SECOND / 10);
    }

    printf("Secondary interface connection timeout\r\n");
    close(fd);
}

return -1;
}

void FailoverExample(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    // Try Ethernet (0) first, fallback to WiFi (1)
    int fd = ConnectWithFailover(serverIp, serverPort, 0, 1);

    if (fd > 0)
    {
        const char *message = "Connected with failover!\r\n";
        write(fd, message, strlen(message));
        close(fd);
    }
    else
    {
        printf("All connection attempts failed\r\n");
    }
}

```

Simultaneous Connections via Different Interfaces

```

#include <tcp.h>
#include <netinterface.h>
#include <nbrtos.h>

void DualInterfaceConnectionTask(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    const int ETHERNET_IF = 0;
    const int WIFI_IF = 1;

    // Verify both interfaces are available
    if (GetNumInterfaces() < 2)
    {
        printf("Need at least 2 interfaces for this example\r\n");
        return;
    }

    IPADDR ethernetIp = GetIPAddress(ETHERNET_IF);
    IPADDR wifiIp = GetIPAddress(WIFI_IF);

    if (ethernetIp == IPADDR_ANY || wifiIp == IPADDR_ANY)
    {
        printf("Both interfaces must have valid IP addresses\r\n");
        return;
    }

    printf("Initiating connections via both interfaces:\r\n");
    printf(" Ethernet (%d): %I\r\n", ETHERNET_IF, ethernetIp);
    printf("  WiFi (%d): %I\r\n", WIFI_IF, wifiIp);

    // Start both connections simultaneously
    int ethernetFd = NoBlockConnectVia(serverIp, serverPort, ETHERNET_IF);
    int wifiFd = NoBlockConnectVia(serverIp, serverPort, WIFI_IF);

    if (ethernetFd < 0 && wifiFd < 0)
    {
        printf("Failed to initiate any connections\r\n");
    }
}

```

```

        return;
    }

    // Wait for at least one connection to establish
    uint32_t startTime = Secs;
    bool ethernetConnected = false;
    bool wifiConnected = false;

    while ((Secs - startTime) < 10 && !(ethernetConnected && wifiConnected))
    {
        if (ethernetFd > 0 && !ethernetConnected)
        {
            TcpState state = TcpGetSocketState(ethernetFd);
            if (state == TCP_STATE_ESTABLISHED)
            {
                ethernetConnected = true;
                printf("Ethernet connection established!\r\n");

                write(ethernetFd, "Via Ethernet\r\n", 14);
            }
            else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
            {
                close(ethernetFd);
                ethernetFd = -1;
            }
        }

        if (wifiFd > 0 && !wifiConnected)
        {
            TcpState state = TcpGetSocketState(wifiFd);
            if (state == TCP_STATE_ESTABLISHED)
            {
                wifiConnected = true;
                printf("WiFi connection established!\r\n");

                write(wifiFd, "Via WiFi\r\n", 10);
            }
            else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
            {
                close(wifiFd);
                wifiFd = -1;
            }
        }

        OSTimeDly(TICKS_PER_SECOND / 10);
    }

    // Report results
    printf("Connection results:\r\n");
    printf("  Ethernet: %s\r\n", ethernetConnected ? "Connected" : "Failed");
    printf("  WiFi: %s\r\n", wifiConnected ? "Connected" : "Failed");

    // Clean up
    if (ethernetFd > 0) close(ethernetFd);
    if (wifiFd > 0) close(wifiFd);
}

```

Interface Selection Based on Network Conditions

```

#include <tcp.h>
#include <netinterface.h>
#include <nbrtos.h>

// Select best interface based on link status and speed
int SelectBestInterface(void)
{
    int bestInterface = -1;
    uint32_t bestSpeed = 0;

    for (int i = 0; i < GetNumInterfaces(); i++)
    {
        // Check if interface has valid IP
        IPADDR ip = GetIPAddress(i);
        if (ip == IPADDR_ANY)
            continue;

        // Check link status
        InterfaceBlock *ib = GetInterfaceBlock(i);
        if (ib == nullptr || !ib->IsLinkActive())
            continue;

        // Get interface speed (if available)
        uint32_t speed = ib->GetSpeed();

        if (speed > bestSpeed)
        {
            bestSpeed = speed;
        }
    }
}

```

```

        bestInterface = i;
    }
}

return bestInterface;
}

void SmartConnectionTask(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    // Automatically select best interface
    int ifnum = SelectBestInterface();

    if (ifnum < 0)
    {
        printf("No suitable interface available\r\n");
        return;
    }

    IPADDR localIp = GetIPAddress(ifnum);
    printf("Selected interface %d (%I) for connection\r\n", ifnum, localIp);

    int fd = NoBlockConnectVia(serverIp, serverPort, ifnum);

    if (fd < 0)
    {
        printf("Connection initiation failed: %d\r\n", fd);
        return;
    }

    // Wait for connection
    uint32_t startTime = Secs;
    while ((Secs - startTime) < 10)
    {
        TcpState state = TcpGetSocketState(fd);

        if (state == TCP_STATE_ESTABLISHED)
        {
            printf("Connected successfully!\r\n");

            // Use connection...
            const char *msg = "Hello from smart interface selection!\r\n";
            write(fd, msg, strlen(msg));

            break;
        }
        else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
        {
            printf("Connection failed\r\n");
            break;
        }

        OSTimeDly(TICKS_PER_SECOND / 10);
    }

    close(fd);
}

```

14.53.14.28 NoBlockConnectwlocal()

```

int NoBlockConnectwlocal (
    const IPADDR & ipAddress,
    uint16_t localPort,
    uint16_t remotePort,
    IPADDR interfaceIpAddress = IPADDR::NullIP(),
    int ifn = -1) [inline]

```

#include <tcp.h>

Initiate a non-blocking TCP connection with full local parameter control.

Creates a TCP socket with complete control over connection parameters and initiates a connection to the remote host. The function returns immediately without waiting for the connection to complete. The application must poll the socket state using [TcpGetSocketState\(\)](#) to determine when the connection is established (TCP_STATE_↵ESTABLISHED).

Parameters

<i>ipAddress</i>	IP address of the remote host

Parameters

<i>localPort</i>	Local port number to bind (recommend 0 for automatic assignment). Warning: Non-zero values can cause connection failures with half-open sockets on the remote side
<i>remotePort</i>	Port number on the remote host
<i>interfaceIpAddress</i>	IP address of local interface (use <code>IPADDR::NullIP()</code> if using <i>ifn</i>)
<i>ifn</i>	Network interface number (use -1 if using <i>interfaceIpAddress</i>)

Returns

Positive file descriptor on success, negative [TCP Socket Status](#) error code on failure

Notes: This function provides the most control over connection parameters but is rarely needed. For most applications, use [NoBlockConnect\(\)](#) or [NoBlockConnectVia\(\)](#) instead.

- You can specify the interface either by IP address (*interfaceIpAddress*) or by interface number (*ifn*), but not both. If both are specified, the interface number takes precedence.

Warning

- Always call [close\(\)](#) to free system resources, even if the socket enters the `TCP_STATE_CLOSING` state. Failure to close will cause a resource leak.
- Specifying a local port number is not recommended. Using a fixed local port can cause connection failures when the remote host has a half-open socket remaining from a previous connection. Always use 0 for automatic port assignment unless you have a specific protocol requirement.

See also

[TcpGetSocketState\(\)](#), [NoBlockConnect\(\)](#), [NoBlockConnectVia\(\)](#), [writeavail\(\)](#), [close\(\)](#)

Expand for Example Usage

Examples

Basic Usage - Let System Assign Local Port (Recommended)

```
#include <tcp.h>
#include <netinterface.h>
#include <nbrtos.h>

void RecommendedUsageTask(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t remotePort = 8080;

    // Get local interface IP
    IPADDR localIp = GetIPAddress(0);

    // Connect with automatic local port assignment (recommended)
    // localPort = 0 means system will choose an available ephemeral port
    int fd = NoBlockConnectwlocal(serverIp, 0, remotePort, localIp);

    if (fd < 0)
    {
        printf("Connection initiation failed: %d\r\n", fd);
        return;
    }

    printf("Connecting to %I:%d from %I:<auto>\r\n",
        serverIp, remotePort, localIp);

    // Wait for connection to establish
    uint32_t startTime = Secs;
    bool connected = false;

    while ((Secs - startTime) < 10)
    {
        TcpState state = TcpGetSocketState(fd);

        if (state == TCP_STATE_ESTABLISHED)
        {
            connected = true;
            printf("Connection established!\r\n");
            break;
        }
    }
}
```

```

    }
    else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
    {
        printf("Connection failed (state: %d)\r\n", state);
        break;
    }

    OSTimeDly(TICKS_PER_SECOND / 10);
}

if (connected)
{
    const char *message = "Hello, Server!\r\n";
    write(fd, message, strlen(message));
}

close(fd);
}

```

Using Interface Number Instead of IP Address

```

#include <tcp.h>
#include <netinterface.h>
#include <nbrtos.h>

void ConnectViaInterfaceNumberTask(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t remotePort = 8080;
    const int ETHERNET_IF = 0;

    // Verify interface exists
    if (ETHERNET_IF >= GetNumInterfaces())
    {
        printf("Interface %d does not exist\r\n", ETHERNET_IF);
        return;
    }

    // Connect using interface number instead of IP address
    // Pass IPADDR::NullIP() for interfaceIpAddress when using ifn
    int fd = NoBlockConnectwlocal(serverIp, 0, remotePort,
                                  IPADDR::NullIP(), ETHERNET_IF);

    if (fd < 0)
    {
        printf("Connection initiation failed: %d\r\n", fd);
        return;
    }

    IPADDR localIp = GetIPAddress(ETHERNET_IF);
    printf("Connecting to %I:%d via interface %d (%I)\r\n",
           serverIp, remotePort, ETHERNET_IF, localIp);

    // Wait for connection
    uint32_t startTime = Secs;
    while ((Secs - startTime) < 10)
    {
        TcpState state = TcpGetSocketState(fd);

        if (state == TCP_STATE_ESTABLISHED)
        {
            printf("Connected via interface %d!\r\n", ETHERNET_IF);

            const char *msg = "Hello from specific interface!\r\n";
            write(fd, msg, strlen(msg));
            break;
        }
        else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
        {
            printf("Connection failed\r\n");
            break;
        }

        OSTimeDly(TICKS_PER_SECOND / 10);
    }

    close(fd);
}

```

Fixed Local Port (Not Recommended - Shows Potential Issues)

```

#include <tcp.h>
#include <netinterface.h>
#include <nbrtos.h>

void FixedLocalPortExample(void *pd)
{

```



```

IPADDR serverIp = AsciiToIp("192.168.1.100");
uint16_t remotePort = 8080;
uint16_t fixedLocalPort = 5000; // WARNING: Not recommended!

IPADDR localIp = GetIPAddress(0);

// Attempt 1: Use fixed local port
printf("Attempting connection with fixed local port %d...\r\n", fixedLocalPort);

int fd = NoBlockConnectwlocal(serverIp, fixedLocalPort, remotePort, localIp);

if (fd < 0)
{
    printf("Connection with fixed port failed: %d\r\n", fd);

    if (fd == TCP_ERR_PORT_IN_USE)
    {
        printf("Local port %d is already in use\r\n", fixedLocalPort);
    }

    // Retry with automatic port assignment (recommended approach)
    printf("Retrying with automatic port assignment...\r\n");
    fd = NoBlockConnectwlocal(serverIp, 0, remotePort, localIp);

    if (fd < 0)
    {
        printf("Connection still failed: %d\r\n", fd);
        return;
    }
}

// Wait for connection
uint32_t startTime = Secs;
bool connected = false;

while ((Secs - startTime) < 10)
{
    TcpState state = TcpGetSocketState(fd);

    if (state == TCP_STATE_ESTABLISHED)
    {
        connected = true;
        printf("Connection established\r\n");
        break;
    }
    else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
    {
        printf("Connection failed after initiation\r\n");
        break;
    }

    OSTimeDly(TICKS_PER_SECOND / 10);
}

if (connected)
{
    const char *msg = "Connection succeeded\r\n";
    write(fd, msg, strlen(msg));
}

close(fd);

// Demonstrate the half-open socket problem
printf("\r\nDemonstrating half-open socket issue...\r\n");
printf("If server still has half-open socket, reconnection with\r\n");
printf("same local port will fail. This is why automatic port\r\n");
printf("assignment (localPort=0) is strongly recommended.\r\n");
}

```

Protocol Requiring Specific Local Port (Rare Use Case)

```

#include <tcp.h>
#include <netinterface.h>
#include <nbrtos.h>

// Example: Some legacy protocols require client to use specific port
void LegacyProtocolClient(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t remotePort = 8080;
    uint16_t requiredLocalPort = 6000; // Protocol requirement

    IPADDR localIp = GetIPAddress(0);

    printf("Legacy protocol requires local port %d\r\n", requiredLocalPort);

    // First, ensure port is not already in use

```

```

// In production, implement proper port cleanup and retry logic

int fd = NoBlockConnectwlocal(serverIp, requiredLocalPort, remotePort, localIp);

if (fd < 0)
{
    if (fd == TCP_ERR_PORT_IN_USE)
    {
        printf("Required port %d is in use. Wait for TIME_WAIT...\r\n",
            requiredLocalPort);

        // In real application, you might:
        // 1. Wait for TIME_WAIT to expire (typically 2*MSL = 240s)
        // 2. Use SO_REUSEADDR if available
        // 3. Inform user of the conflict
        return;
    }
    else
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }
}

printf("Initiated connection from %I:%d to %I:%d\r\n",
    localIp, requiredLocalPort, serverIp, remotePort);

// Wait for connection with timeout
uint32_t startTime = Secs;
bool connected = false;

while ((Secs - startTime) < 10)
{
    TcpState state = TcpGetSocketState(fd);

    if (state == TCP_STATE_ESTABLISHED)
    {
        connected = true;
        printf("Connection established with required local port\r\n");

        // Implement protocol-specific communication
        const char *protocolMsg = "LEGACY_PROTOCOL_HANDSHAKE\r\n";
        write(fd, protocolMsg, strlen(protocolMsg));

        break;
    }
    else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
    {
        printf("Connection failed during establishment\r\n");
        break;
    }

    OSTimeDly(TICKS_PER_SECOND / 10);
}

if (!connected)
{
    printf("Connection timeout\r\n");
}

close(fd);

// Important: After close(), port enters TIME_WAIT state
// It cannot be reused immediately (typically 240 seconds)
printf("Port %d now in TIME_WAIT - cannot reuse immediately\r\n",
    requiredLocalPort);
}

```

Advanced: Multiple Connections with Interface and Port Control

```

#include <tcp.h>
#include <netinterface.h>
#include <nbrtos.h>

struct ManagedConnection
{
    int fd;
    IPADDR serverIp;
    uint16_t serverPort;
    int interface;
    bool established;
};

void AdvancedMultiConnectionTask(void *pd)
{
    const int NUM_CONNECTIONS = 3;
    ManagedConnection connections[NUM_CONNECTIONS];

```

```

// Initialize connection parameters
const char *servers[] = {
    "192.168.1.100",
    "192.168.1.101",
    "192.168.1.102"
};

// Use different interfaces for load balancing
int interfaces[] = {0, 1, 0}; // Alternate between interfaces

// Initiate all connections
for (int i = 0; i < NUM_CONNECTIONS; i++)
{
    connections[i].serverIp = AsciiToIp(servers[i]);
    connections[i].serverPort = 8080;
    connections[i].interface = interfaces[i];
    connections[i].established = false;

    // Verify interface is valid
    if (interfaces[i] >= GetNumInterfaces())
    {
        printf("Interface %d not available for connection %d\r\n",
            interfaces[i], i);
        connections[i].fd = -1;
        continue;
    }

    IPADDR localIp = GetIPAddress(interfaces[i]);
    if (localIp == IPADDR_ANY)
    {
        printf("Interface %d has no IP for connection %d\r\n",
            interfaces[i], i);
        connections[i].fd = -1;
        continue;
    }

    // Connect with automatic local port, specific interface
    connections[i].fd = NoBlockConnectwlocal(
        connections[i].serverIp,
        0, // Automatic local port (recommended)
        connections[i].serverPort,
        IPADDR::NullIP(),
        connections[i].interface
    );

    if (connections[i].fd < 0)
    {
        printf("Failed to initiate connection %d: %d\r\n",
            i, connections[i].fd);
    }
    else
    {
        printf("Connection %d initiated to %I:%d via interface %d\r\n",
            i, connections[i].serverIp, connections[i].serverPort,
            connections[i].interface);
    }
}

// Wait for all connections to establish or fail
uint32_t startTime = Secs;
int establishedCount = 0;

while ((Secs - startTime) < 10 && establishedCount < NUM_CONNECTIONS)
{
    for (int i = 0; i < NUM_CONNECTIONS; i++)
    {
        if (connections[i].fd < 0 || connections[i].established)
            continue;

        TcpState state = TcpGetSocketState(connections[i].fd);

        if (state == TCP_STATE_ESTABLISHED)
        {
            connections[i].established = true;
            establishedCount++;
            printf("Connection %d established!\r\n", i);

            // Send initial data
            char msg[64];
            snprintf(msg, sizeof(msg), "Connection %d via interface %d\r\n",
                i, connections[i].interface);
            write(connections[i].fd, msg, strlen(msg));
        }
        else if (state == TCP_STATE_CLOSED || state == TCP_STATE_CLOSING)
        {
            printf("Connection %d failed\r\n", i);
        }
    }
}

```

```

        close(connections[i].fd);
        connections[i].fd = -1;
        establishedCount++;
    }
}

OSTimeDly(TICKS_PER_SECOND / 20);
}

// Report final status
printf("\r\nFinal connection status:\r\n");
for (int i = 0; i < NUM_CONNECTIONS; i++)
{
    printf("    Connection %d: %s\r\n", i,
           connections[i].established ? "Established" : "Failed");
}

// Clean up all connections
for (int i = 0; i < NUM_CONNECTIONS; i++)
{
    if (connections[i].fd > 0)
    {
        close(connections[i].fd);
    }
}
}

```

14.53.14.29 SetOutOfOrderBuffers()

```

uint8_t SetOutOfOrderBuffers (
    int fd,
    uint8_t max) [inline]
#include <tcp.h>

```

Configure the maximum number of out-of-order buffers for a TCP socket.

Sets the maximum number of out-of-order TCP receive buffers for the socket. TCP may receive packets out of sequence, which must be stored until earlier packets arrive to maintain correct data ordering. Each buffer is approximately 1500 bytes.

The default value is 5 buffers. The maximum is 43 buffers. Increasing this value can improve bulk data throughput on fast networks but may increase TCP latency if connection problems (such as resets) occur.

Parameters

<i>fd</i>	Socket file descriptor
<i>max</i>	Maximum number of out-of-order buffers (1-43, default: 5)

Returns

Status code: 0 on success, negative [TCP Socket Status](#) error code on failure

Notes:

- This setting should be configured before significant data transfer begins, ideally right after the connection is established.
- Higher values improve performance on high-bandwidth, high-latency networks where packet reordering is common, but consume more memory.
- Lower values reduce memory usage but may cause TCP to request retransmissions more frequently when packets arrive out of order.

Warning

Setting this value too high may waste system resources. Setting it too low may reduce throughput on networks with significant packet reordering.

See also

[setsockopt\(\)](#), [getsockopt\(\)](#), [clrsockopt\(\)](#), [SetSocketRxBuffers\(\)](#), [SetSocketTxBuffers\(\)](#)

Expand for Example Usage

Examples

Basic Configuration for High-Speed Transfer

```
#include <tcp.h>
#include <nbrtos.h>

void HighSpeedFileTransfer(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    // Connect to server
    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    printf("Connected to %I:%d\r\n", serverIp, serverPort);

    // Configure for high-speed transfer - increase out-of-order buffers
    uint8_t result = SetOutOfOrderBuffers(fd, 20);

    if (result != 0)
    {
        printf("Failed to set out-of-order buffers: %d\r\n", result);
    }
    else
    {
        printf("Out-of-order buffers set to 20 (30KB buffer space)\r\n");
    }

    // Receive large file
    char buffer[4096];
    uint32_t totalBytes = 0;
    uint32_t startTime = Secs;

    while (1)
    {
        int bytesRead = read(fd, buffer, sizeof(buffer), TICKS_PER_SECOND * 30);

        if (bytesRead <= 0)
        {
            break;
        }

        totalBytes += bytesRead;
    }

    uint32_t duration = Secs - startTime;
    if (duration > 0)
    {
        uint32_t throughputKBps = (totalBytes / 1024) / duration;
        printf("Transfer complete: %lu bytes in %lu sec (%lu KB/s)\r\n",
              totalBytes, duration, throughputKBps);
    }

    close(fd);
}
```

Adaptive Buffer Configuration Based on Network Conditions

```
#include <tcp.h>
#include <nbrtos.h>

void ConfigureBuffersForNetwork(int fd, const char *networkType)
{
    uint8_t bufferCount;

    if (strcmp(networkType, "local") == 0)
    {
        // Local network - low latency, minimal reordering
        bufferCount = 5; // Default
        printf("Local network: using %d out-of-order buffers\r\n", bufferCount);
    }
    else if (strcmp(networkType, "wan") == 0)
    {
        // WAN - higher latency, more reordering
        bufferCount = 15;
    }
}
```

```

        printf("WAN network: using %d out-of-order buffers\r\n", bufferCount);
    }
    else if (strcmp(networkType, "wireless") == 0)
    {
        // Wireless - potentially high packet reordering
        bufferCount = 25;
        printf("Wireless network: using %d out-of-order buffers\r\n", bufferCount);
    }
    else if (strcmp(networkType, "satellite") == 0)
    {
        // Satellite - very high latency, significant reordering
        bufferCount = 43; // Maximum
        printf("Satellite network: using %d out-of-order buffers (max)\r\n",
            bufferCount);
    }
    else
    {
        bufferCount = 10; // Conservative default
        printf("Unknown network type: using %d out-of-order buffers\r\n",
            bufferCount);
    }

    uint8_t result = SetOutOfOrderBuffers(fd, bufferCount);

    if (result != 0)
    {
        printf("WARNING: Failed to set out-of-order buffers: %d\r\n", result);
    }
}

void AdaptiveNetworkTransfer(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Configure based on detected network type
    ConfigureBuffersForNetwork(fd, "wireless");

    // Proceed with data transfer
    const char *message = "Data transfer request\r\n";
    write(fd, message, strlen(message));

    close(fd);
}

```

Memory-Constrained Configuration

```

#include <tcp.h>
#include <nbrtos.h>

void LowMemoryConnection(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Reduce buffers to minimum for memory-constrained environment
    uint8_t result = SetOutOfOrderBuffers(fd, 2);

    if (result == 0)
    {
        printf("Out-of-order buffers reduced to 2 (~3KB)\r\n");
        printf("Memory saved, but throughput may be reduced\r\n");
    }
    else
    {
        printf("Failed to set buffers: %d\r\n", result);
    }

    // Use connection with reduced buffers
    char buffer[256];
    int bytesRead = read(fd, buffer, sizeof(buffer) - 1, TICKS_PER_SECOND * 10);
}

```

```

    if (bytesRead > 0)
    {
        buffer[bytesRead] = '\0';
        printf("Received: %s\r\n", buffer);
    }

    close(fd);
}

```

Performance Testing with Different Buffer Sizes

```

#include <tcp.h>
#include <nbrtos.h>

struct PerformanceResult
{
    uint8_t bufferCount;
    uint32_t bytesReceived;
    uint32_t durationSeconds;
    uint32_t throughputKBps;
    int retransmissions;
};

PerformanceResult TestBufferConfiguration(uint8_t bufferCount)
{
    PerformanceResult result = {0};
    result.bufferCount = bufferCount;

    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed for buffer test %d: %d\r\n",
            bufferCount, fd);
        return result;
    }

    // Configure buffers
    if (SetOutOfOrderBuffers(fd, bufferCount) != 0)
    {
        printf("Failed to set buffers to %d\r\n", bufferCount);
        close(fd);
        return result;
    }

    printf("Testing with %d out-of-order buffers...\r\n", bufferCount);

    // Request data transfer
    const char *request = "SEND_TEST_DATA\r\n";
    write(fd, request, strlen(request));

    // Receive data
    char buffer[4096];
    uint32_t startTime = Secs;
    int initialRtx = GetTcpRtxCount(fd);

    while ((Secs - startTime) < 30) // 30 second test
    {
        int bytesRead = read(fd, buffer, sizeof(buffer), TICKS_PER_SECOND * 5);

        if (bytesRead <= 0)
        {
            break;
        }

        result.bytesReceived += bytesRead;
    }

    result.durationSeconds = Secs - startTime;
    result.retransmissions = GetTcpRtxCount(fd) - initialRtx;

    if (result.durationSeconds > 0)
    {
        result.throughputKBps = (result.bytesReceived / 1024) / result.durationSeconds;
    }

    close(fd);

    return result;
}

void OptimalBufferFinderTask(void *pd)
{

```

```

printf("=== Out-of-Order Buffer Performance Test ===\r\n\r\n");

uint8_t testSizes[] = {2, 5, 10, 20, 30, 43};
PerformanceResult results[6];

for (int i = 0; i < 6; i++)
{
    results[i] = TestBufferConfiguration(testSizes[i]);

    printf("Buffer size %d results:\r\n", results[i].bufferCount);
    printf("  Bytes:      %lu\r\n", results[i].bytesReceived);
    printf("  Duration:    %lu sec\r\n", results[i].durationSeconds);
    printf("  Throughput:   %lu KB/s\r\n", results[i].throughputKBps);
    printf("  Retransmits:  %d\r\n\r\n", results[i].retransmissions);

    // Wait between tests
    OSTimeDly(TICKS_PER_SECOND * 5);
}

// Find optimal configuration
uint32_t bestThroughput = 0;
int bestIndex = 0;

for (int i = 0; i < 6; i++)
{
    if (results[i].throughputKBps > bestThroughput)
    {
        bestThroughput = results[i].throughputKBps;
        bestIndex = i;
    }
}

printf("=== Optimal Configuration ===\r\n");
printf("Buffer count: %d\r\n", results[bestIndex].bufferCount);
printf("Throughput:    %lu KB/s\r\n", results[bestIndex].throughputKBps);
printf("Retransmits:   %d\r\n", results[bestIndex].retransmissions);
}

```

Server Configuration for Different Client Types

```

#include <tcp.h>
#include <nbrtos.h>

void ConfigureClientSocket(int clientFd, const IPADDR &clientAddr)
{
    // Determine client type based on IP range or other criteria
    uint8_t bufferCount = 5; // Default

    // Example: Different config for different subnets
    if ((clientAddr.ip4_addr & 0xFF) == 10) // 10.x.x.x subnet
    {
        // Local network client
        bufferCount = 5;
        printf("Local client %I: using default buffers (%d)\r\n",
            clientAddr, bufferCount);
    }
    else if ((clientAddr.ip4_addr & 0xFF) == 192) // 192.x.x.x
    {
        // Remote client via VPN or similar
        bufferCount = 20;
        printf("Remote client %I: increased buffers (%d)\r\n",
            clientAddr, bufferCount);
    }
    else
    {
        // Internet client - maximize buffers
        bufferCount = 43;
        printf("Internet client %I: maximum buffers (%d)\r\n",
            clientAddr, bufferCount);
    }

    uint8_t result = SetOutOfOrderBuffers(clientFd, bufferCount);

    if (result != 0)
    {
        printf("WARNING: Failed to configure buffers for %I: %d\r\n",
            clientAddr, result);
    }
}

void SmartServerTask(void *pd)
{
    const uint16_t SERVER_PORT = 8080;

    int listenFd = listen(INADDR_ANY, SERVER_PORT, 5);

    if (listenFd < 0)

```



```

{
    printf("Failed to create listening socket: %d\r\n", listenFd);
    return;
}

printf("Smart server listening on port %d\r\n", SERVER_PORT);

while (1)
{
    IPADDR clientAddr;
    uint16_t clientPort;

    int clientFd = accept(listenFd, &clientAddr, &clientPort, 0);

    if (clientFd > 0)
    {
        printf("Client connected: %I:%d\r\n", clientAddr, clientPort);

        // Configure socket based on client characteristics
        ConfigureClientSocket(clientFd, clientAddr);

        // Handle client...
        const char *welcome = "Connection configured\r\n";
        write(clientFd, welcome, strlen(welcome));

        close(clientFd);
    }
}

close(listenFd);
}

```

14.53.14.30 SetSocketRxBuffers()

```

int SetSocketRxBuffers (
    int fd,
    int n)

```

#include <tcp.h>

Configure the number of TCP receive buffers.

Sets the maximum number of receive buffers allocated to the socket. Each buffer is approximately 1548 bytes. The available space in these buffers determines the TCP window size advertised to the remote host, which controls how much data they can send.

As data is read from the socket, buffer space is freed, allowing the remote host to send more data. Since the maximum TCP window size is 65536 bytes, increasing this value beyond 43 buffers (approximately 66KB) provides no additional benefit.

The system default is 5 buffers (defined in [constants.h](#)) to conserve network buffer resources. This default is sufficient for most applications unless transferring large amounts of bulk data.

Parameters

<i>fd</i>	Socket file descriptor
<i>n</i>	Number of receive buffers to allocate

Returns

Status code: [TCP Socket Status](#)

See also

[setsockopt\(\)](#), [getsockopt\(\)](#), [clrsockopt\(\)](#), [SetSocketUnackBuffers\(\)](#), [SetSocketTxBuffers\(\)](#)

Expand for Example Usage

Examples

Basic Usage - Setting Receive Buffers

```

int fd = socket(AF_INET, SOCK_STREAM, 0);
if (fd > 0) {
    // Allocate 10 receive buffers (~15KB window)
    int result = SetSocketRxBuffers(fd, 10);
    if (result == TCP_ERR_NONE) {
        printf("Receive buffers configured successfully\n");
    }
}

```

Maximum Window Size Configuration

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
if (fd > 0) {
    // Set to maximum useful value (43 buffers equals approx. 66KB)
    // Beyond this provides no benefit due to 64KB TCP window limit
    SetSocketRxBuffers(fd, 43);

    bind(fd, ...);
    listen(fd, 5);
}
```

Server Socket with Custom Receive Buffer

```
int listenFd = socket(AF_INET, SOCK_STREAM, 0);
SetSocketRxBuffers(listenFd, 20); // Configure before accepting connections

bind(listenFd, (struct sockaddr*)&addr, sizeof(addr));
listen(listenFd, 5);

int clientFd = accept(listenFd, NULL, NULL);
// clientFd inherits the buffer settings from listenFd
```

Memory-Constrained Application

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
if (fd > 0) {
    // Use minimal buffers to conserve memory
    SetSocketRxBuffers(fd, 3); // ~4.6KB receive window
}
```

14.53.14.31 SetSocketTxBuffers()

```
int SetSocketTxBuffers (
    int fd,
    int n)
#include <tcp.h>
```

Configure the number of TCP transmit buffers.

Sets the maximum number of buffers allocated to the transmit queue. Each buffer is approximately 1548 bytes. The transmit queue accumulates outgoing data up to this limit regardless of the current state of unacknowledged buffers or the remote host's window size. This value effectively sets the maximum size for a single [write\(\)](#) operation.

Note

Always check the return value of write functions to determine how many bytes were actually written to the transmit queue.

Parameters

<i>fd</i>	Socket file descriptor
<i>n</i>	Number of transmit buffers to allocate

Returns

Status code: [TCP Socket Status](#)

See also

[setsockopt\(\)](#), [getsockopt\(\)](#), [clrsockopt\(\)](#), [SetSocketRxBuffers\(\)](#), [SetSocketUnackBuffers\(\)](#), [write\(\)](#), [writeall\(\)](#)

Expand for Example Usage

Examples

Basic Usage - Setting Transmit Buffers

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
if (fd > 0) {
    // Allocate 15 transmit buffers (~23KB queue)
    int result = SetSocketTxBuffers(fd, 15);
    if (result == TCP_ERR_NONE) {
        printf("Transmit buffers configured successfully\n");
    }
}
```

Large Data Transfer Setup

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
if (fd > 0) {
    // Configure for sending large chunks of data
    SetSocketTxBuffers(fd, 30); // ~46KB transmit queue

    connect(fd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));

    // Now we can write larger amounts in a single call
    char largeBuffer[40000];
    int bytesWritten = write(fd, largeBuffer, sizeof(largeBuffer));
    printf("Wrote %d bytes\n", bytesWritten);
}
```

Checking Write Return Values

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
SetSocketTxBuffers(fd, 10); // ~15KB transmit queue
connect(fd, ...);

char data[20000];
int bytesWritten = write(fd, data, sizeof(data));

if (bytesWritten < sizeof(data)) {
    printf("Warning: Only wrote %d of %d bytes\n",
           bytesWritten, sizeof(data));
    // Transmit queue is full, need to wait or use writeall()
}
```

Complete Socket Configuration

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
if (fd > 0) {
    // Configure all buffer settings for optimal performance
    SetSocketTxBuffers(fd, 20); // Transmit queue: ~31KB
    SetSocketRxBuffers(fd, 15); // Receive window: ~23KB
    SetSocketUnackBuffers(fd, 10); // Unacknowledged: 10 buffers

    // Socket is now ready for high-performance communication
    connect(fd, (struct sockaddr*)&addr, sizeof(addr));
}
```

14.53.14.32 SetSocketUnackBuffers()

```
int SetSocketUnackBuffers (
    int fd,
    uint8_t val) [inline]
#include <tcp.h>
```

Configure the maximum number of unacknowledged transmit buffers.

Sets the maximum number of TCP transmit buffers that can remain unacknowledged by the remote host. When data is sent, it moves from the transmit queue to the unacknowledged list. Buffers remain in this list until the remote host sends an acknowledgment (ACK), at which point they are freed. The actual number of unacknowledged buffers is also limited by the remote host's TCP window size.

Increasing this value can improve bulk data throughput on fast networks. However, on lossy networks without selective acknowledgment (SACK), higher values may reduce performance. Note: Most modern PCs support SACK, but many NAT routers do not.

Parameters

<i>fd</i>	Socket file descriptor
<i>val</i>	Maximum number of unacknowledged buffers

Returns

Status code: [TCP Socket Status](#)

See also

[setsockopt\(\)](#), [getsockopt\(\)](#), [clrsockopt\(\)](#), [SetSocketRxBuffers\(\)](#), [SetSocketTxBuffers\(\)](#)

Expand for Example Usage**Examples****Basic Usage - Setting Unacknowledged Buffers**

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
if (fd > 0) {
    // Set to 10 unacknowledged buffers for better throughput
    int result = SetSocketUnackBuffers(fd, 10);
    if (result == TCP_ERR_NONE) {
        printf("Successfully configured unack buffers\n");
    }
}
```

Optimizing for High-Speed Bulk Transfer

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
if (fd > 0) {
    // Configure socket for high-throughput data transfer
    SetSocketTxBuffers(fd, 20);           // Large transmit queue
    SetSocketUnackBuffers(fd, 15);       // Allow more unacknowledged data

    // Now ready for bulk transfer
    connect(fd, ...);
    write(fd, largeDataBuffer, largeDataSize);
}
```

Conservative Setting for Lossy Networks

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
if (fd > 0) {
    // Lower value for networks with packet loss
    SetSocketUnackBuffers(fd, 3); // Conservative for lossy/NAT environments
}
```

14.53.14.33 setsockopt()

```
int setsockopt (
    int fd,
    int option)
```

```
#include <tcp.h>
```

Enable TCP socket options.

Sets one or more option flags for the specified socket. Socket options control various aspects of TCP behavior such as the Nagle algorithm and PUSH flag handling. Multiple options can be combined using bitwise OR.

Parameters

<i>fd</i>	Socket file descriptor
<i>option</i>	Socket option flags to enable: TCP Socket Options

Returns

Bitmask of all currently enabled options for the socket, or negative value on error

Notes:

- Common options include:
 - SO_NODELAY: Disable Nagle algorithm for low-latency transmission
 - SO_PUSH: Set TCP PUSH flag on transmitted data
 - SO_KEEPALIVE: Enable TCP keepalive probes

- Multiple options can be set simultaneously using bitwise OR: `setsockopt(fd, SO_NODELAY | SO_PUSH)` ;
- This function does not clear existing options - it only adds new ones. Use `clrsockopt()` to disable options.

See also

[clrsockopt\(\)](#), [getsockopt\(\)](#), [SetSocketUnackBuffers\(\)](#), [SetSocketRxBuffers\(\)](#), [SetSocketTxBuffers\(\)](#)

Expand for Example Usage

Examples

Disable Nagle Algorithm for Low Latency

```
#include <tcp.h>
#include <nbrtos.h>

void LowLatencyClient(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Disable Nagle algorithm for immediate transmission
    int result = setsockopt(fd, SO_NODELAY);

    if (result >= 0)
    {
        printf("SO_NODELAY enabled - options mask: 0x%X\r\n", result);
    }
    else
    {
        printf("Failed to set SO_NODELAY: %d\r\n", result);
    }

    // Send small messages that will be transmitted immediately
    const char *msg1 = "A";
    const char *msg2 = "B";
    const char *msg3 = "C";

    write(fd, msg1, 1); // Sent immediately (no Nagle delay)
    write(fd, msg2, 1); // Sent immediately
    write(fd, msg3, 1); // Sent immediately

    printf("Small messages sent without Nagle delay\r\n");

    close(fd);
}
```

Enable Multiple Options

```
#include <tcp.h>
#include <nbrtos.h>

void ConfigureOptimalSocket(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Enable multiple options for optimal real-time performance
    int options = SO_NODELAY | SO_PUSH;
    int result = setsockopt(fd, options);

    if (result >= 0)
    {
        printf("Socket configured for low-latency:\r\n");
        printf("  SO_NODELAY: %s\r\n", (result & SO_NODELAY) ? "ON" : "OFF");
        printf("  SO_PUSH:    %s\r\n", (result & SO_PUSH) ? "ON" : "OFF");
    }
}
```

```

    else
    {
        printf("Failed to configure socket: %d\r\n", result);
    }

    // Send time-critical data
    const char *message = "Real-time control command\r\n";
    write(fd, message, strlen(message));

    close(fd);
}

```

Interactive Terminal Application

```

#include <tcp.h>
#include <nbrtos.h>

void InteractiveTerminal(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 23; // Telnet port

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    printf("Connected to telnet server\r\n");

    // Configure for interactive use - disable Nagle for keystroke echo
    int result = setsockopt(fd, SO_NODELAY);

    if (result < 0)
    {
        printf("Warning: Failed to set SO_NODELAY\r\n");
    }
    else
    {
        printf("SO_NODELAY enabled - keystrokes will be sent immediately\r\n");
    }

    // Simulate sending individual keystrokes
    const char *keystrokes = "ls -la\r\n";

    for (int i = 0; keystrokes[i] != '\0'; i++)
    {
        // Each keystroke sent immediately (no buffering delay)
        write(fd, &keystrokes[i], 1);
        OSTimeDly(TICKS_PER_SECOND / 10); // Simulate typing speed
    }

    // Read response
    char buffer[1024];
    int bytesRead = read(fd, buffer, sizeof(buffer) - 1, TICKS_PER_SECOND * 5);

    if (bytesRead > 0)
    {
        buffer[bytesRead] = '\0';
        printf("Server response:\r\n%s\r\n", buffer);
    }

    close(fd);
}

```

Compare Performance With and Without Nagle

```

#include <tcp.h>
#include <nbrtos.h>

void TestNagleImpact(bool disableNagle)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    if (disableNagle)

```

```

{
    setsockopt(fd, SO_NODELAY);
    printf("Testing with SO_NODELAY (Nagle disabled)\r\n");
}
else
{
    printf("Testing with Nagle algorithm enabled (default)\r\n");
}

// Send 100 small messages
uint32_t startTime = TimeTick;

for (int i = 0; i < 100; i++)
{
    char msg[16];
    snprintf(msg, sizeof(msg), "MSG%d\r\n", i);
    write(fd, msg, strlen(msg));
}

uint32_t elapsedTicks = TimeTick - startTime;
uint32_t elapsedMs = (elapsedTicks * 1000) / TICKS_PER_SECOND;

printf("100 messages sent in %lu ms\r\n", elapsedMs);

int rtxCount = GetTcpRtxCount(fd);
printf("Retransmissions: %d\r\n", rtxCount);

close(fd);
}

void NagleComparisonTask(void *pd)
{
    printf("=== Nagle Algorithm Performance Comparison ===\r\n\r\n");

    printf("Test 1: With Nagle (default)\r\n");
    TestNagleImpact(false);

    OSTimeDly(TICKS_PER_SECOND * 2);

    printf("\r\nTest 2: Without Nagle (SO_NODELAY)\r\n");
    TestNagleImpact(true);

    printf("\r\nConclusion:\r\n");
    printf(" - Nagle enabled: Better bandwidth efficiency, higher latency\r\n");
    printf(" - Nagle disabled: Lower latency, more packets sent\r\n");
}

```

Server-Side Socket Configuration

```

#include <tcp.h>
#include <nbrtos.h>

void ConfiguredServerTask(void *pd)
{
    const uint16_t SERVER_PORT = 8080;

    int listenFd = listen(INADDR_ANY, SERVER_PORT, 5);

    if (listenFd < 0)
    {
        printf("Failed to create listening socket: %d\r\n", listenFd);
        return;
    }

    printf("Server listening on port %d\r\n", SERVER_PORT);

    while (1)
    {
        IPADDR clientAddr;
        uint16_t clientPort;

        int clientFd = accept(listenFd, &clientAddr, &clientPort, 0);

        if (clientFd > 0)
        {
            printf("Client connected: %I:%d\r\n", clientAddr, clientPort);

            // Configure client socket for optimal performance
            int options = SO_NODELAY | SO_PUSH;
            int result = setsockopt(clientFd, options);

            if (result >= 0)
            {
                printf("Client socket configured: 0x%X\r\n", result);
            }

            // Handle client with optimized socket

```

```

char buffer[256];
int bytesRead = read(clientFd, buffer, sizeof(buffer) - 1,
                    Ticks_Per_Second * 5);

if (bytesRead > 0)
{
    buffer[bytesRead] = '\0';

    // Echo response will be sent immediately due to SO_NODELAY
    write(clientFd, buffer, bytesRead);
}

close(clientFd);
}

close(listenFd);
}

```

14.53.14.34 SocketPeek()

```

char SocketPeek (
    int fd)
#include <tcp.h>

```

Peek at the next available byte without removing it from the receive buffer.

Examines the next byte that would be read from the socket without actually removing it from the receive buffer. The byte remains available for subsequent read operations. If no data is available, returns 0.

Parameters

<i>fd</i>	Socket file descriptor

Returns

The next byte that would be read, or 0 if no data is available

See also

[read\(\)](#), [dataavail\(\)](#)

Expand for Example Usage

Examples

Basic Usage - Peeking at Next Byte

```

int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

char nextByte = SocketPeek(fd);
if (nextByte != 0) {
    printf("Next byte is: 0x%02X ('%c')\n", nextByte,
        isprint(nextByte) ? nextByte : '.');

    // Byte is still in buffer, can read it normally
    char actualByte;
    read(fd, &actualByte, 1);
    // actualByte == nextByte
}

```

Protocol Detection - Checking Message Type

```

int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

// Peek at message type without consuming it
char msgType = SocketPeek(fd);

if (msgType == 0x01) {
    HandleDataMessage(fd);
} else if (msgType == 0x02) {
    HandleControlMessage(fd);
} else if (msgType == 0x03) {
    HandleErrorMessage(fd);
} else if (msgType == 0) {
    printf("No data available\n");
} else {
    printf("Unknown message type: 0x%02X\n", msgType);
    abortsocket(fd);
}

```


Waiting for Specific Start Byte

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

const char START_MARKER = 0xAA;

// Wait for start marker
while (1) {
    char nextByte = SocketPeek(fd);

    if (nextByte == START_MARKER) {
        // Found start marker, now read the full message
        char buffer[256];
        int n = read(fd, buffer, sizeof(buffer));
        ProcessMessage(buffer, n);
        break;
    } else if (nextByte != 0) {
        // Discard unexpected byte
        char discard;
        read(fd, &discard, 1);
        printf("Discarded unexpected byte: 0x%02X\n", discard);
    } else {
        // No data yet, wait
        OSTimeDly(TICKS_PER_SECOND / 10);
    }
}
```

Non-Destructive Buffer Inspection

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

// Check if data looks like ASCII text or binary
char firstByte = SocketPeek(fd);

if (firstByte == 0) {
    printf("No data available\n");
} else if (isprint(firstByte) || firstByte == '\n' || firstByte == '\r') {
    printf("Data appears to be ASCII text\n");
    // Process as text
    char textBuffer[512];
    int n = read(fd, textBuffer, sizeof(textBuffer) - 1);
    textBuffer[n] = '\0';
    printf("Text: %s\n", textBuffer);
} else {
    printf("Data appears to be binary (first byte: 0x%02X)\n", firstByte);
    // Process as binary
    uint8_t binaryBuffer[512];
    int n = read(fd, binaryBuffer, sizeof(binaryBuffer));
    ProcessBinaryData(binaryBuffer, n);
}
```

Checking Data Availability Before Blocking Read

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

// Use peek to check if data is available before potentially blocking
if (SocketPeek(fd) != 0) {
    char buffer[256];
    int n = read(fd, buffer, sizeof(buffer));
    printf("Read %d bytes\n", n);
} else {
    printf("No data available, skipping read\n");
}
```

14.53.14.35 SockReadWithTimeout()

```
int SockReadWithTimeout (
    int fd,
    char * buf,
    int nbytes,
    uint32_t timeout)
```

#include <tcp.h>

Read from a TCP socket with a timeout.

Attempts to read data from a TCP socket, waiting up to the specified timeout period. The function returns when at least 1 byte of data becomes available or the timeout expires. The nbytes parameter specifies the maximum number

of bytes to read, but the actual number read may be less. Always check the return value to determine how many bytes were actually read.

Note

This function works with standard TCP sockets only; it does not work with TLS sockets.

Parameters

<i>fd</i>	Socket file descriptor
<i>buf</i>	Buffer to store received data
<i>nbytes</i>	Maximum number of bytes to read
<i>timeout</i>	Timeout duration in system ticks. Recommended usage: TICKS_PER_SECOND * n where n is the number of seconds. For example: TICKS_PER_SECOND * 5 for a 5 second timeout

Returns

Number of bytes read on success, negative [TCP Socket Status](#) error code on failure

See also

[ReadWithTimeout\(\)](#)

Expand for Example Usage

Examples

Basic Usage - Read with 5 Second Timeout

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

char buffer[256];
int bytesRead = SockReadWithTimeout(fd, buffer, sizeof(buffer), TICKS_PER_SECOND * 5);

if (bytesRead > 0) {
    printf("Received %d bytes: %.s\n", bytesRead, bytesRead, buffer);
} else if (bytesRead == TCP_ERR_TIMEOUT) {
    printf("Read timeout after 5 seconds\n");
} else {
    printf("Read error: %d\n", bytesRead);
}
```

Request-Response Pattern with Timeout

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

// Send request
const char *request = "GET /data HTTP/1.1\r\n\r\n";
write(fd, request, strlen(request));

// Wait up to 10 seconds for response
char response[1024];
int n = SockReadWithTimeout(fd, response, sizeof(response), TICKS_PER_SECOND * 10);

if (n > 0) {
    response[n] = '\0';
    printf("Server response: %s\n", response);
} else if (n == TCP_ERR_TIMEOUT) {
    printf("Server did not respond within 10 seconds\n");
    abortsocket(fd);
}
```

Multiple Read Attempts with Variable Timeout

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

char buffer[512];
int totalBytes = 0;
int attempts = 0;
const int maxAttempts = 3;

while (attempts < maxAttempts) {
    int n = SockReadWithTimeout(fd, buffer + totalBytes,
```

```

        sizeof(buffer) - totalBytes,
        TICKS_PER_SECOND * 2);

    if (n > 0) {
        totalBytes += n;
        printf("Attempt %d: read %d bytes (total: %d)\n",
            attempts + 1, n, totalBytes);
        break; // Success
    } else if (n == TCP_ERR_TIMEOUT) {
        attempts++;
        printf("Timeout on attempt %d\n", attempts);
    } else {
        printf("Read error: %d\n", n);
        break;
    }
}

if (attempts >= maxAttempts) {
    printf("Failed after %d attempts\n", maxAttempts);
}

```

Polling Loop with Short Timeout

```

int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

bool running = true;
char data[128];

while (running) {
    // Short timeout allows checking other conditions frequently
    int n = SocketReadWithTimeout(fd, data, sizeof(data), TICKS_PER_SECOND / 2);

    if (n > 0) {
        ProcessData(data, n);
    } else if (n == TCP_ERR_TIMEOUT) {
        // No data yet, check other conditions
        if (ShouldShutdown()) {
            running = false;
        }
    } else {
        printf("Socket error, exiting\n");
        running = false;
    }
}

```

14.53.14.36 TcpAllDataAcked()

```

BOOL TcpAllDataAcked (
    int socket)

```

```

#include <tcp.h>

```

Check if all transmitted data has been acknowledged.

Determines whether all data sent on this socket has been acknowledged by the remote host. This is useful for ensuring data has been successfully delivered before closing a connection or performing other operations.

Parameters

<i>socket</i>	Socket file descriptor

Return values

<i>TRUE</i>	All sent data has been acknowledged by the remote host
<i>FALSE</i>	Some data remains unacknowledged

See also

[WaitForSocketFlush\(\)](#)

14.53.14.37 TcpGetLastRxInterval()

```
uint32_t TcpGetLastRxInterval (
    int fd)
```

```
#include <tcp.h>
```

Get the time elapsed since the last received packet.

Calculates the number of system ticks that have elapsed between the current time and when the last packet was received on this socket. This is a convenience function that computes the difference between the current system time (TimeTick) and the socket's lastRxTime value.

Parameters

<i>fd</i>	Socket file descriptor

Returns

Number of system ticks since the last packet was received, or 0 if socket is invalid

Notes:

- This function handles one rollover of the system tick counter (approximately 6 years of uptime at typical tick rates) but not multiple rollovers. Given typical usage patterns and connection lifetimes, this limitation should not present problems in practice.
- The returned value is in system ticks. Use TICKS_PER_SECOND to convert to seconds: `uint32_t seconds = TcpGetLastRxInterval(fd) / TICKS_PER_SECOND;`
- This function is useful for detecting idle connections or implementing timeout logic without needing to manually track timestamps.

See also

[TcpGetLastRxTime\(\)](#), [TimeTick](#), [TICKS_PER_SECOND](#)

Expand for Example Usage**Examples****Basic Idle Time Detection**

```
#include <tcp.h>
#include <nbrtos.h>

#define IDLE_TIMEOUT_SEC 60

void CheckIdleConnection(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    printf("Connected - monitoring idle time\r\n");

    while (1)
    {
        // Check idle time every 5 seconds
        OSTimeDly(TICKS_PER_SECOND * 5);

        uint32_t idleTicks = TcpGetLastRxInterval(fd);
        uint32_t idleSeconds = idleTicks / TICKS_PER_SECOND;

        printf("Connection idle for %lu seconds\r\n", idleSeconds);

        if (idleSeconds > IDLE_TIMEOUT_SEC)
        {
            printf("Idle timeout - closing connection\r\n");
```

```

        break;
    }

    // Check connection state
    TcpState state = TcpGetSocketState(fd);
    if (state != TCP_STATE_ESTABLISHED)
    {
        printf("Connection closed (state: %d)\r\n", state);
        break;
    }
}

close(fd);
}

```

Server with Client Idle Monitoring

```

#include <tcp.h>
#include <nbrtos.h>

#define MAX_IDLE_TIME_SEC 120

void HandleClientWithTimeout(int clientFd, const IPADDR &clientAddr)
{
    printf("Client %I connected\r\n", clientAddr);

    while (1)
    {
        // Check for incoming data with timeout
        char buffer[256];
        int bytesRead = read(clientFd, buffer, sizeof(buffer) - 1,
                             TICKS_PER_SECOND * 5);

        if (bytesRead > 0)
        {
            buffer[bytesRead] = '\0';
            printf("Client %I: %s\r\n", clientAddr, buffer);

            // Echo back
            write(clientFd, buffer, bytesRead);
        }

        // Check idle time
        uint32_t idleTicks = TcpGetLastRxInterval(clientFd);
        uint32_t idleSeconds = idleTicks / TICKS_PER_SECOND;

        if (idleSeconds > MAX_IDLE_TIME_SEC)
        {
            printf("Client %I idle timeout (%lu seconds)\r\n",
                   clientAddr, idleSeconds);

            const char *msg = "Idle timeout - disconnecting\r\n";
            write(clientFd, msg, strlen(msg));
            break;
        }

        // Check connection state
        TcpState state = TcpGetSocketState(clientFd);
        if (state != TCP_STATE_ESTABLISHED)
        {
            printf("Client %I disconnected (state: %d)\r\n",
                   clientAddr, state);
            break;
        }
    }

    close(clientFd);
    printf("Client %I connection closed\r\n", clientAddr);
}

void IdleMonitorServerTask(void *pd)
{
    const uint16_t SERVER_PORT = 8080;

    int listenFd = listen(INADDR_ANY, SERVER_PORT, 5);

    if (listenFd < 0)
    {
        printf("Failed to create listening socket: %d\r\n", listenFd);
        return;
    }

    printf("Idle monitor server listening on port %d\r\n", SERVER_PORT);
    printf("Client idle timeout: %d seconds\r\n", MAX_IDLE_TIME_SEC);

    while (1)
    {

```

```

    IPADDR clientAddr;
    uint16_t clientPort;

    int clientFd = accept(listenFd, &clientAddr, &clientPort, 0);

    if (clientFd > 0)
    {
        HandleClientWithTimeout(clientFd, clientAddr);
    }
}

close(listenFd);
}

```

Progressive Idle Warnings

```

#include <tcp.h>
#include <nbrtos.h>

void ConnectionWithWarnings(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    printf("Connected - idle warning system active\r\n");

    bool warning30 = false;
    bool warning60 = false;
    bool warning90 = false;

    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND * 10);

        uint32_t idleSeconds = TcpGetLastRxInterval(fd) / TICKS_PER_SECOND;

        if (idleSeconds >= 90 && !warning90)
        {
            printf("WARNING: Connection idle for 90+ seconds!\r\n");
            warning90 = true;

            // Final warning - send keepalive
            TcpSendKeepAlive(fd);
        }
        else if (idleSeconds >= 60 && !warning60)
        {
            printf("WARNING: Connection idle for 60+ seconds\r\n");
            warning60 = true;
        }
        else if (idleSeconds >= 30 && !warning30)
        {
            printf("INFO: Connection idle for 30+ seconds\r\n");
            warning30 = true;
        }
        else if (idleSeconds < 30)
        {
            // Reset warnings if activity resumes
            if (warning30 || warning60 || warning90)
            {
                printf("Activity resumed - warnings cleared\r\n");
                warning30 = warning60 = warning90 = false;
            }
        }

        if (idleSeconds >= 120)
        {
            printf("TIMEOUT: Connection idle for 120+ seconds - disconnecting\r\n");
            break;
        }

        // Check connection validity
        TcpState state = TcpGetSocketState(fd);
        if (state != TCP_STATE_ESTABLISHED)
        {
            printf("Connection closed (state: %d)\r\n", state);
            break;
        }
    }
}

```

```

        close(fd);
    }

```

Adaptive Timeout Based on Activity

```

#include <tcp.h>
#include <nbrtos.h>

void AdaptiveTimeoutConnection(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    uint32_t baseTimeoutSec = 60;
    uint32_t currentTimeoutSec = baseTimeoutSec;
    uint32_t maxTimeoutSec = 300; // 5 minutes max

    printf("Connected - adaptive timeout starting at %lu seconds\r\n",
        currentTimeoutSec);

    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND * 10);

        uint32_t idleSeconds = TcpGetLastRxInterval(fd) / TICKS_PER_SECOND;

        if (idleSeconds > currentTimeoutSec)
        {
            printf("Idle timeout (%lu > %lu seconds) - checking connection\r\n",
                idleSeconds, currentTimeoutSec);

            uint32_t rxBefore = TcpGetLastRxTime(fd);
            TcpSendKeepAlive(fd);
            OSTimeDly(TICKS_PER_SECOND * 3);
            uint32_t rxAfter = TcpGetLastRxTime(fd);

            if (rxAfter != rxBefore)
            {
                // Connection alive but idle - increase timeout
                currentTimeoutSec = currentTimeoutSec * 2;
                if (currentTimeoutSec > maxTimeoutSec)
                {
                    currentTimeoutSec = maxTimeoutSec;
                }

                printf("Connection alive - timeout increased to %lu seconds\r\n",
                    currentTimeoutSec);
            }
            else
            {
                printf("Connection dead - no response\r\n");
                break;
            }
        }
        else if (idleSeconds < (baseTimeoutSec / 2))
        {
            // Recent activity - reset timeout
            if (currentTimeoutSec != baseTimeoutSec)
            {
                currentTimeoutSec = baseTimeoutSec;
                printf("Activity detected - timeout reset to %lu seconds\r\n",
                    currentTimeoutSec);
            }
        }

        // Check connection state
        TcpState = TcpGetSocketState(fd);
        if (state != TCP_STATE_ESTABLISHED)
        {
            printf("Connection closed (state: %d)\r\n", state);
            break;
        }
    }

    close(fd);
}

```

14.53.14.38 TcpGetLastRxTime()

```
uint32_t TcpGetLastRxTime (
    int fd)
```

```
#include <tcp.h>
```

Get the system tick count when the last packet was received.

Returns the timestamp (in system ticks) of the last received packet on this connection. Each TCP socket maintains a lastRxTime value that is updated whenever a packet arrives. This function is primarily used with [TcpSendKeepAlive\(\)](#) to implement TCP keepalive functionality and detect stale connections.

To detect a lost connection:

1. Record lastRxTime before sending a keepalive packet
2. Send the keepalive packet with [TcpSendKeepAlive\(\)](#)
3. Wait sufficient time for a response
4. Check if lastRxTime has changed
5. If unchanged, the remote host has not responded and the connection may be lost

Parameters

<i>fd</i>	Socket file descriptor

Returns

System tick count when the last packet was received, or 0 if socket is invalid

Notes:

- The returned value is in system ticks. Use TICKS_PER_SECOND to convert to seconds.
- This timestamp is updated on any received packet, including ACKs, data packets, and keepalive responses.

Warning

Do not call this function more frequently than once per second to avoid unnecessary overhead.

See also

[TcpSendKeepAlive\(\)](#), [TcpGetLastRxInterval\(\)](#), [TimeTick](#), [TICKS_PER_SECOND](#)

Expand for Example Usage

Examples

Basic Keepalive Detection

```
#include <tcp.h>
#include <nbrtos.h>

bool IsConnectionAlive(int fd, uint32_t timeoutSeconds)
{
    // Get last RX time before sending keepalive
    uint32_t lastRxTimeBefore = TcpGetLastRxTime(fd);

    // Send keepalive packet
    TcpSendKeepAlive(fd);

    printf("Keepalive sent, waiting for response...\r\n");

    // Wait for response (convert timeout to ticks)
    OSTimeDly(timeoutSeconds * TICKS_PER_SECOND);

    // Check if we received a response
    uint32_t lastRxTimeAfter = TcpGetLastRxTime(fd);

    if (lastRxTimeAfter != lastRxTimeBefore)
    {
```



```

        printf("Connection alive - received response\r\n");
        return true;
    }
    else
    {
        printf("Connection dead - no response to keepalive\r\n");
        return false;
    }
}

void ConnectionMonitorTask(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    printf("Connected to %I:%d\r\n", serverIp, serverPort);

    // Monitor connection with keepalives every 30 seconds
    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND * 30);

        if (!IsConnectionAlive(fd, 5))
        {
            printf("Connection lost, reconnecting...\r\n");
            close(fd);

            fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);
            if (fd < 0)
            {
                printf("Reconnection failed: %d\r\n", fd);
                break;
            }

            printf("Reconnected successfully\r\n");
        }

        if (fd > 0) close(fd);
    }
}

```

Display Last Packet Receive Time

```

#include <tcp.h>
#include <nbrtos.h>

void DisplayLastPacketInfo(int fd)
{
    uint32_t lastRxTime = TcpGetLastRxTime(fd);
    uint32_t currentTime = TimeTick;

    // Calculate elapsed time in ticks
    uint32_t elapsedTicks = currentTime - lastRxTime;

    // Convert to seconds
    uint32_t elapsedSeconds = elapsedTicks / TICKS_PER_SECOND;

    printf("Last packet received:\r\n");
    printf("  Timestamp:    %lu ticks\r\n", lastRxTime);
    printf("  Current time:  %lu ticks\r\n", currentTime);
    printf("  Time since RX: %lu seconds\r\n", elapsedSeconds);
}

void PacketTimingExample(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    // Initial display
    DisplayLastPacketInfo(fd);
}

```

```

// Send some data
const char *message = "Test message\r\n";
write(fd, message, strlen(message));

// Read response
char buffer[256];
int bytesRead = read(fd, buffer, sizeof(buffer) - 1, TICKS_PER_SECOND * 5);

if (bytesRead > 0)
{
    buffer[bytesRead] = '\0';
    printf("Received: %s\r\n", buffer);

    // Display timing after receiving data
    DisplayLastPacketInfo(fd);
}

close(fd);
}

```

Advanced Keepalive with Retry Logic

```

#include <tcp.h>
#include <nbrtos.h>

#define MAX_KEEPALIVE_RETRIES 3
#define KEEPALIVE_INTERVAL_SEC 30
#define KEEPALIVE_TIMEOUT_SEC 5

typedef enum
{
    CONN_ALIVE,
    CONN_SUSPECT,
    CONN_DEAD
} ConnectionState;

ConnectionState CheckConnectionHealth(int fd)
{
    int failedProbes = 0;

    for (int i = 0; i < MAX_KEEPALIVE_RETRIES; i++)
    {
        uint32_t lastRxBefore = TcpGetLastRxTime(fd);

        TcpSendKeepAlive(fd);
        printf("Keepalive probe %d/%d sent\r\n", i + 1, MAX_KEEPALIVE_RETRIES);

        OSTimeDly(KEEPALIVE_TIMEOUT_SEC * TICKS_PER_SECOND);

        uint32_t lastRxAfter = TcpGetLastRxTime(fd);

        if (lastRxAfter != lastRxBefore)
        {
            printf("Response received on probe %d\r\n", i + 1);
            return CONN_ALIVE;
        }
        else
        {
            failedProbes++;
            printf("No response on probe %d\r\n", i + 1);
        }

        // Small delay between retries
        if (i < MAX_KEEPALIVE_RETRIES - 1)
        {
            OSTimeDly(TICKS_PER_SECOND);
        }
    }

    if (failedProbes >= MAX_KEEPALIVE_RETRIES)
    {
        return CONN_DEAD;
    }

    return CONN_SUSPECT;
}

void RobustConnectionTask(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
    }
}

```

```

        return;
    }

    printf("Connected - starting keepalive monitoring\r\n");

    while (1)
    {
        // Wait before next check
        OSTimeDly(KEEPALIVE_INTERVAL_SEC * TICKS_PER_SECOND);

        ConnectionState state = CheckConnectionHealth(fd);

        switch (state)
        {
            case CONN_ALIVE:
                printf("Connection healthy\r\n");
                break;

            case CONN_SUSPECT:
                printf("Connection suspect - some probes failed\r\n");
                break;

            case CONN_DEAD:
                printf("Connection dead - all probes failed\r\n");
                close(fd);

                // Attempt reconnection
                printf("Attempting reconnection...\r\n");
                OSTimeDly(TICKS_PER_SECOND * 5);

                fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);
                if (fd < 0)
                {
                    printf("Reconnection failed: %d\r\n", fd);
                    return;
                }

                printf("Reconnected successfully\r\n");
                break;
        }
    }

    if (fd > 0) close(fd);
}

```

Server-Side Connection Monitoring

```

#include <tcp.h>
#include <nbrtos.h>

#define CLIENT_TIMEOUT_SEC 60

void MonitorClientConnection(int clientFd, const IPADDR &clientAddr)
{
    uint32_t lastCheckTime = Secs;

    while (1)
    {
        // Check every 10 seconds
        OSTimeDly(TICKS_PER_SECOND * 10);

        // Get time since last packet
        uint32_t lastRxTime = TcpGetLastRxTime(clientFd);
        uint32_t currentTime = TimeTick;
        uint32_t idleTimeSeconds = (currentTime - lastRxTime) / TICKS_PER_SECOND;

        printf("Client %I idle for %lu seconds\r\n", clientAddr, idleTimeSeconds);

        if (idleTimeSeconds > CLIENT_TIMEOUT_SEC)
        {
            printf("Client %I timeout - checking if alive\r\n", clientAddr);

            uint32_t rxBefore = TcpGetLastRxTime(clientFd);
            TcpSendKeepAlive(clientFd);
            OSTimeDly(TICKS_PER_SECOND * 5);
            uint32_t rxAfter = TcpGetLastRxTime(clientFd);

            if (rxAfter == rxBefore)
            {
                printf("Client %I not responding - closing connection\r\n",
                    clientAddr);
                close(clientFd);
                return;
            }
            else
            {
                printf("Client %I responded to keepalive\r\n", clientAddr);
            }
        }
    }
}

```

```

    }
}

// Check if socket is still valid
TcpState state = TcpGetSocketState(clientFd);
if (state != TCP_STATE_ESTABLISHED)
{
    printf("Client %I connection closed (state: %d)\r\n",
           clientAddr, state);
    close(clientFd);
    return;
}
}

void TimeoutServerTask(void *pd)
{
    const uint16_t SERVER_PORT = 8080;

    int listenFd = listen(INADDR_ANY, SERVER_PORT, 5);

    if (listenFd < 0)
    {
        printf("Failed to create listening socket: %d\r\n", listenFd);
        return;
    }

    printf("Timeout server listening on port %d\r\n", SERVER_PORT);
    printf("Client timeout: %d seconds\r\n", CLIENT_TIMEOUT_SEC);

    while (1)
    {
        IPADDR clientAddr;
        uint16_t clientPort;

        int clientFd = accept(listenFd, &clientAddr, &clientPort, 0);

        if (clientFd > 0)
        {
            printf("Client connected: %I:%d\r\n", clientAddr, clientPort);

            // Monitor this client (in real app, spawn a task)
            MonitorClientConnection(clientFd, clientAddr);

            printf("Client %I monitoring ended\r\n", clientAddr);
        }

        close(listenFd);
    }
}

```

14.53.14.39 TcpGetRxBufferSpaceUsed()

```
uint16_t TcpGetRxBufferSpaceUsed (
    int fd)
```

```
#include <tcp.h>
```

Get the number of bytes currently in the receive buffer.

Returns the amount of received data waiting to be read from the socket's receive buffer.

Parameters

<i>fd</i>	Socket file descriptor

Returns

Number of bytes available to read

See also

[TcpGetTxBufferAvailSpace\(\)](#), [TcpGetTxDataWaiting\(\)](#)

14.53.14.40 TcpGetSocketInterface()

```
int TcpGetSocketInterface (
    int fd)
```

```
#include <tcp.h>
```

Get the network interface number associated with a TCP socket.

Retrieves the network interface number being used by an established TCP connection.

Note

The return value is undefined for listening sockets.

Parameters

<i>fd</i>	Socket file descriptor

Returns

Network interface number

See also

[GetInterfaceBlock\(\)](#), [InterfaceBlock](#)

Expand for Example Usage

Examples

Basic Usage - Getting Socket Interface

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

int ifNum = TcpGetSocketInterface(fd);
printf("Socket is using interface number: %d\n", ifNum);

close(fd);
```

Multi-Interface System - Identifying Active Interface

```
int fd = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(8080);
serverAddr.sin_addr.s_addr = inet_addr("192.168.1.100");

if (connect(fd, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) == 0) {
    int ifNum = TcpGetSocketInterface(fd);
    InterfaceBlock *pIfBlock = GetInterfaceBlock(ifNum);

    if (pIfBlock != NULL) {
        printf("Connected via interface %d\n", ifNum);
        printf("Interface IP: %I\n", pIfBlock->ip4.cur_addr.i4);
        printf("Interface name: %s\n", pIfBlock->GetInterfaceName());
    }
}

close(fd);
```

Routing Verification - Check Expected Interface

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr*)&addr, sizeof(addr));

int actualInterface = TcpGetSocketInterface(fd);
int expectedInterface = 0; // Expected to use eth0

if (actualInterface == expectedInterface) {
    printf("Connection using correct interface\n");
} else {
    printf("Warning: Using interface %d, expected %d\n",
        actualInterface, expectedInterface);
}

close(fd);
```

Server Connection Logging

```
void HandleClient(int clientFd) {
    int ifNum = TcpGetSocketInterface(clientFd);
    InterfaceBlock *pIfBlock = GetInterfaceBlock(ifNum);

    struct sockaddr_in clientAddr;
    socklen_t addrLen = sizeof(clientAddr);
    getpeername(clientFd, (struct sockaddr*)&clientAddr, &addrLen);

    printf("Client %I connected via interface %d (%s)\n",
        clientAddr.sin_addr.s_addr,
        ifNum,
        pIfBlock ? pIfBlock->GetInterfaceName() : "unknown");

    // Process client request...

    close(clientFd);
}
```

Load Balancing - Track Interface Usage

```
struct InterfaceStats {
    int connectionCount;
    uint32_t bytesTransferred;
};

InterfaceStats ifStats[2] = {0}; // Track 2 interfaces

void TrackConnection(int fd, int bytesTransferred) {
    int ifNum = TcpGetSocketInterface(fd);

    if (ifNum >= 0 && ifNum < 2) {
        ifStats[ifNum].connectionCount++;
        ifStats[ifNum].bytesTransferred += bytesTransferred;

        printf("Interface %d stats: %d connections, %lu bytes\n",
            ifNum,
            ifStats[ifNum].connectionCount,
            ifStats[ifNum].bytesTransferred);
    }
}
```

Failover Detection

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
int primaryInterface = 0;

connect(fd, (struct sockaddr*)&addr, sizeof(addr));

int activeInterface = TcpGetSocketInterface(fd);

if (activeInterface != primaryInterface) {
    printf("WARNING: Connection failed over to backup interface %d\n",
        activeInterface);
    // Log failover event
    LogFailoverEvent(primaryInterface, activeInterface);
} else {
    printf("Connection established on primary interface %d\n",
        primaryInterface);
}
```

Diagnostic Output for All Connections

```
void DiagnoseConnections(int *fdArray, int fdCount) {
    printf("\n=== Connection Diagnostics ===\n");

    for (int i = 0; i < fdCount; i++) {
        int fd = fdArray[i];
        int ifNum = TcpGetSocketInterface(fd);

        struct sockaddr_in peerAddr;
        socklen_t addrLen = sizeof(peerAddr);
        getpeername(fd, (struct sockaddr*)&peerAddr, &addrLen);

        printf("FD %d: Peer %I via interface %d\n",
            fd, peerAddr.sin_addr.s_addr, ifNum);
    }

    printf("=====\n\n");
}
```

14.53.14.41 TcpGetSocketState()

```
uint8_t TcpGetSocketState (  
    int fd)
```

```
#include <tcp.h>
```

Get the current TCP state of a socket.

Queries the current state of a TCP socket connection. The returned value corresponds to one of the states in the TCP state machine as defined in RFC 793.

Parameters

<i>fd</i>	Socket file descriptor

Returns

Current socket state: [TCP Socket State](#)

14.53.14.42 TcpGetTxBufferAvailSpace()

```
uint16_t TcpGetTxBufferAvailSpace (  
    int fd)
```

```
#include <tcp.h>
```

Get the available space in the transmit buffer.

Returns the number of bytes that can be written to the socket's transmit buffer before it becomes full.

Parameters

<i>fd</i>	Socket file descriptor

Returns

Number of bytes available for writing

See also

[TcpGetRxBufferSpaceUsed\(\)](#), [TcpGetTxDataWaiting\(\)](#)

14.53.14.43 TcpGetTxDataWaiting()

```
uint16_t TcpGetTxDataWaiting (  
    int fd)
```

```
#include <tcp.h>
```

Get the number of bytes queued for transmission.

Returns the number of bytes currently in the transmit buffer waiting to be sent to the remote host. This includes both data that has not yet been transmitted and data that has been transmitted but not yet acknowledged.

Parameters

<i>fd</i>	Socket file descriptor

Returns

Number of bytes queued for transmission

See also

[TcpGetTxBufferAvailSpace\(\)](#), [TcpGetRxBufferSpaceUsed\(\)](#)

14.53.14.44 TcpSendKeepAlive()

```
void TcpSendKeepAlive (
    int fd)
```

```
#include <tcp.h>
```

Send a TCP keepalive probe packet.

Transmits a TCP keepalive packet to the remote host to verify the connection is still active. The keepalive packet is an empty segment with a sequence number one less than the last sent packet. If the remote host is still connected, it will respond with a TCP ACK packet, which will update the socket's lastRxTime.

This function is used with [TcpGetLastRxTime\(\)](#) to implement TCP keepalive functionality and detect stale or dead connections.

Parameters

<i>fd</i>	Socket file descriptor

Notes:

- The keepalive probe does not contain any data - it's purely a mechanism to elicit an ACK response from the remote host.
- If the remote host is unreachable or the connection is dead, no response will be received and the lastRxTime will not change.
- This is a lightweight operation that generates minimal network traffic (typically just the TCP header with no payload).

See also

[TcpGetLastRxTime\(\)](#), [TcpGetLastRxInterval\(\)](#)

Expand for Example Usage

Examples

Basic Keepalive Usage

```
#include <tcp.h>
#include <nbrtos.h>

void SimpleKeepaliveExample(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    printf("Connected - sending keepalive every 30 seconds\r\n");

    while (1)
    {
        // Wait 30 seconds
        OSTimeDly(TICKS_PER_SECOND * 30);

        // Send keepalive
        TcpSendKeepAlive(fd);
        printf("Keepalive sent at %lu seconds\r\n", Secs);

        // Check if connection is still valid
        TcpState state = TcpGetSocketState(fd);
        if (state != TCP_STATE_ESTABLISHED)
        {
            printf("Connection lost (state: %d)\r\n", state);
            break;
        }
    }

    close(fd);
}
```


Keepalive with Response Verification

```
#include <tcp.h>
#include <nbrtos.h>

void VerifiedKeepaliveExample(void *pd)
{
    IPADDR serverIp = AsciiToIp("192.168.1.100");
    uint16_t serverPort = 8080;

    int fd = connect(serverIp, serverPort, TICKS_PER_SECOND * 10);

    if (fd < 0)
    {
        printf("Connection failed: %d\r\n", fd);
        return;
    }

    printf("Connected - monitoring with keepalives\r\n");

    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND * 30);

        // Get RX time before keepalive
        uint32_t rxBefore = TcpGetLastRxTime(fd);

        // Send keepalive
        TcpSendKeepAlive(fd);
        printf("Keepalive sent, waiting for ACK...\r\n");

        // Wait for response (typically very quick)
        OSTimeDly(TICKS_PER_SECOND * 3);

        // Check if we got a response
        uint32_t rxAfter = TcpGetLastRxTime(fd);

        if (rxAfter != rxBefore)
        {
            uint32_t responseTimeMs = ((rxAfter - rxBefore) * 1000) / TICKS_PER_SECOND;
            printf("Keepalive ACK received (RTT: %lu ms)\r\n", responseTimeMs);
        }
        else
        {
            printf("WARNING: No response to keepalive!\r\n");

            // Retry once more
            rxBefore = TcpGetLastRxTime(fd);
            TcpSendKeepAlive(fd);
            OSTimeDly(TICKS_PER_SECOND * 5);
            rxAfter = TcpGetLastRxTime(fd);

            if (rxAfter == rxBefore)
            {
                printf("Connection dead - no response to keepalive retry\r\n");
                break;
            }
        }
    }

    close(fd);
}
```

Multiple Connections with Coordinated Keepalives

```
#include <tcp.h>
#include <nbrtos.h>

#define MAX_CONNECTIONS 5

struct ManagedConnection
{
    int fd;
    IPADDR serverIp;
    uint32_t lastKeepaliveTime;
    bool alive;
};

void SendKeepalivesToAll(ManagedConnection *connections, int count)
{
    uint32_t currentTime = Secs;

    for (int i = 0; i < count; i++)
    {
        if (connections[i].fd > 0 && connections[i].alive)
        {
            // Check if it's time for keepalive (every 30 seconds)

```

```

        if ((currentTime - connections[i].lastKeepaliveTime) >= 30)
        {
            uint32_t rxBefore = TcpGetLastRxTime(connections[i].fd);

            TcpSendKeepAlive(connections[i].fd);
            printf("Keepalive sent to %I\r\n", connections[i].serverIp);

            connections[i].lastKeepaliveTime = currentTime;

            // Quick check for response
            OSTimeDly(TICKS_PER_SECOND);

            uint32_t rxAfter = TcpGetLastRxTime(connections[i].fd);

            if (rxAfter == rxBefore)
            {
                printf("No keepalive response from %I\r\n",
                    connections[i].serverIp);
                connections[i].alive = false;
            }
        }
    }
}

void MultiConnectionKeepaliveTask(void *pd)
{
    ManagedConnection connections[MAX_CONNECTIONS];

    // Initialize connections
    const char *servers[] = {
        "192.168.1.100",
        "192.168.1.101",
        "192.168.1.102",
        "192.168.1.103",
        "192.168.1.104"
    };

    for (int i = 0; i < MAX_CONNECTIONS; i++)
    {
        connections[i].serverIp = AsciiToIp(servers[i]);
        connections[i].fd = connect(connections[i].serverIp, 8080,
            TICKS_PER_SECOND * 10);
        connections[i].lastKeepaliveTime = Secs;
        connections[i].alive = (connections[i].fd > 0);

        if (connections[i].alive)
        {
            printf("Connected to %I\r\n", connections[i].serverIp);
        }
    }

    // Monitor all connections
    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND * 5);

        SendKeepalivesToAll(connections, MAX_CONNECTIONS);
    }

    // Cleanup
    for (int i = 0; i < MAX_CONNECTIONS; i++)
    {
        if (connections[i].fd > 0)
        {
            close(connections[i].fd);
        }
    }
}

```

14.53.14.45 WaitForSocketFlush()

```

BOOL WaitForSocketFlush (
    int fd,
    uint32_t ticks)

```

```
#include <tcp.h>
```

Wait for all transmitted data to be acknowledged.

Blocks until all data sent on the socket has been acknowledged by the remote host or until the specified timeout expires. A socket is considered "flushed" when all transmitted data has been acknowledged.

Parameters

<i>fd</i>	Socket file descriptor
<i>ticks</i>	Maximum wait time in system ticks

Return values

<i>TRUE</i>	All sent data has been acknowledged
<i>FALSE</i>	Timeout occurred before all data was acknowledged

See also

[TcpAllDataAcked\(\)](#)

14.53.15 TCP Notification System

Advanced callback mechanism for asynchronous TCP socket event notification.

Typedefs

- typedef void [tcp_notify_handler](#)(int tcpFd)
TCP notification callback function type.

Functions

- void [RegisterTCPReadNotify](#) (int tcpFd, [tcp_notify_handler](#) *notifyHandler)
Register a callback for read event notifications.
- void [RegisterTCPWriteNotify](#) (int tcpFd, [tcp_notify_handler](#) *notifyHandler)
Register a callback for write event notifications.

14.53.15.1 Detailed Description

Advanced callback mechanism for asynchronous TCP socket event notification.

```
#include< tcp.h>
```

The TCP notification system provides an event-driven approach to handling TCP socket activity. Instead of polling sockets for read/write availability, applications can register callback functions that are automatically invoked when:

- Data arrives and is ready to be read (read notification)
- Buffer space becomes available for writing (write notification)

Callback functions have full access to normal TCP socket operations and can perform any standard read/write operations during notification handling.

14.53.15.2 Typedef Documentation

14.53.15.2.1 tcp_notify_handler

```
typedef void tcp_notify_handler(int tcpFd)
#include <tcp.h>
```

TCP notification callback function type.

Applications implement callback functions of this type to handle TCP socket notifications. The callback is invoked from the TCP stack when the registered event occurs (data available for reading or buffer space available for writing).

Parameters

<i>tcpFd</i>	File descriptor of the socket that triggered the notification

See also

[RegisterTCPReadNotify\(\)](#), [RegisterTCPWriteNotify\(\)](#)

Expand for Example Usage

Examples

Basic Read Callback Implementation

```
void MyReadHandler(int tcpFd) {
    char buffer[256];
    int bytesRead = read(tcpFd, buffer, sizeof(buffer));

    if (bytesRead > 0) {
        printf("Received %d bytes via callback\n", bytesRead);
        buffer[bytesRead] = '\0';
        printf("Data: %s\n", buffer);
    }
}
```

Basic Write Callback Implementation

```
void MyWriteHandler(int tcpFd) {
    const char *data = "Response from write callback\n";
    int bytesWritten = write(tcpFd, data, strlen(data));

    if (bytesWritten > 0) {
        printf("Wrote %d bytes via callback\n", bytesWritten);
    }
}
```

State Machine Callback

```
enum ConnectionState { WAITING_HEADER, WAITING_DATA, PROCESSING };

struct ClientContext {
    ConnectionState state;
    int expectedBytes;
    char buffer[1024];
    int bufferPos;
};

ClientContext contexts[10]; // Support up to 10 clients

void StateMachineHandler(int tcpFd) {
    ClientContext *ctx = &contexts[tcpFd % 10];

    switch (ctx->state) {
        case WAITING_HEADER:
            if (dataavail(tcpFd) >= 4) {
                read(tcpFd, &ctx->expectedBytes, 4);
                ctx->state = WAITING_DATA;
                ctx->bufferPos = 0;
            }
            break;

        case WAITING_DATA:
            int available = dataavail(tcpFd);
            int toRead = (ctx->expectedBytes - ctx->bufferPos);
            if (toRead > available) toRead = available;

            int n = read(tcpFd, ctx->buffer + ctx->bufferPos, toRead);
            ctx->bufferPos += n;

            if (ctx->bufferPos >= ctx->expectedBytes) {
                ctx->state = PROCESSING;
                ProcessData(ctx->buffer, ctx->expectedBytes);
                ctx->state = WAITING_HEADER;
            }
            break;
    }
}
```

Multiple Socket Handler with Context

```

struct SocketData {
    int fd;
    char name[32];
    uint32_t bytesReceived;
};

SocketData socketContexts[5];

void UniversalReadHandler(int tcpFd) {
    // Find context for this socket
    SocketData *ctx = NULL;
    for (int i = 0; i < 5; i++) {
        if (socketContexts[i].fd == tcpFd) {
            ctx = &socketContexts[i];
            break;
        }
    }

    if (ctx != NULL) {
        char buffer[128];
        int n = read(tcpFd, buffer, sizeof(buffer));
        ctx->bytesReceived += n;

        printf("[%s] Received %d bytes (total: %lu)\n",
            ctx->name, n, ctx->bytesReceived);
    }
}

```

14.53.15.3 Function Documentation

14.53.15.3.1 RegisterTCPReadNotify()

```

void RegisterTCPReadNotify (
    int tcpFd,
    tcp_notify_handler * notifyHandler)

```

#include <tcp.h>

Register a callback for read event notifications.

Registers a callback function to be invoked whenever data becomes available for reading on the specified TCP socket. The callback is called automatically by the TCP stack when new data arrives.

Parameters

<i>tcpFd</i>	File descriptor of the socket to monitor
<i>notifyHandler</i>	Pointer to the callback function

See also

[RegisterTCPWriteNotify\(\)](#), [tcp_notify_handler](#)

Expand for Example Usage

Examples

Basic Registration - Echo Server

```

void EchoHandler(int tcpFd) {
    char buffer[256];
    int n = read(tcpFd, buffer, sizeof(buffer));
    if (n > 0) {
        write(tcpFd, buffer, n); // Echo back
    }
}

void UserMain(void *pd) {
    init();

    int listenFd = socket(AF_INET, SOCK_STREAM, 0);
    // ... bind and listen ...

    int clientFd = accept(listenFd, NULL, NULL);

    // Register read notification
    RegisterTCPReadNotify(clientFd, EchoHandler);

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

```
    }
}
```

Multiple Client Handler

```
void ClientReadHandler(int tcpFd) {
    char buffer[512];
    int n = read(tcpFd, buffer, sizeof(buffer));

    if (n > 0) {
        buffer[n] = '\0';
        printf("Client FD %d: %s\n", tcpFd, buffer);
        ProcessCommand(tcpFd, buffer);
    } else if (n < 0) {
        printf("Client FD %d disconnected\n", tcpFd);
        close(tcpFd);
    }
}

void ServerTask(void *pd) {
    int listenFd = socket(AF_INET, SOCK_STREAM, 0);
    // ... bind and listen ...

    while (1) {
        int clientFd = accept(listenFd, NULL, NULL);
        if (clientFd > 0) {
            printf("New client connected, FD: %d\n", clientFd);
            RegisterTCPReadNotify(clientFd, ClientReadHandler);
        }
        OSTimeDly(TICKS_PER_SECOND / 10);
    }
}
```

Event-Driven HTTP Server

```
void HttpReadHandler(int tcpFd) {
    static char requestBuffer[2048];
    static int bufferPos = 0;

    int n = read(tcpFd, requestBuffer + bufferPos,
        sizeof(requestBuffer) - bufferPos - 1);

    if (n > 0) {
        bufferPos += n;
        requestBuffer[bufferPos] = '\0';

        // Check for end of HTTP headers
        if (strstr(requestBuffer, "\r\n\r\n") != NULL) {
            ProcessHttpRequest(tcpFd, requestBuffer);
            bufferPos = 0; // Reset for next request
        }
    }
}

void StartEventDrivenHttpServer() {
    int listenFd = socket(AF_INET, SOCK_STREAM, 0);
    // ... bind to port 80 and listen ...

    while (1) {
        int clientFd = accept(listenFd, NULL, NULL);
        if (clientFd > 0) {
            RegisterTCPReadNotify(clientFd, HttpReadHandler);
        }
    }
}
```

Removing Notification (Unregister)

```
void TemporaryReadHandler(int tcpFd) {
    char buffer[128];
    int n = read(tcpFd, buffer, sizeof(buffer));

    if (n > 0) {
        printf("Received: %.*s\n", n, buffer);

        // Unregister after first read
        RegisterTCPReadNotify(tcpFd, NULL);
        printf("Read notification removed\n");
    }
}

void UserMain(void *pd) {
    init();
    int fd = socket(AF_INET, SOCK_STREAM, 0);
    connect(fd, ...);
}
```

```

    RegisterTCPReadNotify(fd, TemporaryReadHandler);

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Protocol Parser with Notification

```

#define PROTO_START_BYTE 0xAA
#define PROTO_END_BYTE 0x55

void ProtocolReadHandler(int tcpFd) {
    static char msgBuffer[256];
    static int msgPos = 0;
    static bool inMessage = false;

    while (dataavail(tcpFd) > 0) {
        char byte;
        read(tcpFd, &byte, 1);

        if (byte == PROTO_START_BYTE) {
            inMessage = true;
            msgPos = 0;
        } else if (byte == PROTO_END_BYTE && inMessage) {
            ProcessProtocolMessage(msgBuffer, msgPos);
            inMessage = false;
            msgPos = 0;
        } else if (inMessage) {
            if (msgPos < sizeof(msgBuffer)) {
                msgBuffer[msgPos++] = byte;
            }
        }
    }
}

```

14.53.15.3.2 RegisterTCPWriteNotify()

```

void RegisterTCPWriteNotify (
    int tcpFd,
    tcp_notify_handler * notifyHandler)

```

#include <tcp.h>

Register a callback for write event notifications.

Registers a callback function to be invoked whenever buffer space becomes available for writing on the specified TCP socket. The callback is called automatically by the TCP stack when the transmit buffer has space available.

Parameters

<i>tcpFd</i>	File descriptor of the socket to monitor
<i>notifyHandler</i>	Pointer to the callback function

See also

[RegisterTCPReadNotify\(\)](#), [tcp_notify_handler](#)

Expand for Example Usage

Examples

Basic Write Notification - Queued Data Transfer

```

struct DataQueue {
    char buffer[4096];
    int size;
    int sent;
} dataQueue;

void WriteNotifyHandler(int tcpFd) {
    if (dataQueue.sent < dataQueue.size) {
        int remaining = dataQueue.size - dataQueue.sent;
        int n = write(tcpFd, dataQueue.buffer + dataQueue.sent, remaining);

        if (n > 0) {
            dataQueue.sent += n;
            printf("Sent %d bytes, %d remaining\n", n,
                dataQueue.size - dataQueue.sent);
        }
    }
}

```

```

        if (dataQueue.sent >= dataQueue.size) {
            // All data sent, unregister
            RegisterTCPWriteNotify(tcpFd, NULL);
            printf("Transfer complete\n");
        }
    }
}

void QueueDataForSend(int fd, const char *data, int len) {
    memcpy(dataQueue.buffer, data, len);
    dataQueue.size = len;
    dataQueue.sent = 0;

    RegisterTCPWriteNotify(fd, WriteNotifyHandler);
}

```

Large File Transfer with Write Notifications

```

struct FileTransfer {
    int fd;
    FILE *file;
    uint32_t totalBytes;
    uint32_t sentBytes;
};

FileTransfer transfer;

void FileWriteHandler(int tcpFd) {
    char buffer[1024];
    size_t bytesRead = fread(buffer, 1, sizeof(buffer), transfer.file);

    if (bytesRead > 0) {
        int n = write(tcpFd, buffer, bytesRead);
        if (n > 0) {
            transfer.sentBytes += n;
            printf("Progress: %lu / %lu bytes (%.1f%%)\n",
                transfer.sentBytes, transfer.totalBytes,
                (float)transfer.sentBytes / transfer.totalBytes * 100);
        }
    }

    if (feof(transfer.file)) {
        fclose(transfer.file);
        RegisterTCPWriteNotify(tcpFd, NULL);
        printf("File transfer complete\n");
    }
}

void StartFileTransfer(int tcpFd, const char *filename) {
    transfer.file = fopen(filename, "rb");
    if (transfer.file) {
        fseek(transfer.file, 0, SEEK_END);
        transfer.totalBytes = ftell(transfer.file);
        fseek(transfer.file, 0, SEEK_SET);
        transfer.sentBytes = 0;
        transfer.fd = tcpFd;

        RegisterTCPWriteNotify(tcpFd, FileWriteHandler);
    }
}

```

Periodic Data Streaming

```

struct StreamContext {
    uint32_t sequenceNumber;
    bool streaming;
} stream = {0, true};

void StreamWriteHandler(int tcpFd) {
    if (stream.streaming) {
        char packet[64];
        int len = snprintf(packet, sizeof(packet),
            "STREAM:%lu:%lu\n",
            stream.sequenceNumber++, Secs);

        int n = write(tcpFd, packet, len);
        if (n > 0) {
            printf("Streamed packet #%lu\n", stream.sequenceNumber - 1);
        }

        // Continue streaming (notification will fire when space available)
    } else {
        // Stop streaming, unregister
        RegisterTCPWriteNotify(tcpFd, NULL);
    }
}

```



```

    }

    void StartStreaming(int fd) {
        stream.sequenceNumber = 0;
        stream.streaming = true;
        RegisterTCPWriteNotify(fd, StreamWriteHandler);
    }

    void StopStreaming() {
        stream.streaming = false;
    }

```

Write Notification for Flow Control

```

volatile bool canWrite = true;

void FlowControlWriteHandler(int tcpFd) {
    // Buffer space available, we can write again
    canWrite = true;

    // Unregister until we need it again
    RegisterTCPWriteNotify(tcpFd, NULL);
    printf("Write buffer available\n");
}

void SendDataWithFlowControl(int tcpFd, const char *data, int len) {
    int n = write(tcpFd, data, len);

    if (n < len) {
        // Couldn't send all data, buffer is full
        printf("Buffer full, sent only %d of %d bytes\n", n, len);
        canWrite = false;

        // Register for notification when space becomes available
        RegisterTCPWriteNotify(tcpFd, FlowControlWriteHandler);

        // Wait for space
        while (!canWrite) {
            OSTimeDly(TICKS_PER_SECOND / 100);
        }

        // Retry sending remaining data
        SendDataWithFlowControl(tcpFd, data + n, len - n);
    }
}

```

Combined Read and Write Notifications

```

void CombinedReadHandler(int tcpFd) {
    char buffer[256];
    int n = read(tcpFd, buffer, sizeof(buffer));

    if (n > 0) {
        printf("Read %d bytes\n", n);
        ProcessAndQueueResponse(tcpFd, buffer, n);
    }
}

void CombinedWriteHandler(int tcpFd) {
    if (HasQueuedResponse()) {
        char *response = GetQueuedResponse();
        int len = strlen(response);
        int n = write(tcpFd, response, len);

        if (n == len) {
            // All sent, unregister write notify
            RegisterTCPWriteNotify(tcpFd, NULL);
        }
    }
}

void SetupConnection(int tcpFd) {
    RegisterTCPReadNotify(tcpFd, CombinedReadHandler);
    RegisterTCPWriteNotify(tcpFd, CombinedWriteHandler);
}

```

14.53.16 TCP Socket Options

Configuration flags for TCP socket behavior.

Macros

- #define SO_DEBUG 1

Reserved for debugging purposes.

- **#define SO_NONAGLE 2**
Disable Nagle's algorithm (send data immediately without delay)
- **#define SO_NOPUSH 4**
Disable TCP PUSH flag (buffer data before sending)

14.53.16.1 Detailed Description

Configuration flags for TCP socket behavior.

```
#include< tcp.h>
```

Socket options for use with [setsockopt\(\)](#), [clrsockopt\(\)](#), and [getsockopt\(\)](#).

14.53.17 TCP Socket State

TCP connection state machine values (RFC 793)

Macros

- **#define TCP_STATE_CLOSED (0)**
Socket is closed, no connection exists.
- **#define TCP_STATE_LISTEN (1)**
Socket is listening for incoming connections.
- **#define TCP_STATE_SYN_SENT (2)**
SYN packet sent, waiting for SYN-ACK.
- **#define TCP_STATE_SYN_RCVD (3)**
SYN received, SYN-ACK sent.
- **#define TCP_STATE_WAIT_FOR_ACCEPT (4)**
Connection pending, waiting for [accept\(\)](#) call.
- **#define TCP_STATE_ESTABLISHED (5)**
Connection established, data transfer possible.
- **#define TCP_STATE_CLOSE_WAIT (6)**
FIN received from remote host, local close pending.
- **#define TCP_STATE_LAST_ACK (7)**
FIN sent, waiting for final ACK.
- **#define TCP_STATE_FIN_WAIT_1 (8)**
Local FIN sent, waiting for ACK.
- **#define TCP_STATE_FIN_WAIT_2 (9)**
Local FIN acknowledged, waiting for remote FIN.
- **#define TCP_STATE_CLOSING (10)**
Both sides closing simultaneously.
- **#define TCP_STATE_TIME_WAIT (11)**
Connection closed, waiting to ensure remote received final ACK.

14.53.17.1 Detailed Description

TCP connection state machine values (RFC 793)

```
#include< tcp.h>
```

These constants represent the current state of a TCP connection according to the TCP state machine defined in RFC 793. These values should not be modified as they correspond to the TCP protocol specification.

14.53.18 TCP Socket Status

Status codes and error values for TCP socket operations.

Macros

- `#define TCP_ERR_NORMAL (0)`
Operation completed successfully with no errors.
- `#define TCP_ERR_TIMEOUT (-1)`
Operation timed out before completion.
- `#define TCP_ERR_NOCON (-2)`
No active connection exists on this socket.
- `#define TCP_ERR_CLOSING (-3)`
Socket is currently in the closing state.
- `#define TCP_ERR_NOSUCH_SOCKET (-4)`
The specified socket does not exist.
- `#define TCP_ERR_NONE_AVAIL (-5)`
No sockets available in the system pool.
- `#define TCP_ERR_CON_RESET (-6)`
Connection was reset by remote host.
- `#define TCP_ERR_CON_ABORT (-7)`
Connection was aborted.

14.53.18.1 Detailed Description

Status codes and error values for TCP socket operations.

```
#include< tcp.h>
```

When calling API functions that return a file descriptor (such as [listen\(\)](#), [accept\(\)](#), [connect\(\)](#)), a negative value indicates an error condition, while a positive value represents a valid file descriptor number.

Note

See detailed return code information, causes and solutions at: [Programmers Guide TCP Error Codes](#)

14.54 TFTP

Topics

- [TFTP Client](#)
- [TFTP Server](#)

14.54.1 Detailed Description

The NetBurner TFTP Group

```
#include< tftp_server.h>
```

The NetBurner TFTP Group

14.54.2 TFTP Client

Topics

- [TFTP Return Codes](#)

Functions

- int [GetTFTP](#) (PCSTR fileName, PCSTR mode, puint8_t buffer, int &len, uint32_t timeout, [IPADDR4](#) server, uint16_t port=IANA_TFTP_PORT)
Get a file from a TFTP server.
- int [SendTFTP](#) (PCSTR fileName, PCSTR mode, puint8_t buffer, int len, uint32_t timeout, uint32_t packetTimeout, [IPADDR4](#) server, uint16_t port=IANA_TFTP_PORT)
Send a file to a TFTP server.

14.54.2.1 Detailed Description

```
#include< tftp.h>
```

The NetBurner TFTP Client Group

14.54.2.2 Function Documentation

14.54.2.2.1 GetTFTP()

```
int GetTFTP (
    PCSTR fileName,
    PCSTR mode,
    puint8_t buffer,
    int & len,
    uint32_t timeout,
    IPADDR4 server,
    uint16_t port = IANA_TFTP_PORT)
```

```
#include <tftp.h>
```

Get a file from a TFTP server.

Parameters

<i>fileName</i>	Name of file to retrieve
<i>mode</i>	Transfer mode. 'b' = binary, 't' = text
<i>buffer</i>	Data destination buffer
<i>len</i>	Maximum length when function is called, bytes read upon function return.
<i>timeout</i>	Total amount of time for file transfer, in system TimeTicks
<i>server</i>	IPv4 address of TFTP server
<i>port</i>	Optional TFTP server port number. If not specified the default TFTP port number 69 will be used.

Returns

[TFTP Return Codes](#)

See also

[SendTFTP](#)

14.54.2.2.2 SendTFTP()

```
int SendTFTP (
    PCSTR fileName,
    PCSTR mode,
    puint8_t buffer,
    int len,
    uint32_t timeout,
    uint32_t packetTimeout,
```

```

        IPADDR4 server,
        uint16_t port = IANA_TFTP_PORT)
#include <tftp.h>
Send a file to a TFTP server.

```

Parameters

<i>fileName</i>	Name of file to retrieve
<i>mode</i>	Transfer mode. 'b' = binary, 't' = text
<i>buffer</i>	Data destination buffer
<i>len</i>	Length of file to transfer, in bytes
<i>timeout</i>	Total amount of time for file transfer, in system TimeTicks
<i>packetTimeout</i>	Timeout for any single UDP packet to successfully transfer and be acknowledged, in system Time Ticks
<i>server</i>	IPv4 address of TFTP server
<i>port</i>	Optional TFTP server port number. If not specified the default TFTP port number 69 will be used.

Returns

[TFTP Return Codes](#)

See also

[SendTFTP](#)

14.54.2.3 TFTP Return Codes**Macros**

- `#define TFTP_OK (0)`
Success.
- `#define TFTP_TIMEOUT (1)`
TFTP Timeout.
- `#define TFTP_ERROR (2)`
TFTP Error.

14.54.2.3.1 Detailed Description

```
#include< tftp.h>
```

14.54.3 TFTP Server**Classes**

- class [TFTP_Server](#)
TFTP Server Class.
- class [TFTP_Server_source_File](#)
TFTP Server Source File.

14.54.3.1 Detailed Description

```
#include< tftp_server.h>
```

The NetBurner TFTP Server Group

14.55 Time

Time management functions including time zones, NTP, and system clock.

Classes

- struct [TimeZoneRecord](#)
Time zone record structure containing complete time zone information.

Functions

- time_t [time](#) (time_t *pt)
Gets the current system GMT time.
- time_t [set_time](#) (time_t time_to_set, uint32_t fraction=0)
Set the system time to the value passed in the time_t parameter.
- time_t [timegm](#) (struct tm *bts)
Converts a tm structure to time_t in GMT.
- void [tzsetchar](#) (const char *tzenv)
Set the system local time.

Variables

- const [TimeZoneRecord](#) * [TZRecords](#)
Global array of time zone records.

14.55.1 Detailed Description

Time management functions including time zones, NTP, and system clock.

```
#include< nbttime.h>
```

The NBTime library provides methods to set and read the system time. The system time will reset whenever power is lost or the system is reset. On power-up, system time can be set:

- With a NTP server (most common)
- From a real time clock (RTC)
- Manually

Examples are provided in the example section for each of these methods.

Time is always stored as GMT. If you are using an internal or external real-time clock, it should also be set to GMT. Example application function calls are shown below. Refer to the examples for your specific NetBurner platform for RTC functions. Functions with an "_r" are reentrant.

A partial list of common standard Time library functions include:

- time_t [time](#) (time_t* timeT) Returns current system GMT time as a time_t, either as an argument or return value. (Provide NULL as an argument to just return a time_t)
- time_t [mktime](#) (tm* time) Converts local given time in struct tm format to GMT time as a time_t. Also validates and modifies the time struct to their proper ranges.
- size_t [strftime](#) (char* str, size_t count, const char* format, const tm* tp) Creates a formatted time string from the tm struct tp and stores it in a buffer str.
- tm* [gmtime](#) (time_t &timeT) Converts given time since epoch (a time_t value pointed to by timeT) to calendar time, expressed in Coordinated Universal Time (UTC) in the struct tm format.

- `tm* gmtime_r(time_t &timeT, tm &stmUTC)` Converts given time since epoch (a `time_t` value pointed to by `timeT`) to calendar time, expressed in Coordinated Universal Time (UTC) in the struct `tm` format. The result is stored in the struct `tm` pointed to by `stmUTC`.
- `tm* localtime(time_t &tv)` Converts given time since epoch (`time_t`) to calendar time, expressed in local time, in the struct `tm` format.
- `tm* localtime_r(time_t &tv, tm &tm_struct)` Converts given time since epoch (`time_t`) to calendar time, expressed in local time, in the struct `tm` format. The result is stored in the `tm` struct.
- `size_t strftime(char* str, size_t count, const char* format, const struct tm* tp)` Creates a formatted time string and stores it in a buffer.
- `double difftime(time_t t1, time_t t2)` Returns the difference between two `time_t` values.

For example, to get local time:

```
char buf[80];
time_t RawTime;
struct tm *TimeInfo;

time(&RawTime);           // Get system time in time_t format
TimeInfo = localtime(&RawTime); // Convert to struct tm format
strftime(buf, 79, "%l:%M:%S %p", TimeInfo);
printf("Formatted time string: %s\r\n", buf);
#include< timezones.h>
```

The `NBTime` module provides comprehensive time management capabilities:

- Time zone database with daylight saving time support
- NTP client for network time synchronization
- System clock management and conversions
- Local time calculations with automatic DST adjustments

14.55.2 Function Documentation

14.55.2.1 set_time()

```
time_t set_time (
    time_t time_to_set,
    uint32_t fraction = 0)
#include <nbtime.h>
```

Set the system time to the value passed in the `time_t` parameter.

Parameters

<i>time_to_set</i>	<code>time_t</code> value to set the system time.
<i>fraction</i>	the fraction (in $1/(2^{32})$ of the second)

Returns

The system time `time_t` value

14.55.2.2 time()

```
time_t time (
    time_t * pt)
#include <nbtime.h>
```

Gets the current system GMT time.

Parameters

<i>pt</i>	Optional pointer to a <code>time_t</code> to fill with the current time (may be NULL)

Returns

The system time `time_t` value

14.55.2.3 timegm()

```
time_t timegm (
    struct tm * bts)
```

```
#include <nbtime.h>
```

Converts a `tm` structure to `time_t` in GMT.

Performs the reverse translation that is done by `gmtime()`. This is not to be confused with the `mktime()` function, which performs the reverse conversion of the `localtime()` function.

Parameters

<i>bts</i>	Pointer to a <code>tm</code> structure

Returns

`time_t` value of `tm`

14.55.2.4 tzsetchar()

```
void tzsetchar (
    const char * tzenv)
```

```
#include <nbtime.h>
```

Set the system local time.

This function initializes the TZ environment variable to contain the local time zone offset information from the standard time (UTC) and, if applicable, daylight saving time (DST). This does not overwrite the system time to the local time. The TZ environment variable is used by the `ctime()`, `localtime()`, `mktime()`, and `strftime()` functions of the standard `time.h` library to calculate the local time.

Once the TZ variable is set, the time library functions use it to determine when DST is in effect, when type conversions are made, or what time zone string is provided in the case of the `strftime()` function.

The system provides time zones around the world that can be referenced in the `TZRecords` variable in [timezones.h](#). The time zone strings are located in `timezones.cpp`. Please refer to the `timezones` and `RTC` example programs for methods on utilizing these time zone settings.

Parameters

<i>tzenv</i>	The time zone string. For example: <code>tzsetchar("EST5EDT4,M3.2.0/02:00:00,M11.1.0/02:00:00");</code> Eastern

14.55.3 Variable Documentation**14.55.3.1 TZRecords**

```
const TimeZoneRecord* TZRecords [extern]
#include <timezones.h>
```

Global array of time zone records.

This constant array contains over 75 time zone definitions covering all major regions worldwide. The array is NULL-terminated (the last record has `Posix = NULL`) to allow iteration through all available time zones.

The array is defined in `nbrtos/source/timezones.cpp` and includes time zones sorted by GMT offset and then alphabetically by region.

Accessing Time Zones:

Time zones can be accessed by:

- Iterating through the array until Posix == NULL
- Searching by description string
- Filtering by bUsCanada flag
- Matching against known POSIX strings

Complete Usage Example:

```
#include <timezones.h>
#include <ntp.h>
#include <string.h>

// List all available time zones
void ListTimeZones() {
    iprintf("Available Time Zones:\r\n");
    const TimeZoneRecord *tz = TZRecords;
    int index = 0;

    while (tz->Posix != NULL) {
        iprintf("%3d: %s\r\n", index, tz->Description);
        tz++;
        index++;
    }
}

// Find and set a specific time zone
bool SetTimeZoneByDescription(const char *searchStr) {
    const TimeZoneRecord *tz = TZRecords;

    while (tz->Posix != NULL) {
        if (strstr(tz->Description, searchStr) != NULL) {
            // Found matching time zone
            tzset(tz->Posix);
            iprintf("Time zone set to: %s\r\n", tz->Description);
            return true;
        }
        tz++;
    }

    iprintf("Time zone not found: %s\r\n", searchStr);
    return false;
}

// Get only US/Canada time zones
void ListUSTimeZones() {
    iprintf("US/Canada Time Zones:\r\n");
    const TimeZoneRecord *tz = TZRecords;

    while (tz->Posix != NULL) {
        if (tz->bUsCanada) {
            iprintf("  %s (%s)\r\n", tz->Short_name, tz->Description);
        }
        tz++;
    }
}

// Complete NTP + Time Zone setup
void SetupNetworkTime() {
    // Start NTP client
    if (SetTimeNTPServer("pool.ntp.org")) {
        iprintf("NTP synchronization started\r\n");

        // Wait for time sync
        while (!GetTimeNTPStatus()) {
            OSTimeDly(TICKS_PER_SECOND);
        }

        // Set time zone
        SetTimeZoneByDescription("Pacific Time");

        // Display current local time
        time_t now = time(NULL);
        struct tm *local = localtime(&now);
        iprintf("Current time: %02d:%02d:%02d\r\n",
            local->tm_hour, local->tm_min, local->tm_sec);
    }
}
```

Note

The array is const and should not be modified at runtime

Always check for NULL termination when iterating

Time zone data is stored in flash/ROM and does not consume RAM

14.56 Timers

Topics

- [High Resolution Delay Timer](#)
High-precision microsecond delay functionality.
- [Interval Timer](#)
- [Stopwatch Timer](#)

14.56.1 Detailed Description

Hardware based high resolution timers:

- [StopWatch](#) to time events
- Interval Timer for periodic semaphore posts or interrupts
- HiResDelay ([DelayObject](#)) providing microsecond delay functions

14.56.2 High Resolution Delay Timer

High-precision microsecond delay functionality.

Classes

- class [DelayObject](#)
High-resolution microsecond delay timer class.

14.56.2.1 Detailed Description

High-precision microsecond delay functionality.

```
#include< HiResDelay.h>
```

The High Resolution Delay Timer provides microsecond-precision timing by utilizing hardware system timers. This is ideal for applications requiring precise timing control such as:

- Protocol timing requirements
- Sensor polling with specific intervals
- Hardware interface timing
- Signal generation

14.56.2.2 Usage Overview

When a [DelayObject](#) is created, it automatically allocates an available hardware system timer from the processor's timer pool. The timer remains allocated for the lifetime of the object and is automatically freed when the object goes out of scope or is destroyed.

14.56.2.3 Timing Precision

- **Minimum delay:** Platform-dependent, typically ~ 1 -10 microseconds
- **Maximum delay:** Limited by 32-bit counter (up to $\sim 4,294$ seconds)
- **Precision:** Hardware timer precision (typically sub-microsecond)
- **Accuracy:** Depends on system load and interrupt latency

14.56.2.4 Example Usage

```
#include <HiResDelay.h>

void performPreciseTiming() {
    // Create delay object - allocates hardware timer
    static DelayObject myHiResDelay;

    // Verify timer was successfully allocated
    if (!myHiResDelay.valid()) {
        iprintf("Error: No free timers available\n");
        return;
    }

    // Perform precise delays
    myHiResDelay.DelayUsec(100); // Delay 100 microseconds
    myHiResDelay.DelayUsec(200); // Delay 200 microseconds
    myHiResDelay.DelayUsec(1000); // Delay 1 millisecond
}

// Timer is automatically freed when object goes out of scope
```

14.56.2.5 Important Notes

- Hardware timers are a limited resource - check valid() after construction
- Delays block the calling task (use RTOS tasks for concurrent operations)
- Very short delays ($< 10\mu s$) may have reduced accuracy due to overhead
- For non-blocking timing, consider using the Interval Timer classes
- Static or global DelayObjects prevent timer reallocation overhead

14.56.2.6 Comparison with Other Timing Methods

Method	Resolution	Blocking	Use Case
OSTimeDly()	20ms (1 tick)	Yes	General delays
DelayObject	Microseconds	Yes	Precise hardware timing
IntervalTimer	Microseconds	No	Periodic callbacks
Stopwatch	Microseconds	No	Time measurement

See also

[Interval Timer](#) Interval Timer for non-blocking periodic operations

[Stopwatch Timer](#) Stopwatch for time measurement

OSTimeDly() Standard RTOS delay function

14.56.3 Interval Timer

Functions

- int [IntervalOSSem](#) (OS_SEM *p_toSem, int num_per_sec, int timer=FIRST_UNUSED_TIMER)
Create a periodic timer that posts to an RTOS semaphore.
- int [IntervalOSFlag](#) (OS_FLAGS *p_toFlag, uint32_t flag_value, int num_per_sec, int timer=FIRST_UNUSED_TIMER)

Create a periodic timer that sets RTOS flags.

- int [IntervalInterruptCallback](#) (void(*p_toCallbackFunc)(), int num_per_sec, int timer=FIRST_UNUSED_TIMER)

Create a periodic timer that calls a function from interrupt context.

- void [IntervalStop](#) (int timer_number)

Stop and release an interval timer.

14.56.3.1 Detailed Description

```
#include< IntervalTimer.h>
```

The Interval Timer API provides hardware-based periodic interrupts for real-time applications. Unlike software timers, these use dedicated hardware timer peripherals to generate precise, low-jitter periodic events.

14.56.3.1.1 Features

- Periodic interrupt generation with configurable frequency (minimum 20 Hz)
- Direct callback execution from interrupt context (for immediate response)
- RTOS semaphore posting (for task synchronization)
- RTOS flag setting (for event notification)
- Multiple independent timers (platform-dependent)
- Automatic timer allocation or manual timer selection
- Microsecond-level precision and consistency

14.56.3.1.2 Use Cases

- **Periodic sampling:** ADC reads, sensor polling at fixed intervals
- **Real-time control:** PID loops, motor control updates
- **Task synchronization:** Waking tasks at precise intervals
- **Time-critical events:** Watchdog servicing, status updates
- **Signal generation:** PWM, timing signals for external devices
- **Rate limiting:** Throttling operations to specific frequencies

14.56.3.1.3 Basic Usage - Semaphore Posting

```
// Create and initialize a semaphore
OS_SEM timerSem;
timerSem.Init();

// Create interval timer that posts 20 times per second
int timerNumber = IntervalOSSem(&timerSem, 20);
if (timerNumber < 0)
{
    iprintf("Error creating interval timer: %d\r\n", timerNumber);
}
else
{
    iprintf("Timer %d created successfully\r\n", timerNumber);
}

// Task waits for periodic semaphore posts
while (running)
{
    timerSem.Pend(); // Blocks until timer posts (50ms intervals)
    ProcessPeriodicData();
}

// Clean up when done
IntervalStop(timerNumber);
```

14.56.3.1.4 Advanced Usage - Multiple Timers with Different Rates

```

OS_SEM fastSem, slowSem;
fastSem.Init();
slowSem.Init();

// Fast sampling at 1000 Hz
int fastTimer = IntervalOSSem(&fastSem, 1000);

// Slow processing at 10 Hz
int slowTimer = IntervalOSSem(&slowSem, 10);

// Fast task
DWORD FastTask(void *pd)
{
    while (1)
    {
        fastSem.Pend(); // Every 1ms
        ReadSensors();
    }
}

// Slow task
DWORD SlowTask(void *pd)
{
    while (1)
    {
        slowSem.Pend(); // Every 100ms
        UpdateDisplay();
    }
}

// Cleanup
IntervalStop(fastTimer);
IntervalStop(slowTimer);

```

14.56.3.1.5 Interrupt Callback Usage

```

volatile uint32_t interruptCount = 0;

// Callback runs in interrupt context - keep it fast!
void TimerCallback()
{
    interruptCount++;
    // Toggle a GPIO pin
    Pins[42].toggle();
}

// Setup 100 Hz callback
int timer = IntervalInterruptCallback(TimerCallback, 100);

// Main loop can monitor interrupt count
while (1)
{
    OSTimeDly(TICKS_PER_SECOND);
    iprintf("Interrupts per second: %lu\n", interruptCount);
    interruptCount = 0;
}

IntervalStop(timer);

```

14.56.3.1.6 OS Flag Usage

```

OS_FLAGS controlFlags;
const uint32_t TIMER_EVENT = 0x0001;
const uint32_t OTHER_EVENT = 0x0002;

// Setup 50 Hz flag posting
int timer = IntervalOSFlag(&controlFlags, TIMER_EVENT, 50);

// Task waits for flags
while (running)
{
    uint32_t flags = controlFlags.PendAll(TIMER_EVENT | OTHER_EVENT,
                                           TICKS_PER_SECOND);

    if (flags & TIMER_EVENT)
        HandlePeriodicEvent();
    if (flags & OTHER_EVENT)
        HandleOtherEvent();
}

IntervalStop(timer);

```

14.56.3.1.7 Important Considerations

14.56.3.1.7.1 Interrupt Context (for `IntervalInterruptCallback`)

Callback functions execute in interrupt context with the following restrictions:

- **Keep callbacks short:** Long execution blocks other interrupts
- **No blocking calls:** Cannot call `Pend()`, `OSTimeDly()`, or other blocking functions
- **Be careful with shared data:** Use volatile and appropriate synchronization
- **Limited stack space:** Interrupt stacks are smaller than task stacks
- **Higher priority:** Interrupts preempt all tasks

14.56.3.1.7.2 Frequency Limitations

- **Minimum frequency:** 20 Hz (50ms period)
- **Maximum frequency:** Platform and system load dependent
- **CPU overhead:** Higher frequencies consume more CPU time
- **Jitter:** Very high frequencies may experience more timing variation

14.56.3.1.7.3 Timer Resources

- Hardware timers are limited (typically 4-8 per platform)
- Each `IntervalOSSem/Flag/Callback` call uses one timer
- Always call `IntervalStop()` to free timers when done
- Use `FIRST_UNUSED_TIMER` for automatic allocation

14.56.3.1.7.4 Thread Safety

- Timer setup/teardown is not automatically synchronized
- Protect timer creation/destruction with mutexes if called from multiple tasks
- Callbacks and posts are inherently thread-safe (interrupt-driven)

14.56.3.1.8 Performance Guidelines

Frequency	Use Case	Callback Execution Time
20-100 Hz	Display updates, slow sensors	< 100 us
100-1000 Hz	Control loops, fast sampling	< 10 us
1000+ Hz	High-speed I/O, signal generation	< 1 us

14.56.3.1.9 Error Codes

- **-1:** No hardware timer available (all timers in use)
- **-2:** Invalid frequency (`num_per_sec < 20`)
- **Positive value:** Timer number (success)

14.56.3.1.10 Best Practices

1. **Always check return values** from Interval functions
2. **Call [IntervalStop\(\)](#)** when timer no longer needed to free resources
3. **Use semaphores for task sync** rather than callbacks when possible
4. **Profile callback execution time** to ensure it doesn't exceed period
5. **Avoid dynamic memory allocation** in interrupt callbacks
6. **Use volatile** for variables shared between interrupts and tasks
7. **Document timer usage** in your code to track resource allocation

14.56.3.1.11 Common Pitfalls

- Forgetting to call [IntervalStop\(\)](#) causes timer leaks
- Callbacks that take too long can cause system instability
- Using blocking functions in callbacks causes crashes
- Not checking return values leads to silent failures
- Excessive interrupt frequency can starve normal tasks

See also

[High Resolution Delay Timer](#) For precise one-shot delays

[Stopwatch Timer](#) For measuring time intervals

[OSTimeDly\(\)](#) For task-level delays

14.56.3.2 Function Documentation

14.56.3.2.1 IntervalInterruptCallback()

```
int IntervalInterruptCallback (
    void(* p_toCallbackFunc )(),
    int num_per_sec,
    int timer = FIRST_UNUSED_TIMER)
```

```
#include <IntervalTimer.h>
```

Create a periodic timer that calls a function from interrupt context.

Creates a hardware interval timer that calls the specified callback function at the requested frequency. The callback executes in interrupt context, providing the lowest possible latency and jitter for time-critical operations.

Use with extreme caution. Interrupt callbacks have strict requirements and restrictions. They are appropriate for simple, fast operations like toggling GPIO pins, reading sensors, or updating counters. For most periodic tasks, [IntervalOSSem\(\)](#) or [IntervalOSFlag\(\)](#) are safer and more appropriate choices.

14.56.3.2.2 When to Use Interrupt Callbacks

- Time-critical operations requiring sub-millisecond response
- Simple hardware I/O that must occur at precise intervals
- Generating timing signals or PWM-like patterns
- High-frequency data acquisition where task overhead is too high

14.56.3.2.3 When NOT to Use Interrupt Callbacks

- Any operation that can be done from a task (use IntervalOSSem instead)
- Operations requiring significant processing time
- Code that needs to call blocking functions
- Complex logic with multiple decision points

14.56.3.2.4 Interrupt Context Restrictions

Your callback function **MUST NOT**:

- Call blocking functions (Pend(), OSTimeDly(), etc.)
- Perform long computations (keep under 10 us at typical frequencies)
- Call printf() or iprintf() (extremely slow)
- Perform dynamic memory allocation
- Access non-volatile shared data without proper synchronization
- Call non-reentrant library functions

Your callback function **SHOULD**:

- Complete as quickly as possible (microseconds, not milliseconds)
- Use volatile for variables shared with tasks
- Be reentrant and thread-safe
- Use atomic operations for counters if possible
- Post to semaphores/flags if triggering task work

Parameters

<i>p_toCallbackFunc</i>	Pointer to the callback function. Must have signature: void MyCallback(void). The function will be called from interrupt context at the specified frequency. Must remain valid (not be deleted) while timer is active.
<i>num_per_sec</i>	Callback frequency in Hz (calls per second). Valid range is 20 to platform-dependent maximum. Consider: • 20-100 Hz: Low-frequency periodic tasks • 100-1000 Hz: Control loops, moderate-speed sampling • 1000-10000 Hz: High-speed I/O, signal generation Higher frequencies require shorter execution times.
<i>timer</i>	Hardware timer number to use. Options: • FIRST_UNUSED_TIMER (default, -1): Automatically selects the first available hardware timer • 0, 1, 2, etc.: Specific timer number (platform-dependent) Use default unless you need specific timer for hardware reasons.

Returns

On success: Non-negative timer number (0, 1, 2, etc.) that identifies this interval timer. Save this value to pass to [IntervalStop\(\)](#). On failure:

- -1: No hardware timer available (all timers in use)
- -2: Invalid num_per_sec (less than 20 Hz)

Note

The callback executes at higher priority than all tasks and most interrupts.

Long callbacks can cause system instability by blocking other interrupts.

If callback execution time exceeds the timer period, interrupts will be missed.

Always call [IntervalStop\(\)](#) when done to free the hardware timer resource.

Warning

Callbacks run with interrupts disabled for their priority level. Keep execution time minimal to avoid blocking other system functions.

Do not call any RTOS blocking functions from the callback. This will cause system crashes or undefined behavior.

The callback function pointer must remain valid. Do not use member functions or lambdas unless you understand their lifetime implications.

Example - Simple counter increment (safe):

```
volatile uint32_t interruptCount = 0;

void CounterCallback()
{
    interruptCount++; // Fast, atomic on most platforms
}

void UserMain(void *pd)
{
    init();

    // Increment counter 1000 times per second
    int timer = IntervalInterruptCallback(CounterCallback, 1000);
    if (timer < 0)
    {
        iprintf("Timer creation failed: %d\n", timer);
        return;
    }

    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND);
        uint32_t count = interruptCount; // Read volatile
        iprintf("Interrupts per second: %lu\n", count);
        interruptCount = 0;
    }
}
```

Example - GPIO toggling for timing signal (safe):

```
void TogglePin()
{
    Pins[42].toggle(); // Fast GPIO operation
}

// Create 1 kHz square wave on pin 42
int timer = IntervalInterruptCallback(TogglePin, 2000); // 2000 Hz = 1 kHz square wave
```

Example - ADC sampling with semaphore post (safe):

```
volatile uint16_t adcValue = 0;
OS_SEM processSem;

void SampleADC()
{
    adcValue = ReadADCRegister(); // Fast hardware read
    processSem.Post(); // Wake processing task
}

void Setup()
{
    processSem.Init();
    int timer = IntervalInterruptCallback(SampleADC, 1000); // 1 kHz sampling
}
```

```

}

void ProcessTask(void *pd)
{
    while (1)
    {
        processSem.Pend();
        uint16_t value = adcValue; // Read volatile
        ProcessSample(value);      // Complex processing in task context
    }
}

```

Example - WRONG - Do not do this (unsafe):

```

void BadCallback()
{
    // WRONG: printf is extremely slow (milliseconds)
    iprintf("Timer fired\n"); // DON'T DO THIS!

    // WRONG: Blocking call in interrupt context
    OSTimeDly(1); // DON'T DO THIS!

    // WRONG: Complex processing taking too long
    for (int i = 0; i < 10000; i++)
        ProcessData(); // DON'T DO THIS!

    // WRONG: Dynamic memory allocation
    char *buffer = new char[100]; // DON'T DO THIS!
}

```

Example - Measuring callback overhead:

```

volatile uint32_t maxCallbackTime = 0;

void MonitoredCallback()
{
    uint32_t start = GetHighResTimer(); // Platform-specific

    // Your time-critical code here
    DoFastOperation();

    uint32_t elapsed = GetHighResTimer() - start;
    if (elapsed > maxCallbackTime)
        maxCallbackTime = elapsed;
}

// In main loop, check maxCallbackTime to ensure it's acceptable

```

Example - Proper synchronization with tasks:

```

volatile uint32_t sensorValue = 0;
volatile bool newDataReady = false;
OS_SEM dataSem;

void SensorCallback()
{
    // Read sensor quickly
    sensorValue = ReadSensor();
    newDataReady = true;
    dataSem.Post(); // Signal task
}

void SensorTask(void *pd)
{
    while (1)
    {
        dataSem.Pend();
        if (newDataReady)
        {
            uint32_t value = sensorValue;
            newDataReady = false;

            // Process in task context (can be slow)
            ProcessSensorData(value);
        }
    }
}

```

See also

[IntervalStop\(\)](#)
[IntervalOSSem\(\)](#)
[IntervalOSFlag\(\)](#)

14.56.3.2.5 IntervalOSFlag()

```

int IntervalOSFlag (

```

```
OS_FLAGS * p_toFlag,
uint32_t flag_value,
int num_per_sec,
int timer = FIRST_UNUSED_TIMER)
```

```
#include <IntervalTimer.h>
```

Create a periodic timer that sets RTOS flags.

Creates a hardware interval timer that sets the specified flag bit(s) in an OS_FLAGS object at the requested frequency. This is ideal for event notification where multiple event sources need to signal a single task, or when you need to distinguish between different types of periodic events.

Unlike semaphores which simply count posts, flags maintain distinct bit states allowing a task to wait on multiple different events simultaneously and determine which event(s) occurred.

14.56.3.2.6 Typical Usage Pattern

1. Create an OS_FLAGS object
2. Define flag bit values for different events
3. Call [IntervalOSFlag\(\)](#) to set specific flag bit(s) periodically
4. Task(s) wait on flags using [PendAll\(\)](#), [PendAny\(\)](#), etc.
5. Call [IntervalStop\(\)](#) when timer is no longer needed

Parameters

<i>p_toFlag</i>	Pointer to an OS_FLAGS object. The flags object need not be initialized before use (has no Init() method). Must remain valid for the lifetime of the interval timer.
<i>flag_value</i>	The flag bit(s) to set on each timer period. Can be a single bit (e.g., 0x0001) or multiple bits (e.g., 0x0003). Use bitwise OR to combine multiple flags. Bits are set, not toggled. Common patterns: • Single bit: 0x0001, 0x0002, 0x0004, 0x0008, etc. • Multiple bits: 0x0003 (bits 0 and 1), 0x000F (bits 0-3)
<i>num_per_sec</i>	Flag setting frequency in Hz (sets per second). Valid range is 20 to platform-dependent maximum (typically 10,000+). Higher frequencies increase CPU overhead. Common values: • 20-50 Hz: UI updates, slow polling • 100-500 Hz: Control system events • 1000+ Hz: High-speed event signaling
<i>timer</i>	Hardware timer number to use. Options: • FIRST_UNUSED_TIMER (default, -1): Automatically selects the first available hardware timer • 0, 1, 2, etc.: Specific timer number (platform-dependent) Use default unless you need specific timer for hardware reasons.

Returns

On success: Non-negative timer number (0, 1, 2, etc.) that identifies this interval timer. Save this value to pass to [IntervalStop\(\)](#). On failure:

- -1: No hardware timer available (all timers in use)
- -2: Invalid num_per_sec (less than 20 Hz)

Note

Flags are set (ORed), not cleared. If no task is pending on the flags, they will remain set until a task checks them.

Multiple interval timers can set different flags in the same OS_FLAGS object.

Use flag consumption mode (OSFlagPend with consume option) if you need to clear flags after reading.

Always call [IntervalStop\(\)](#) when done to free the hardware timer resource.

Warning

Do not delete or let the flags object go out of scope while the timer is active. Call [IntervalStop\(\)](#) first.

Example - Basic flag-based periodic event:

```
OS_FLAGS eventFlags;
const uint32_t TIMER_EVENT = 0x0001;
int timerNum;

void EventTask(void *pd)
{
    // Set flag 50 times per second (20ms period)
    timerNum = IntervalOSFlag(&eventFlags, TIMER_EVENT, 50);
    if (timerNum < 0)
    {
        iprintf("Failed to create timer: %d\n", timerNum);
        return;
    }

    while (1)
    {
        // Wait for timer flag, auto-clear on return
        uint32_t flags = eventFlags.Pend(TIMER_EVENT, OS_FLAG_WAIT_SET_ALL);

        // This runs exactly 50 times per second
        PeriodicProcessing();
    }
}
```

Example - Multiple periodic events with different rates:

```
OS_FLAGS systemFlags;
const uint32_t FAST_EVENT = 0x0001; // Bit 0
const uint32_t SLOW_EVENT = 0x0002; // Bit 1

int fastTimer, slowTimer;

void Setup()
{
    // Fast event: 100 Hz
    fastTimer = IntervalOSFlag(&systemFlags, FAST_EVENT, 100);

    // Slow event: 10 Hz
    slowTimer = IntervalOSFlag(&systemFlags, SLOW_EVENT, 10);
}

void ControlTask(void *pd)
{
    while (1)
    {
        // Wait for either event
        uint32_t flags = systemFlags.Pend(FAST_EVENT | SLOW_EVENT,
                                         OS_FLAG_WAIT_SET_ANY |
                                         OS_FLAG_CONSUME);

        if (flags & FAST_EVENT)
        {
            FastProcessing(); // Runs every 10ms
        }

        if (flags & SLOW_EVENT)
        {
            SlowProcessing(); // Runs every 100ms
        }
    }
}

void Cleanup()
{
    IntervalStop(fastTimer);
    IntervalStop(slowTimer);
}
```

Example - Combining timer events with other events:

```
OS_FLAGS controlFlags;
```

```

const uint32_t TIMER_EVENT      = 0x0001;
const uint32_t BUTTON_EVENT    = 0x0002;
const uint32_t NETWORK_EVENT    = 0x0004;

int timer = IntervalOSFlag(&controlFlags, TIMER_EVENT, 20);

// Other parts of code can set BUTTON_EVENT or NETWORK_EVENT
void ButtonISR()
{
    controlFlags.Post(BUTTON_EVENT);
}

void StateMachine(void *pd)
{
    while (1)
    {
        // Wait for any event
        uint32_t events = controlFlags.Pend(TIMER_EVENT |
                                           BUTTON_EVENT |
                                           NETWORK_EVENT,
                                           OS_FLAG_WAIT_SET_ANY |
                                           OS_FLAG_CONSUME);

        // Handle each event type
        if (events & TIMER_EVENT)
            HandlePeriodicUpdate();

        if (events & BUTTON_EVENT)
            HandleButtonPress();

        if (events & NETWORK_EVENT)
            HandleNetworkData();
    }
}

```

Example - Watchdog pattern:

```

OS_FLAGS watchdogFlags;
const uint32_t WATCHDOG_FLAG = 0x0001;
const uint32_t TASK_ALIVE_FLAG = 0x0002;

int watchdogTimer = IntervalOSFlag(&watchdogFlags, WATCHDOG_FLAG, 1); // 1 Hz

void WatchdogTask(void *pd)
{
    while (1)
    {
        uint32_t flags = watchdogFlags.Pend(WATCHDOG_FLAG,
                                           OS_FLAG_WAIT_SET_ALL |
                                           OS_FLAG_CONSUME,
                                           TICKS_PER_SECOND * 2);

        if (flags & WATCHDOG_FLAG)
        {
            // Check if monitored task is alive
            uint32_t status = watchdogFlags.Pend(TASK_ALIVE_FLAG,
                                                OS_FLAG_WAIT_SET_ALL |
                                                OS_FLAG_CONSUME,
                                                0); // No wait

            if (!(status & TASK_ALIVE_FLAG))
            {
                iprintf("ERROR: Monitored task not responding!\n");
                HandleTaskFailure();
            }
        }
    }
}

// Monitored task periodically sets its alive flag
void MonitoredTask(void *pd)
{
    while (1)
    {
        DoWork();
        watchdogFlags.Post(TASK_ALIVE_FLAG);
        OSTimeDly(TICKS_PER_SECOND / 2);
    }
}

```

See also

[IntervalStop\(\)](#)
[IntervalOSSem\(\)](#)
[IntervalInterruptCallback\(\)](#)
[OS_FLAGS](#)

14.56.3.2.7 IntervalOSSem()

```
int IntervalOSSem (
    OS_SEM * p_toSem,
    int num_per_sec,
    int timer = FIRST_UNUSED_TIMER)
```

```
#include <IntervalTimer.h>
```

Create a periodic timer that posts to an RTOS semaphore.

Creates a hardware interval timer that posts to the specified semaphore at the requested frequency. This is the preferred method for periodic task synchronization, as it allows tasks to block efficiently on the semaphore while the hardware timer ensures precise timing.

The timer uses a hardware peripheral to generate periodic interrupts. Each interrupt posts to the semaphore, waking any task waiting on it. This provides low-jitter, precise periodic events for time-critical operations.

14.56.3.2.8 Typical Usage Pattern

1. Create and initialize an OS_SEM
2. Call [IntervalOSSem\(\)](#) to start periodic posting
3. Task(s) call [Pend\(\)](#) on the semaphore to wait for each period
4. Call [IntervalStop\(\)](#) when timer is no longer needed

Parameters

<i>p_toSem</i>	Pointer to an initialized OS_SEM object. The semaphore must be initialized with Init() before passing to this function. Must remain valid for the lifetime of the interval timer.
<i>num_per_sec</i>	Posting frequency in Hz (posts per second). Valid range is 20 to platform-dependent maximum (typically 10,000+). Higher frequencies increase CPU overhead. Common values: • 20-50 Hz: UI updates, slow sensors • 100-500 Hz: Control loops, moderate sampling • 1000+ Hz: High-speed data acquisition
<i>timer</i>	Hardware timer number to use. Options: • FIRST_UNUSED_TIMER (default, -1): Automatically selects the first available hardware timer • 0, 1, 2, etc.: Specific timer number (platform-dependent) Use default unless you need specific timer for hardware reasons.

Returns

On success: Non-negative timer number (0, 1, 2, etc.) that identifies this interval timer. Save this value to pass to [IntervalStop\(\)](#). On failure:

- -1: No hardware timer available (all timers in use)
- -2: Invalid num_per_sec (less than 20 Hz)

Note

The semaphore counter will increment each period until a task calls [Pend\(\)](#). If no task is pending, posts accumulate (up to semaphore max count).

Multiple tasks can wait on the same semaphore. The RTOS will wake them according to priority and FIFO order.

Always call [IntervalStop\(\)](#) when done to free the hardware timer resource.

Warning

Do not delete or let the semaphore go out of scope while the timer is active. Call [IntervalStop\(\)](#) first.

Frequencies above 1000 Hz may consume significant CPU time. Profile your system to ensure adequate CPU remains for other tasks.

Example - Basic periodic task:

```
OS_SEM periodicSem;
int timerNum;

void PeriodicTask(void *pd)
{
    periodicSem.Init();

    // Post 100 times per second (10ms period)
    timerNum = IntervalOSSem(&periodicSem, 100);
    if (timerNum < 0)
    {
        iprintf("Failed to create timer: %d\n", timerNum);
        return;
    }

    while (1)
    {
        // Wait for next period
        periodicSem.Pend();

        // This runs exactly 100 times per second
        ProcessData();
    }
}

// Cleanup function
void StopPeriodicTask()
{
    if (timerNum >= 0)
        IntervalStop(timerNum);
}
```

Example - Multiple tasks synchronized to one timer:

```
OS_SEM syncSem;
int timer;

void TaskA(void *pd)
{
    while (1)
    {
        syncSem.Pend();
        DoTaskAWork(); // Runs every 50ms
    }
}

void TaskB(void *pd)
{
    while (1)
    {
        syncSem.Pend();
        DoTaskBWork(); // Also runs every 50ms
    }
}

void UserMain(void *pd)
{
    init();
    syncSem.Init(0, 100); // Init with count 0, max 100

    // 20 Hz timer (50ms period)
    timer = IntervalOSSem(&syncSem, 20);

    OSTaskCreate(TaskA, ...);
    OSTaskCreate(TaskB, ...);

    // Both tasks wake on each timer period
}
```

Example - Error handling:

```
OS_SEM sem;
sem.Init();

int timer = IntervalOSSem(&sem, 500);
switch (timer)
{
    case -1:
        iprintf("ERROR: No hardware timers available\n");
        iprintf("Check for timer leaks - are you calling IntervalStop()?\n");
        break;
}
```

```

    case -2:
        iprintf("ERROR: Invalid frequency (minimum 20 Hz)\n");
        break;
    default:
        iprintf("Timer %d created successfully at 500 Hz\n", timer);
        break;
}

```

Example - Timeout with periodic check:

```

OS_SEM checkSem;
checkSem.Init();

int timer = IntervalOSSem(&checkSem, 10); // Check 10 times/second

bool WaitForConditionWithTimeout(int timeoutSeconds)
{
    int checks = timeoutSeconds * 10; // 10 checks per second

    for (int i = 0; i < checks; i++)
    {
        checkSem.Pend();
        if (ConditionMet())
        {
            IntervalStop(timer);
            return true;
        }
    }

    IntervalStop(timer);
    return false; // Timeout
}

```

See also

[IntervalStop\(\)](#)
[IntervalOSFlag\(\)](#)
[IntervalInterruptCallback\(\)](#)
[OS_SEM](#)

14.56.3.2.9 IntervalStop()

```

void IntervalStop (
    int timer_number)
#include <IntervalTimer.h>

```

Stop and release an interval timer.

Stops the specified interval timer and frees the associated hardware timer resource, making it available for reuse. After calling this function, the timer will no longer generate interrupts, post to semaphores, set flags, or call callbacks.

Always call this function when you no longer need an interval timer. Failing to stop timers causes resource leaks, eventually exhausting the limited number of hardware timers available on the platform.

This function is safe to call from any context (task or interrupt) and is safe to call multiple times on the same timer number (subsequent calls have no effect).

14.56.3.2.10 When to Call IntervalStop()

- When your application no longer needs periodic events
- Before re-configuring a timer with different parameters
- In cleanup code before task termination
- When switching between different operational modes
- Before system shutdown or reset

14.56.3.2.11 Resource Management Pattern

```

// Setup
int timer = IntervalOSSem(...);
if (timer >= 0)
{
    // Use timer
    ...
}

```



```

    // Cleanup
    IntervalStop(timer);
}

```

Parameters

<i>timer_number</i>	The timer number returned by IntervalOSSem() , IntervalOSFlag() , or IntervalInterruptCallback() . Valid timer numbers are non-negative integers (0, 1, 2, etc.). Negative values are ignored (function returns immediately).

Note

After stopping, the timer number becomes invalid and should not be reused.

For [IntervalOSSem\(\)](#), pending semaphore posts are preserved - the semaphore counter is not cleared by stopping the timer.

For [IntervalOSFlag\(\)](#), flags remain in their current state - they are not cleared by stopping the timer.

For [IntervalInterruptCallback\(\)](#), the callback will not be called again after this function returns, but you must ensure the callback function itself remains valid if it was mid-execution.

This function is thread-safe and can be called from any task or interrupt.

It is safe to call [IntervalStop\(\)](#) with an invalid or already-stopped timer number - the function will simply return without error.

Example - Basic cleanup:

```

OS_SEM timerSem;
int timer;

void StartPeriodicTask()
{
    timerSem.Init();
    timer = IntervalOSSem(&timerSem, 100);
}

void StopPeriodicTask()
{
    if (timer >= 0)
    {
        IntervalStop(timer);
        timer = -1; // Mark as stopped
    }
}

```

Example - RAIL-style resource management:

```

class PeriodicTask
{
private:
    OS_SEM sem;
    int timerNum;

public:
    PeriodicTask(int hz) : timerNum(-1)
    {
        sem.Init();
        timerNum = IntervalOSSem(&sem, hz);
    }

    ~PeriodicTask()
    {
        if (timerNum >= 0)
            IntervalStop(timerNum);
    }

    void WaitForPeriod()
    {
        sem.Pend();
    }
};

void UsePeriodicTask()
{
    PeriodicTask task(50); // 50 Hz

    while (running)
    {
        task.WaitForPeriod();
    }
}

```

```

        DoWork();
    }

    // Timer automatically stopped when task goes out of scope
}

```

Example - Multiple timers management:

```

#define MAX_TIMERS 4
int activeTimers[MAX_TIMERS];
int timerCount = 0;

int CreateTimer(...)
{
    int timer = IntervalOSSem(...);
    if (timer >= 0 && timerCount < MAX_TIMERS)
    {
        activeTimers[timerCount++] = timer;
    }
    return timer;
}

void StopAllTimers()
{
    for (int i = 0; i < timerCount; i++)
    {
        IntervalStop(activeTimers[i]);
    }
    timerCount = 0;
}

```

Example - Mode switching:

```

int currentTimer = -1;

void SwitchToFastMode()
{
    // Stop old timer if active
    if (currentTimer >= 0)
        IntervalStop(currentTimer);

    // Start new timer at higher frequency
    currentTimer = IntervalOSSem(&sem, 1000); // 1 kHz
}

void SwitchToSlowMode()
{
    if (currentTimer >= 0)
        IntervalStop(currentTimer);

    currentTimer = IntervalOSSem(&sem, 10); // 10 Hz
}

```

Example - Safe shutdown:

```

volatile bool shutdownRequested = false;
int timer = -1;

void PeriodicTask(void *pd)
{
    OS_SEM sem;
    sem.Init();

    timer = IntervalOSSem(&sem, 100);
    if (timer < 0)
    {
        iprintf("Timer creation failed\n");
        return;
    }

    while (!shutdownRequested)
    {
        sem.Pend();
        DoWork();
    }

    // Clean shutdown
    IntervalStop(timer);
    timer = -1;
    iprintf("Task shutdown complete\n");
}

void RequestShutdown()
{
    shutdownRequested = true;
    // Timer will be stopped by the task itself
}

```

Example - Error recovery:

```

int timer = -1;

bool RecoverFromError()

```

```

{
    // Stop old timer if it exists
    if (timer >= 0)
    {
        IntervalStop(timer);
        timer = -1;
    }

    // Try to restart
    timer = IntervalOSSem(&sem, 100);
    if (timer < 0)
    {
        iprintf("Recovery failed - no timers available\n");
        return false;
    }

    iprintf("Recovery successful - timer %d\n", timer);
    return true;
}

```

See also

[IntervalOSSem\(\)](#)

[IntervalOSFlag\(\)](#)

[IntervalInterruptCallback\(\)](#)

14.56.4 Stopwatch Timer

Classes

- class [StopWatch](#)

High-resolution stopwatch timer for precise event timing and performance measurement.

14.56.4.1 Detailed Description

```
#include< stopwatch.h>
```

The [StopWatch](#) class provides high-resolution timing capabilities for measuring code execution time, profiling performance, and timing events with precision beyond the standard RTOS tick resolution.

14.56.4.1.1 Features

- High-resolution timing using hardware timers
- Start/Stop functionality with cumulative timing
- Clear and reset capabilities
- Time reading without stopping the timer
- Platform-independent API with automatic timer allocation

14.56.4.1.2 Basic Usage

```

StopWatch myStopwatch;

myStopwatch.Start();
for (volatile int i = 0; i < numLoop; i++)
{
    count++;
}
myStopwatch.Stop();

unsigned long long elapsed = myStopwatch.GetTime();
double seconds = myStopwatch.Convert(elapsed);
printf("Elapsed time: %f seconds\n", seconds);

```

14.56.4.1.3 Advanced Usage - Cumulative Timing

```
StopWatch timer;

// Time multiple operations cumulatively
timer.Start();
DoOperation1();
timer.Stop();

// Continue timing without clearing
timer.Start(); // Continues from previous elapsed time
DoOperation2();
timer.Stop();

double totalTime = timer.Convert(timer.GetTime());
printf("Total time for both operations: %f seconds\n", totalTime);
```

14.56.4.1.4 Timing Without Stopping

```
StopWatch timer;
timer.Start();

while (processing)
{
    DoWork();

    // Check elapsed time without stopping
    unsigned long long current = timer.GetTime();
    if (current > MAX_ALLOWED_TIME)
    {
        printf("Warning: Operation taking too long!\n");
        break;
    }
}
timer.Stop();
```

14.56.4.1.5 Timer Resolution

```
StopWatch timer;
double resolution = timer.CountResolution();
printf("Timer resolution: %f seconds per tick\n", resolution);
```

Note

On platforms with dedicated RTOS timers, the `timer_number` parameter in the constructor can be omitted and the system will automatically allocate an appropriate timer.

The [StopWatch](#) maintains cumulative time across Start/Stop cycles unless explicitly cleared with `Clear()` or `Start(true)`.

See also

- [Interval Timer](#) For periodic timing and callbacks
- [High Resolution Delay Timer](#) For precise delays
- `OSTimeDly()` For RTOS-based delays

14.57 UDP

NetBurner User Datagram Protocol (UDP) Implementation.

Topics

- [UDP Error Codes](#)
- [UDP Packet API](#)
- [UDP Socket API](#)

BSD-style socket functions for UDP communication.

Classes

- class [UDPPacket](#)

UDP Packet Class - Complete UDP packet management.

Typedefs

- typedef void [udp_data_notify](#)(OS_FIFO *pfifo, uint16_t port)
UDP packet arrival notification callback function type.

Functions

- bool [RegisterUDPFifo](#) (uint16_t listenPort, OS_FIFO *pFifo)
Register FIFO to receive UDP packets on specified port.
- bool [RegisterUDPFifoVia](#) (uint16_t listenPort, OS_FIFO *pFifo, int interface)
Register FIFO to receive UDP packets on specified port via specific network interface.
- uint16_t [RegisterEphemeralFifo](#) (OS_FIFO *pfifo, int ifn=-1)
Register UDP FIFO on a random unused ephemeral port.
- bool [RegisterUDPFifoWithNotify](#) (uint16_t listenPort, OS_FIFO *pFifo, [udp_data_notify](#) *pNotifyFunction)
Register UDP FIFO with callback notification on packet arrival.
- bool [RegisterUDPFifoWithNotifyVia](#) (uint16_t listenPort, OS_FIFO *pFifo, [udp_data_notify](#) *pNotifyFunction, int interface)
Register UDP FIFO with callback notification on specific network interface.
- void [UnregisterUDPFifo](#) (uint16_t listenPort, bool drain=false)
Unregister UDP FIFO and stop receiving packets on specified port.

14.57.1 Detailed Description

NetBurner User Datagram Protocol (UDP) Implementation.

```
#include< udp.h>
```

Complete UDP protocol implementation providing both object-oriented C++ interface ([UDPPacket](#) class) and BSD-style socket functions for flexible network communication.

14.57.1.1 Features

- **Dual API Design:** Choose between OOP [UDPPacket](#) class or familiar BSD sockets
- **IPv4 and IPv6 Support:** Full dual-stack networking capabilities
- **Zero-Copy Operations:** Direct buffer access for maximum efficiency
- **Quality of Service (QoS):** DSCP marking for traffic prioritization
- **Multi-Interface Routing:** Send packets via specific network interfaces
- **Automatic Memory Management:** RAII-based resource cleanup in C++ API
- **Fragment Handling:** Support for large datagrams exceeding MTU
- **FIFO-Based Reception:** Non-blocking packet queue management

14.57.1.2 When to Use UDP

UDP is ideal for applications requiring:

- **Low latency:** No connection overhead or retransmission delays
- **Broadcast/Multicast:** One-to-many communication patterns
- **Real-time data:** Streaming sensor data, video, audio, gaming
- **Simple request/response:** DNS, SNMP, NTP, TFTP protocols
- **Custom protocols:** Where application handles reliability

Not suitable for: File transfers, critical data, ordered delivery requirements (use TCP for these scenarios)

14.57.1.3 API Selection Guide

14.57.1.3.1 Use UDPPacket Class When:

- Writing new C++ code
- Need automatic memory management
- Sending to multiple destinations
- Require direct buffer manipulation
- Working with NetBurner-specific features (OS_FIFO integration)

14.57.1.3.2 Use BSD Socket Functions When:

- Porting existing socket code
- Compatibility with standard POSIX code
- Integration with [select\(\)/poll\(\)](#) event loops
- Writing in C (not C++)
- Familiarity with Berkeley sockets

14.57.1.4 Quick Start Examples

Examples

Example 1: Simple UDP Echo Server (UDPPacket)

```
#include <ucos.h>
#include <udp.h>

const char *AppName = "UDP Echo Server";

void UserMain(void *pd) {
    init();
    StartHttp();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    // Create FIFO for receiving packets
    OS_FIFO udpFifo;
    OSFifoInit(&udpFifo);
    RegisterUDPFifo(5000, &udpFifo); // Listen on port 5000

    printf("UDP Echo Server listening on port 5000\r\n");

    while (1) {
        UDPPacket pkt(&udpFifo, TICKS_PER_SECOND * 60);

        if (pkt.Validate()) {
            IPADDR srcIP = pkt.GetSourceAddress();
            uint16_t srcPort = pkt.GetSourcePort();
            uint16_t dataLen = pkt.GetDataSize();

            printf("Received %d bytes from %s:%d\r\n",
                  dataLen, AsciiIP(srcIP).c_str(), srcPort);

            // Echo back to sender
            pkt.SetDestinationPort(srcPort);
            pkt.Send(srcIP);
        }
    }
}
```

Example 2: UDP Sensor Client (UDPPacket)

```
#include <ucos.h>
#include <udp.h>

void SendSensorData(IPADDR serverIP, uint16_t serverPort, int sensorValue) {
    UDPPacket pkt;

    pkt.SetSourcePort(0); // Auto-assign ephemeral port
    pkt.SetDestinationPort(serverPort);
    pkt.SetDSCP(46); // Mark as high-priority (VoIP class)

    // Format message directly into packet buffer
```

```

uint16_t len = sprintf(pkt.GetDataBuffer(),
                      "SENSOR:%d,TEMP:%d",
                      GetDeviceID(), sensorValue);

pkt.SetDataSize(len);

pkt.Send(serverIP); // Send and auto-free buffer
}

```

Example 3: UDP Server with BSD Sockets

```

#include <ucos.h>
#include <udp.h>

void UserMain(void *pd) {
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    int sock = CreateRxUdpSocket(8080);
    if (sock < 0) {
        printf("Failed to create socket\r\n");
        return;
    }

    uint8_t buffer[1024];
    IPADDR clientIP;
    uint16_t localPort, remotePort;

    while (1) {
        int n = recvfrom(sock, buffer, sizeof(buffer),
                        &clientIP, &localPort, &remotePort);

        if (n > 0) {
            buffer[n] = '\0'; // Null-terminate for printing
            printf("Received: %s from %s:%d\r\n",
                  buffer, AsciiIP(clientIP).c_str(), remotePort);

            // Echo back
            sendto(sock, buffer, n, clientIP, remotePort);
        }
    }
}

```

Example 4: Multi-Interface UDP Routing

```

// Send packet via specific network interface
UDPPacket pkt;
pkt.SetSourcePort(3000);
pkt.SetDestinationPort(4000);
pkt.AddData("Hello from interface 1");

// Send via interface 1 (e.g., second Ethernet port)
pkt.SendViaInterfaceNum(destIP, 1);

```

Example 5: Broadcast UDP Discovery

```

void SendDiscoveryBroadcast() {
    UDPPacket pkt;
    pkt.SetSourcePort(5555);
    pkt.SetDestinationPort(5555);
    pkt.AddData("DISCOVER:NetBurner");

    // Broadcast to local subnet
    IPADDR broadcastAddr = AsciiToIp("255.255.255.255");
    pkt.Send(broadcastAddr);
}

```

14.57.1.5 Memory Management

14.57.1.5.1 UDPPacket Automatic Cleanup

The `UDPPacket` class uses RAIL (Resource Acquisition Is Initialization):

- Constructor allocates buffer from pool
- Destructor automatically returns buffer to pool
- `Send()` frees buffer after transmission
- `SendAndKeep()` retains buffer for multiple sends

14.57.1.5.2 Buffer Lifecycle Example

```
void SendToMultipleHosts() {
    UDPPacket pkt;
    pkt.SetSourcePort(1000);
    pkt.SetDestinationPort(2000);
    pkt.AddData("Broadcast message");

    // Send copies to multiple destinations
    pkt.SendAndKeep(host1IP); // Sends copy, keeps buffer
    pkt.SendAndKeep(host2IP); // Sends copy, keeps buffer
    pkt.Send(host3IP);        // Sends and frees buffer

    // pkt destructor runs here, but buffer already freed by Send()
}
```

14.57.1.6 Port Number Guidelines

- **0**: Auto-assign ephemeral port (49152-65535 range)
- **1-1023**: Well-known ports (requires root on some systems)
- **1024-49151**: Registered ports for applications
- **49152-65535**: Dynamic/private ports

Common UDP Ports:

- 53: DNS
- 67/68: [DHCP](#)
- 123: NTP (Network Time Protocol)
- 161/162: SNMP
- 514: Syslog
- 5353: mDNS
- 1900: SSDP (UPnP)

14.57.1.7 Quality of Service (QoS)

Use `SetDSCP()` to prioritize traffic in QoS-enabled networks:

DSCP Value | Traffic Class | Use Case

- 0 (CS0): Best Effort - Normal traffic
- 8 (CS1): Bulk Data - Background transfers
- 10 (AF11): High-throughput data
- 18 (AF21): Low-latency data
- 26 (AF31): Multimedia streaming
- 34 (AF41): Multimedia conferencing
- 46 (EF): Expedited Forwarding - VoIP, real-time gaming
- 48 (CS6): Network control traffic

```
UDPPacket voipPacket;
voipPacket.SetDSCP(46); // Mark as VoIP for low latency
```


14.57.1.8 Performance Considerations

14.57.1.8.1 Buffer Size Limits

- **IPv4 MTU:** 1500 bytes (typical Ethernet)
- **UDP Header:** 8 bytes
- **IP Header:** 20 bytes (IPv4), 40 bytes (IPv6)
- **Maximum Safe Payload:** ~1472 bytes (IPv4), ~1452 bytes (IPv6)
- **Fragmentation:** Available for larger packets (use [SendFragmentedUdpPacket\(\)](#))

14.57.1.8.2 Optimization Tips

1. Use appropriate data insertion method:

- `GetDataBuffer()` + `SetDataSize()`: Best for `sprintf`/binary writes
- `AddData()`: Best for incremental assembly

2. Minimize packet sends:

- Batch data when possible
- Use `SendAndKeep()` only when necessary

3. FIFO sizing:

- Size FIFO based on expected burst traffic
- Monitor FIFO with `GetFifoMessageCount()`

4. Interface binding:

- Bind sockets to specific interfaces in multi-homed systems
- Reduces routing overhead

14.57.1.9 Error Handling

14.57.1.9.1 UDPPacket Validation

```
UDPPacket pkt(&fifo, timeout);

if (!pkt.Validate()) {
    // Could be timeout, checksum error, or invalid packet
    printf("Packet validation failed\r\n");
    return;
}

// Safe to access packet data
puint8_t data = pkt.GetDataBuffer();
```

14.57.1.9.2 Socket Error Codes

```
int sock = CreateRxUdpSocket(port);
if (sock < 0) {
    switch (sock) {
        case UDP_ERR_NOSUCH_SOCKET:
            printf("Socket does not exist\r\n");
            break;
        case UDP_ERR_NOTOPEN_TO_WRITE:
            printf("Socket not open for writing\r\n");
            break;
        case UDP_ERR_NOTOPEN_TO_READ:
            printf("Socket not open for reading\r\n");
            break;
    }
}
```

14.57.1.10 Thread Safety

- **UDPPacket**: Each packet object should be used by single task
- **OS_FIFO**: Thread-safe for multiple consumers/producers
- **BSD Sockets**: Each socket should be accessed by single task
- **Multiple packets**: Create separate **UDPPacket** for each task

14.57.1.11 Advanced Features

14.57.1.11.1 IPv6 Dual-Stack

```
#ifdef IPV6
// Code automatically handles both IPv4 and IPv6
IPADDR destIP = AsciiToIp("2001:db8::1"); // IPv6
// or
IPADDR destIP = AsciiToIp("192.168.1.100"); // IPv4-mapped

UDPPacket pkt;
pkt.Send(destIP); // Works for both
#endif
```

14.57.1.11.2 Fragment Handling

```
// Send large packet (>MTU) with automatic fragmentation
uint8_t largeData[8000];
// ... fill largeData ...

int result = SendFragmentedUdpPacket(
    destIP,
    5000, // source port (0 = auto)
    6000, // dest port
    largeData,
    sizeof(largeData)
);

if (result != sizeof(largeData)) {
    printf("Fragmented send failed\r\n");
}
```

14.57.1.11.3 MAC Address Access

```
UDPPacket pkt(&fifo, timeout);
if (pkt.Validate()) {
    MACADDR srcMAC = pkt.GetMacSource();
    printf("From MAC: %02X:%02X:%02X:%02X:%02X\r\n",
        srcMAC.bytes[0], srcMAC.bytes[1], srcMAC.bytes[2],
        srcMAC.bytes[3], srcMAC.bytes[4], srcMAC.bytes[5]);
}
```

14.57.1.12 Debugging Tips

1. Use **EnableSystemDiagnostics()**: Monitor system resources
2. **Check FIFO overflow**: Increase FIFO size if packets dropped
3. **Validate packets**: Always call Validate() after receiving
4. **Monitor port conflicts**: Ensure unique port assignments
5. **Network tools**: Use Wireshark to capture actual packets

14.57.1.13 Common Pitfalls

14.57.1.13.1 [NO] Forgetting to set data size

```
UDPPacket pkt;
sprintf(pkt.GetDataBuffer(), "Hello");
pkt.Send(ip); // BUG: Size not set!
```

14.57.1.13.2 [YES] Correct approach

```
UDPPacket pkt;
uint16_t len = sprintf(pkt.GetDataBuffer(), "Hello");
pkt.SetDataSize(len); // REQUIRED!
pkt.Send(ip);
```

14.57.1.13.3 [NO] Using packet after Send()

```
UDPPacket pkt;
pkt.AddData("Hello");
pkt.Send(ip);
pkt.Send(ip2); // BUG: Buffer freed by first Send()!
```

14.57.1.13.4 [YES] Use SendAndKeep() for multiple sends

```
UDPPacket pkt;
pkt.AddData("Hello");
pkt.SendAndKeep(ip);
pkt.Send(ip2); // OK: Buffer kept until last Send()
```

14.57.1.13.5 [NO] Not validating received packets

```
UDPPacket pkt(&fifo, timeout);
uint8_t data = pkt.GetDataBuffer(); // BUG: Might be null!
```

14.57.1.13.6 [YES] Always validate

```
UDPPacket pkt(&fifo, timeout);
if (pkt.Validate()) {
    uint8_t data = pkt.GetDataBuffer(); // Safe
}
```

14.57.2 Typedef Documentation**14.57.2.1 udp_data_notify**

```
typedef void udp_data_notify(OS_FIFO *pfifo, uint16_t port)
#include <udp.h>
```

UDP packet arrival notification callback function type.

Defines the function signature for callbacks that are invoked when a UDP packet arrives on a registered port. The callback is executed in interrupt context, allowing immediate notification of packet arrival without polling.

Use this callback type when registering a UDP FIFO with notification functions like [RegisterUDPFifoWithNotify\(\)](#) or [RegisterUDPFifoWithNotifyVia\(\)](#).

Parameters

<i>pfifo</i>	Pointer to the OS_FIFO where the packet will be queued. This is the same FIFO that was registered with the port.
<i>port</i>	The UDP port number on which the packet was received. Useful when the same callback handles multiple ports.

Note

This callback is invoked from interrupt context - keep processing minimal and fast

Do not perform blocking operations, lengthy calculations, or printf statements

The callback is invoked before the packet is queued to the FIFO

You can use OSFifoPost() or other RTOS primitives to signal tasks

Warning

Never call blocking functions or perform long operations in this callback

Do not call [UnregisterUDPFifo\(\)](#) from within this callback

Avoid dynamic memory allocation in interrupt context

Expand for Example Usage**Examples**

Example implementation:

```
// Simple notification - just count packets
void MyUdpNotify(OS_FIFO *pfifo, uint16_t port) {
    static volatile uint32_t packetCount = 0;
    packetCount++; // Fast operation only
}
```

Example with task notification:

```
OS_SEM udpSemaphore;

// Callback signals a semaphore to wake processing task
void UdpArrivalNotify(OS_FIFO *pfifo, uint16_t port) {
    OSSemPost(&udpSemaphore); // Wake up processing task
}

// In task context
void ProcessingTask(void *pdata) {
    while (1) {
        OSSemPend(&udpSemaphore, 0); // Wait for packet arrival

        // Now safe to do full packet processing
        UDPPacket packet;
        if (ReadUDPPacket(&packet, myPort, 0)) {
            ProcessPacket(&packet);
            ReleaseUDPPacket(packet);
        }
    }
}
```

Example with multiple ports:

```
// Single callback handles multiple ports
void MultiPortNotify(OS_FIFO *pfifo, uint16_t port) {
    switch (port) {
        case 5000:
            controlPacketFlag = true;
            break;
        case 5001:
            dataPacketFlag = true;
            break;
        default:
            unknownPacketFlag = true;
            break;
    }
}
```

See also

[RegisterUDPFifoWithNotify\(\)](#)

[RegisterUDPFifoWithNotifyVia\(\)](#)

14.57.3 Function Documentation**14.57.3.1 RegisterEphemeralFifo()**

```
uint16_t RegisterEphemeralFifo (
    OS_FIFO * pfifo,
    int ifn = -1)
```

```
#include <udp.h>
```

Register UDP FIFO on a random unused ephemeral port.

Automatically selects and registers an available UDP port from the ephemeral port range (typically 49152-65535). This is particularly useful for client applications that need a source port for receiving replies but don't require a specific port number.

The function ensures the selected port is not already in use and associates it with the provided FIFO for packet queuing. This is the recommended approach for client-side UDP communication where the remote server will reply to the source port.

Parameters

<i>pfifo</i>	Pointer to an OS_FIFO structure for queuing received packets. The FIFO must be initialized before calling this function. Recommended queue depth: 5-20 packets depending on traffic volume.
<i>ifn</i>	Network interface number to bind to. Use -1 for default interface (recommended). Specify a specific interface (0, 1, 2, etc.) to receive only on that interface.

Returns

The assigned ephemeral port number (49152-65535) on success, or 0 on failure (e.g., no ports available, invalid FIFO, or interface doesn't exist)

Note

The assigned port remains registered until [UnregisterUDPFifo\(\)](#) is called
 Ephemeral ports are typically in the range 49152-65535 (IANA recommended)
 You must call [UnregisterUDPFifo\(\)](#) when done to free the port and resources

Warning

The FIFO must remain valid for the lifetime of the port registration
 Multiple calls will allocate multiple ephemeral ports - track and unregister all of them

Expand for Example Usage

Examples

Example (client sending request and receiving reply):

```
OS_FIFO udpFifo;
OSFifoInit(&udpFifo);

// Get an ephemeral port for receiving replies
uint16_t myPort = RegisterEphemeralFifo(&udpFifo);
if (myPort == 0) {
    printf("Failed to register ephemeral port\n");
    return;
}

printf("Listening on ephemeral port %d\n", myPort);

// Send request to server (server will reply to myPort)
IPADDR server = AsciiToIp("192.168.1.100");
SendUdpPacket(server, myPort, 8000, (uint8_t*)"REQUEST", 7);

// Wait for reply
UDPPacket reply;
if (ReadUDPPacket(&reply, myPort, TICKS_PER_SECOND * 5)) {
    printf("Got reply from %s\n", reply.sourceIpAddr.toString());
    ReleaseUDPPacket(reply);
}

// Cleanup
UnregisterUDPFifo(myPort, true);
```

Example (multiple concurrent requests):

```
struct ClientConnection {
    OS_FIFO fifo;
    uint16_t port;
};

ClientConnection connections[5];

// Create multiple ephemeral ports for parallel requests
for (int i = 0; i < 5; i++) {
    OSFifoInit(&connections[i].fifo);
    connections[i].port = RegisterEphemeralFifo(&connections[i].fifo);
    if (connections[i].port == 0) {
        printf("Failed to register port for connection %d\n", i);
    }
}

// ... use connections ...

// Cleanup all ports
for (int i = 0; i < 5; i++) {
    if (connections[i].port != 0) {
        UnregisterUDPFifo(connections[i].port, true);
    }
}
```

See also

[RegisterUDPFifo\(\)](#) to register a specific port number
[UnregisterUDPFifo\(\)](#) to release the ephemeral port when done
[ReadUDPPacket\(\)](#) to [read](#) packets from the FIFO

14.57.3.2 RegisterUDPFifo()

```
bool RegisterUDPFifo (  
    uint16_t listenPort,  
    OS_FIFO * pFifo)
```

```
#include <udp.h>
```

Register FIFO to receive UDP packets on specified port.

Registers a UDP port for receiving packets and associates it with a FIFO queue for packet buffering. This is the standard method for setting up UDP reception on NetBurner devices. Incoming packets are automatically queued to the FIFO and can be retrieved using [ReadUDPPacket\(\)](#).

The FIFO depth determines how many packets can be buffered before new packets are dropped. Choose a depth based on your expected traffic volume and processing speed. A depth of 5-10 is typical for low-traffic applications, while 20+ may be needed for high-traffic scenarios.

Parameters

<i>listenPort</i>	UDP port number to register (1-65535). Common ports: 80 (HTTP), 443 (HTTPS), 161 (SNMP), etc. Must not already be registered. Port becomes unavailable to other applications.
<i>pFifo</i>	Pointer to an initialized OS_FIFO structure for queuing received packets. Must be initialized with OSFifoInit() before calling this function. The FIFO depth should match your traffic expectations (typically 5-20). The FIFO must remain valid until UnregisterUDPFifo() is called.

Returns

true if port successfully registered, false on failure (e.g., port already in use, invalid FIFO pointer, or out of resources)

Note

The port remains registered until explicitly unregistered with [UnregisterUDPFifo\(\)](#)

Packets exceeding the FIFO depth will be silently dropped

This function registers on all active network interfaces (use [RegisterUDPFifoVia](#) for specific interface)

Well-known ports (1-1023) may require special privileges depending on the system

Warning

The FIFO must remain valid for the entire time the port is registered

Always call [UnregisterUDPFifo\(\)](#) before destroying or deallocating the FIFO

Port numbers must be unique - attempting to register an in-use port will fail

Expand for Example Usage**Examples**

Basic example:

```
OS_FIFO udpFifo;  
OSFifoInit(&udpFifo); // Initialize with default depth  
  
if (RegisterUDPFifo(1234, &udpFifo)) {  
    printf("UDP port 1234 registered successfully\n");  
} else {  
    printf("Failed to register port 1234\n");  
}
```

Example with custom FIFO depth:

```
OS_FIFO highTrafficFifo;
```

```

OSFifoInit(&highTrafficFifo, 20); // Deep queue for high traffic

if (RegisterUDPFifo(5000, &highTrafficFifo)) {
    printf("High-traffic port 5000 registered with 20 packet buffer\n");

    // Process packets in a loop
    UDPPacket packet;
    while (true) {
        if (ReadUDPPacket(&packet, 5000, TICKS_PER_SECOND)) {
            printf("Received %d bytes from %s:%d\n",
                packet.receivedDataLen,
                packet.sourceIpAddr.toString(),
                packet.sourcePort);
            ProcessPacket(&packet);
            ReleaseUDPPacket(packet);
        }
    }
}

```

Example with proper cleanup:

```

OS_FIFO myFifo;
OSFifoInit(&myFifo);

if (RegisterUDPFifo(8000, &myFifo)) {
    // ... do work ...

    // Clean shutdown
    UnregisterUDPFifo(8000, true); // Drain queued packets
    printf("Port 8000 unregistered\n");
}

```

Example - multiple ports:

```

OS_FIFO controlFifo, dataFifo, statusFifo;
OSFifoInit(&controlFifo);
OSFifoInit(&dataFifo);
OSFifoInit(&statusFifo);

// Register multiple ports for different purposes
RegisterUDPFifo(5000, &controlFifo); // Control channel
RegisterUDPFifo(5001, &dataFifo);    // Data channel
RegisterUDPFifo(5002, &statusFifo);  // Status channel

// Each port has its own FIFO and can be read independently
UDPPacket controlPacket;
if (ReadUDPPacket(&controlPacket, 5000, 0)) {
    HandleControl(&controlPacket);
    ReleaseUDPPacket(controlPacket);
}

```

Example - error handling:

```

OS_FIFO myFifo;
OSFifoInit(&myFifo);

const uint16_t PORT = 9000;

if (!RegisterUDPFifo(PORT, &myFifo)) {
    printf("ERROR: Failed to register port %d\n", PORT);
    printf("Possible causes:\n");
    printf(" - Port already in use\n");
    printf(" - Invalid FIFO pointer\n");
    printf(" - Out of system resources\n");
    return -1;
}

printf("Port %d ready to receive packets\n", PORT);

```

See also

- OSFifoInit() to initialize the FIFO structure
- ReadUDPPacket() to read packets from the FIFO
- ReleaseUDPPacket() to free packet buffers after processing
- UnregisterUDPFifo() to unregister the port and free resources
- RegisterUDPFifoVia() to register on a specific network interface
- RegisterUDPFifoWithNotify() to add notification callback
- RegisterEphemeralFifo() to auto-assign an ephemeral port

14.57.3.3 RegisterUDPFifoVia()

```

bool RegisterUDPFifoVia (
    uint16_t listenPort,

```

```

    OS_FIFO * pFifo,
    int interface)
#include <udp.h>

```

Register FIFO to receive UDP packets on specified port via specific network interface.

Similar to [RegisterUDPFifo\(\)](#), but allows explicit selection of which network interface to listen on. This is essential for multi-homed systems (devices with multiple network connections) where you need to receive UDP packets on a specific interface only, such as separating management traffic from data traffic, or implementing failover systems. Use this function when your device has multiple active network interfaces (Ethernet, WiFi, cellular, etc.) and you need to control which interface receives UDP traffic on a given port. The same port number can be registered on different interfaces simultaneously.

Parameters

<i>listenPort</i>	UDP port number to register (1-65535). Must not already be registered on the specified interface. The same port can be registered on different interfaces.
<i>pFifo</i>	Pointer to an initialized OS_FIFO structure for queuing received packets. Must be initialized with OSFifoInit() before calling. Recommended depth: 5-20 packets depending on traffic. The FIFO must remain valid until UnregisterUDPFifo() is called.
<i>interface</i>	Network interface number (0-based index): • 0 = First/primary interface (typically Ethernet) • 1 = Second interface (WiFi, secondary Ethernet, etc.) • 2+ = Additional interfaces if present The interface must be active and properly configured. Use GetInterfaceCount() to determine available interfaces.

Returns

true if port successfully registered on the specified interface, false on failure (e.g., port already in use on this interface, interface doesn't exist, invalid FIFO)

Note

Only packets arriving on the specified interface will be received

The same port can be registered on multiple interfaces with different FIFOs

Interface numbers are platform and configuration specific

The interface must remain active while the port is registered

Warning

Verify the interface exists and is active before registering

The FIFO must remain valid until [UnregisterUDPFifo\(\)](#) is called

If an interface goes down, registered ports on that interface stop receiving packets

Expand for Example Usage

Examples

Example (dual-homed device - management on eth0, data on eth1):

```

OS_FIFO mgmtFifo, dataFifo;
OSFifoInit(&mgmtFifo);
OSFifoInit(&dataFifo);

// Management interface (interface 0 - secure network)
if (RegisterUDPFifoVia(161, &mgmtFifo, 0)) { // SNMP on eth0
    printf("Management port 161 registered on interface 0\n");
}

// Data interface (interface 1 - production network)
if (RegisterUDPFifoVia(5000, &dataFifo, 1)) {
    printf("Data port 5000 registered on interface 1\n");
}

// Each interface receives independently

```



```

UDPPacket mgmtPacket;
if (ReadUDPPacket(&mgmtPacket, 161, 0)) {
    printf("Management packet from secure network\n");
    ReleaseUDPPacket(mgmtPacket);
}

```

Example (same port on multiple interfaces):

```

OS_FIFO primaryFifo, backupFifo;
OSFifoInit(&primaryFifo);
OSFifoInit(&backupFifo);

// Listen on port 8000 on both interfaces
RegisterUDPFifoVia(8000, &primaryFifo, 0); // Primary Ethernet
RegisterUDPFifoVia(8000, &backupFifo, 1);  // Backup WiFi

// Can distinguish which interface received packet by which FIFO
UDPPacket packet;
if (ReadUDPPacket(&packet, 8000, 0)) {
    // Check which FIFO had data to know which interface
    printf("Packet received on one of the interfaces\n");
    ReleaseUDPPacket(packet);
}

```

Example (interface availability checking):

```

int targetInterface = 1;

// Verify interface exists before attempting registration
if (GetInterfaceCount() > targetInterface) {
    if (InterfaceActive(targetInterface)) {
        OS_FIFO myFifo;
        OSFifoInit(&myFifo);

        if (RegisterUDPFifoVia(7000, &myFifo, targetInterface)) {
            printf("Port 7000 registered on interface %d\n", targetInterface);
        } else {
            printf("Failed to register on interface %d\n", targetInterface);
        }
    } else {
        printf("Interface %d exists but is not active\n", targetInterface);
    }
} else {
    printf("Interface %d does not exist\n", targetInterface);
}

```

Example (failover system):

```

OS_FIFO primaryFifo, failoverFifo;
OSFifoInit(&primaryFifo, 10);
OSFifoInit(&failoverFifo, 10);

const uint16_t SERVICE_PORT = 9000;
const int PRIMARY_IF = 0;
const int FAILOVER_IF = 1;

// Register on both interfaces
bool primaryOk = RegisterUDPFifoVia(SERVICE_PORT, &primaryFifo, PRIMARY_IF);
bool failoverOk = RegisterUDPFifoVia(SERVICE_PORT, &failoverFifo, FAILOVER_IF);

if (primaryOk && failoverOk) {
    printf("Failover system ready on port %d\n", SERVICE_PORT);

    while (true) {
        UDPPacket packet;

        // Check primary interface first
        if (ReadUDPPacket(&packet, SERVICE_PORT, 0)) {
            // Determine which interface by checking both FIFOs
            printf("Packet received - processing\n");
            ProcessPacket(&packet);
            ReleaseUDPPacket(packet);
        }

        OSTimeDly(TICKS_PER_SECOND / 10);
    }
}

```

Example (WiFi vs Ethernet separation):

```

#define ETH_INTERFACE 0
#define WIFI_INTERFACE 1

OS_FIFO ethernetFifo, wifiFifo;
OSFifoInit(&ethernetFifo);
OSFifoInit(&wifiFifo);

// Register different services on different interfaces
RegisterUDPFifoVia(80, &ethernetFifo, ETH_INTERFACE); // HTTP on wired only
RegisterUDPFifoVia(53, &wifiFifo, WIFI_INTERFACE);    // DNS on WiFi only

printf("HTTP on Ethernet, DNS on WiFi\n");

```

See also

[RegisterUDPFifo\(\)](#) for registration on all interfaces (simpler, most common)
[GetInterfaceCount\(\)](#) to determine number of available interfaces
[InterfaceActive\(\)](#) to check if an interface is active
[GetInterfaceIP\(\)](#) to get interface IP address
[UnregisterUDPFifo\(\)](#) to unregister the port
[RegisterUDPFifoWithNotifyVia\(\)](#) to add notification callback on specific interface

14.57.3.4 RegisterUDPFifoWithNotify()

```

bool RegisterUDPFifoWithNotify (
    uint16_t listenPort,
    OS_FIFO * pFifo,
    udp_data_notify * pNotifyFunction)
#include <udp.h>

```

Register UDP FIFO with callback notification on packet arrival.

Registers a UDP port for receiving packets and associates it with both a FIFO queue and a notification callback function. When a packet arrives, the callback is invoked immediately (typically from an interrupt context), allowing for prompt handling of time-sensitive data without polling.

The notification function is called before the packet is queued to the FIFO, enabling priority processing or filtering. This is useful for protocols requiring immediate response or for implementing custom packet filtering logic.

Parameters

<i>listenPort</i>	UDP port number to register (1-65535). Must not already be registered.
<i>pFifo</i>	Pointer to an OS_FIFO structure for queuing packets. The FIFO must be initialized before calling this function. Can be NULL if you only want notification callbacks.
<i>pNotifyFunction</i>	Pointer to callback function invoked when packet arrives. Function signature: void callback(uint16_t port, uint8_t* data, int len, IPADDR& srcAddr) Called in interrupt context - keep processing minimal and fast. Can be NULL if you only want FIFO queuing without notification.

Returns

true if port successfully registered with notification, false on failure (e.g., port already in use, invalid parameters)

Note

The notification callback is invoked from interrupt context - keep it fast and minimal

Do not perform blocking operations or lengthy processing in the callback

Both pFifo and pNotifyFunction can be provided, or just one (the other can be NULL)

Packets are queued to FIFO after the notification callback returns

Warning

The callback executes in interrupt context - avoid blocking calls, printf, or long operations

The FIFO and callback function must remain valid until [UnregisterUDPFifo\(\)](#) is called

Do not call [UnregisterUDPFifo\(\)](#) from within the notification callback

Expand for Example Usage**Examples**

Example (immediate response protocol):

```
OS_FIFO pingFifo;
OSFifoInit(&pingFifo);

// Notification callback - responds immediately to ping requests
void PingNotify(uint16_t port, uint8_t* data, int len, IPADDR& srcAddr) {
    if (len >= 4 && memcmp(data, "PING", 4) == 0) {
        // Immediate pong response
        SendUdpPacket(srcAddr, port, port, (uint8_t*)"PONG", 4);
    }
}

// Register with notification
if (RegisterUDPFifoWithNotify(9000, &pingFifo, PingNotify)) {
    printf("Ping responder registered on port 9000\n");
}

// Packets are also queued for later processing if needed
UDP_PACKET packet;
while (ReadUDPPacket(&packet, 9000, 0)) {
    printf("Logged ping from %s\n", packet.sourceIpAddr.toString());
    ReleaseUDPPacket(packet);
}
```

Example (packet filtering and statistics):

```
OS_FIFO datagramFifo;
OSFifoInit(&datagramFifo);

volatile uint32_t packetCount = 0;
volatile uint32_t filteredCount = 0;

// Notification for statistics and filtering
void DatagramNotify(uint16_t port, uint8_t* data, int len, IPADDR& srcAddr) {
    packetCount++;

    // Filter packets - could modify data or reject certain sources
    if (len < 10) {
        filteredCount++;
        // Packet still gets queued to FIFO
    }
}

RegisterUDPFifoWithNotify(5000, &datagramFifo, DatagramNotify);

// Main loop can check statistics
printf("Received: %lu, Filtered: %lu\n", packetCount, filteredCount);

Example (notification only, no FIFO):
// Just count packets, don't queue them
volatile uint32_t statsPackets = 0;

void StatsOnlyNotify(uint16_t port, uint8_t* data, int len, IPADDR& srcAddr) {
    statsPackets++;
}

// NULL FIFO - only notification callback
RegisterUDPFifoWithNotify(6000, NULL, StatsOnlyNotify);
```

See also

[RegisterUDPFifo\(\)](#) for registration without notification callback

[RegisterUDPFifoWithNotifyVia\(\)](#) to specify network interface

[UnregisterUDPFifo\(\)](#) to unregister the port

14.57.3.5 RegisterUDPFifoWithNotifyVia()

```
bool RegisterUDPFifoWithNotifyVia (
    uint16_t listenPort,
    OS_FIFO * pFifo,
    udp_data_notify * pNotifyFunction,
    int interface)

#include <udp.h>
```

Register UDP FIFO with callback notification on specific network interface.

Similar to [RegisterUDPFifoWithNotify\(\)](#), but allows explicit selection of which network interface to listen on. This is essential for multi-homed systems (devices with multiple network connections) where you need to receive UDP packets on a specific interface only.

Use this function when your device has multiple network interfaces (Ethernet, WiFi, etc.) and you need to isolate UDP traffic by interface. For example, listening for management traffic only on the Ethernet interface while ignoring

similar packets from WiFi.

Parameters

<i>listenPort</i>	UDP port number to register (1-65535). Must not already be registered on the specified interface.
<i>pFifo</i>	Pointer to an OS_FIFO structure for queuing packets. Must be initialized before calling. Can be NULL for notification-only.
<i>pNotifyFunction</i>	Pointer to callback function invoked when packet arrives. Function signature: void callback(uint16_t port, uint8_t* data, int len, IPADDR& srcAddr) Called in interrupt context. Can be NULL for FIFO-only operation.
<i>interface</i>	Network interface number (0-based): • 0 = First interface (typically primary Ethernet) • 1 = Second interface (WiFi, secondary Ethernet, etc.) • 2+ = Additional interfaces if present The interface must be active and configured.

Returns

true if port successfully registered on the specified interface, false on failure (e.g., port already in use on this interface, interface doesn't exist, invalid parameters)

Note

Only packets arriving on the specified interface will be received

The same port can be registered on different interfaces simultaneously

The notification callback is invoked from interrupt context - keep it minimal

Interface numbers are system-specific - check your platform documentation

Warning

The callback executes in interrupt context - avoid blocking operations

The FIFO and callback must remain valid until [UnregisterUDPFifo\(\)](#) is called

Verify the interface exists and is active before registering

Expand for Example Usage

Examples

Example (dual-homed device - different ports per interface):

```
OS_FIFO ethernetFifo, wifiFifo;
OSFifoInit(&ethernetFifo);
OSFifoInit(&wifiFifo);

// Listen on port 8000 on Ethernet interface only
if (RegisterUDPFifoWithNotifyVia(8000, &ethernetFifo, NULL, 0)) {
    printf("Registered on Ethernet (interface 0)\n");
}

// Listen on port 8000 on WiFi interface only (same port, different interface)
if (RegisterUDPFifoWithNotifyVia(8000, &wifiFifo, NULL, 1)) {
    printf("Registered on WiFi (interface 1)\n");
}

// Now we can distinguish packets by which FIFO they arrive on
```

Example (management vs. data network separation):

```
OS_FIFO mgmtFifo, dataFifo;
OSFifoInit(&mgmtFifo);
OSFifoInit(&dataFifo);

// Management traffic on dedicated Ethernet interface
RegisterUDPFifoWithNotifyVia(161, &mgmtFifo, NULL, 0); // SNMP on eth0

// High-speed data on separate interface
```

```

RegisterUDPFifoWithNotifyVia(5000, &dataFifo, NULL, 1); // Data on eth1

// Process management packets
UDPPacket mgmtPacket;
if (ReadUDPPacket(&mgmtPacket, 161, 0)) {
    printf("Management packet from secure network\n");
    ReleaseUDPPacket(mgmtPacket);
}

Example (interface-specific notification):
OS_FIFO primaryFifo, backupFifo;
OSFifoInit(&primaryFifo);
OSFifoInit(&backupFifo);

void PrimaryNotify(uint16_t port, uint8_t* data, int len, IPADDR& srcAddr) {
    // Log that primary interface received packet
    primaryPacketCount++;
}

void BackupNotify(uint16_t port, uint8_t* data, int len, IPADDR& srcAddr) {
    // Alert if backup interface is receiving traffic
    backupPacketCount++;
    TriggerFailoverAlert();
}

// Same port, different interfaces with different callbacks
RegisterUDPFifoWithNotifyVia(9000, &primaryFifo, PrimaryNotify, 0);
RegisterUDPFifoWithNotifyVia(9000, &backupFifo, BackupNotify, 1);

Example (checking interface availability first):
int targetInterface = 1;

// Verify interface exists and is active
if (GetInterfaceCount() > targetInterface && InterfaceActive(targetInterface)) {
    OS_FIFO myFifo;
    OSFifoInit(&myFifo);

    if (RegisterUDPFifoWithNotifyVia(7000, &myFifo, NULL, targetInterface)) {
        printf("Successfully registered on interface %d\n", targetInterface);
    } else {
        printf("Failed to register on interface %d\n", targetInterface);
    }
} else {
    printf("Interface %d not available\n", targetInterface);
}

```

See also

[RegisterUDPFifoWithNotify\(\)](#) for default interface registration
[RegisterUDPFifo\(\)](#) for basic registration without notification
[GetInterfaceCount\(\)](#) to determine available interfaces
[InterfaceActive\(\)](#) to check if an interface is active
[UnregisterUDPFifo\(\)](#) to unregister the port

14.57.3.6 UnregisterUDPFifo()

```

void UnregisterUDPFifo (
    uint16_t listenPort,
    bool drain = false)

```

```
#include <udp.h>
```

Unregister UDP FIFO and stop receiving packets on specified port.

Stops listening for UDP packets on the specified port and removes the associated FIFO queue from the system.

This function should be called when a UDP port is no longer needed to free system resources.

If the FIFO has queued packets waiting to be processed, the drain parameter controls whether these packets are discarded or left in the queue. Typically, you should drain the queue unless you have already processed all packets.

Parameters

<i>listenPort</i>	The UDP port number to unregister (must match a previously registered port)
<i>drain</i>	If true, frees all queued packets in the FIFO and releases their memory. If false, leaves queued packets in memory (default: false). In most cases, you should set this to true to prevent memory leaks.

Note

After calling this function, the port becomes available for other applications
 Any subsequent packets arriving on this port will be ignored until re-registered
 If drain is false and packets remain queued, you may have a memory leak

Warning

Always drain the FIFO (drain=true) unless you have a specific reason not to
 Unregistering a port that was never registered has no effect but is safe

Expand for Example Usage**Examples**

Example (typical cleanup):

```
// Registered earlier
RegisterUDPFifo(5000, 10); // Port 5000 with queue depth 10

// ... process packets ...
```

```
// Clean shutdown - drain all remaining packets
UnregisterUDPFifo(5000, true);
```

Example (graceful shutdown - process remaining packets first):

```
// Process all remaining packets before unregistering
UDPPacket packet;
while (ReadUDPPacket(&packet, 5000, 0)) { // 0 = non-blocking
    ProcessPacket(packet);
    ReleaseUDPPacket(packet);
}
```

```
// Now safe to unregister without drain since we processed everything
UnregisterUDPFifo(5000, false);
```

Example (application shutdown):

```
void ShutdownNetworking() {
    // Unregister all UDP ports and free resources
    UnregisterUDPFifo(5000, true); // Control port
    UnregisterUDPFifo(5001, true); // Data port
    UnregisterUDPFifo(5002, true); // Status port
}
```

See also

[RegisterUDPFifo\(\)](#) to register a UDP port for receiving
[ReadUDPPacket\(\)](#) to read packets from the FIFO
[ReleaseUDPPacket\(\)](#) to free individual packet buffers

14.57.4 UDP Error Codes**Macros**

- **#define UDP_ERR_NOSUCH_SOCKET (-1)**
Socket does not exist.
- **#define UDP_ERR_NOTOPEN_TO_WRITE (-2)**
Socket not open for write.
- **#define UDP_ERR_NOTOPEN_TO_READ (-3)**
Socket not open for read.

14.57.4.1 Detailed Description

These negative values are returned by UDP socket creation and operation functions to indicate specific error conditions. All successful operations return non-negative values (valid file descriptors ≥ 0 or byte counts ≥ 0).

Note

See detailed return code information, causes and solutions at:
[Programmers Guide UDP Error Codes](#)

14.57.5 UDP Packet API

Refer to UDP Packet Class and API: [UDPPacket](#)

14.57.6 UDP Socket API

BSD-style socket functions for UDP communication.

Functions

- int [CreateRxUdpSocket](#) (uint16_t listening_port)
Create receive-only UDP socket bound to specified port.
- int [CreateTxUdpSocket](#) (const [IPADDR](#) &send_to_addr, uint16_t remote_port, uint16_t local_port)
Create transmit-only UDP socket for sending to specified destination.
- int [CreateRxTxUdpSocket](#) (const [IPADDR](#) &send_to_addr, uint16_t send_to_remote_port, uint16_t local_port)
Create bidirectional UDP socket for both sending and receiving.
- int [sendto](#) (int sock, uint8_t what_to_send, int len_to_send, const [IPADDR](#) &to_addr, uint16_t remote_port)
Send UDP packet through socket to specified destination.
- int [sendtovia](#) (int sock, uint8_t what_to_send, int len_to_send, const [IPADDR](#) &to_addr, uint16_t remote_port, int intfnum)
Send via specific interface.
- int [recvfrom](#) (int sock, uint8_t buffer, int len, [IPADDR](#) *pAddr, uint16_t *pLocal_port, uint16_t *pRemote_port)
Receive UDP packet from socket with sender information.
- int [SendFragmentedUdpPacket](#) (const [IPADDR](#) &to, uint16_t source_port, uint16_t dest_port, uint8_t data, int length)
Send large UDP packet (>MTU) with IP fragmentation.

14.57.6.1 Detailed Description

BSD-style socket functions for UDP communication.

Provides familiar Berkeley Sockets (BSD) style interface for UDP networking. These functions offer an alternative to the [UDPPacket](#) class for developers who prefer traditional socket programming or are porting existing code.

14.57.6.1.1 Overview

The UDP Socket API provides three main operation types:

- **Receive-only sockets:** Listen for incoming packets on a specific port
- **Transmit-only sockets:** Send packets to a specific destination
- **Bidirectional sockets:** Combined receive and transmit capabilities

14.57.6.1.2 Key Concepts

14.57.6.1.2.1 Socket Types

1. Receive-Only Socket (Server/Listener)

- Created with [CreateRxUdpSocket\(\)](#)
- Binds to local port to receive packets
- Can receive from any sender
- Typical use: Servers, listeners, broadcast receivers

2. Transmit-Only Socket (Client/Sender)

- Created with [CreateTxUdpSocket\(\)](#)
- Optionally binds to specific local port
- Sends to predefined destination
- Typical use: Clients, sensors, periodic transmitters

3. Bidirectional Socket (Request/Response)

- Created with [CreateRxTxUdpSocket\(\)](#)
- Combines both receive and transmit
- Binds local port and sets remote endpoint
- Typical use: Client-server patterns, RPC-style communication

14.57.6.1.2.2 Port Number Selection

Local Port (Binding)

- **0**: System assigns ephemeral port (49152-65535) - recommended for clients
- **1-1023**: Well-known ports (e.g., 53=DNS, 123=NTP) - requires privileges
- **1024-49151**: Registered ports - typical for custom servers
- **49152-65535**: Dynamic/private ports - ephemeral port range

Remote Port (Destination)

- Specify the destination service port
- Must match the listening port on remote system

```
// Server pattern - fixed local port
int serverSock = CreateRxUdpSocket(8080);

// Client pattern - auto-assign local port
int clientSock = CreateTxUdpSocket(serverIP, 8080, 0);
```

14.57.6.1.3 Common Usage Patterns

Examples

Pattern 1: Simple UDP Server

```
#include <udp.h>
#include <init.h>

void UdpServerTask(void* pd) {
    int sock = CreateRxUdpSocket(5000);
    if (sock < 0) {
        printf("Failed to create socket: %d\r\n", sock);
        return;
    }

    uint8_t buffer[1024];
    IPADDR clientIP;
    uint16_t localPort, remotePort;

    while (1) {
        int n = recvfrom(sock, buffer, sizeof(buffer),
                        &clientIP, &localPort, &remotePort);

        if (n > 0) {
            printf("Received %d bytes from %s:%d\r\n",
                  n, AsciiIP(clientIP).c_str(), remotePort);

            // Echo back to sender
            sendto(sock, buffer, n, clientIP, remotePort);
        }
    }
}
```


Pattern 2: UDP Client (One-Way Sender)

```

void SendSensorData(IPADDR serverIP, int sensorValue) {
    // Create socket with auto-assigned local port (0)
    int sock = CreateTxUdpSocket(serverIP, 5000, 0);
    if (sock < 0) {
        printf("Socket creation failed\r\n");
        return;
    }

    char msg[64];
    int len = sprintf(msg, "SENSOR:%d,VALUE:%d",
                      GetDeviceID(), sensorValue);

    sendto(sock, (uint8_t*)msg, len, serverIP, 5000);

    close(sock);
}

```

Pattern 3: Request/Response (Bidirectional)

```

int QueryServer(IPADDR serverIP, const char* request,
                char* response, int maxLen) {
    // Create bidirectional socket
    int sock = CreateRxTxUdpSocket(serverIP, 6000, 0);
    if (sock < 0) return -1;

    // Send request
    sendto(sock, (uint8_t*)request, strlen(request), serverIP, 6000);

    // Wait for response with timeout
    fd_set readfds;
    struct timeval tv = {5, 0}; // 5 second timeout
    FD_ZERO(&readfds);
    FD_SET(sock, &readfds);

    if (select(sock + 1, &readfds, NULL, NULL, &tv) > 0) {
        IPADDR srcIP;
        uint16_t localPort, remotePort;
        int n = recvfrom(sock, (uint8_t*)response, maxLen,
                        &srcIP, &localPort, &remotePort);

        close(sock);
        return n;
    }

    close(sock);
    return 0; // Timeout
}

```

Pattern 4: Multi-Interface Routing

```

// Send via specific network interface (multi-homed device)
void SendViaInterface(IPADDR destIP, int interfaceNum) {
    int sock = CreateTxUdpSocketVia(destIP, 8080, 0, interfaceNum);
    if (sock >= 0) {
        uint8_t data[] = "Hello from specific interface";
        sendto(sock, data, sizeof(data), destIP, 8080, interfaceNum);
        close(sock);
    }
}

```

Pattern 5: Broadcast Discovery

```

void DiscoverDevices() {
    IPADDR broadcastAddr = AsciiToIp("255.255.255.255");
    int sock = CreateTxUdpSocket(broadcastAddr, 9999, 0);

    if (sock >= 0) {
        uint8_t discover[] = "DISCOVER:NetBurner";
        sendto(sock, discover, sizeof(discover), broadcastAddr, 9999);

        // Listen for responses
        uint8_t buf[256];
        IPADDR responderIP;
        uint16_t lp, rp;

        while (recvfrom(sock, buf, sizeof(buf), &responderIP, &lp, &rp) > 0) {
            printf("Found device at %s\r\n", AsciiIP(responderIP).c_str());
        }

        close(sock);
    }
}

```

Pattern 6: Using select() for Multiple Sockets

```
void MultiSocketHandler() {
    int sock1 = CreateRxUdpSocket(5000);
    int sock2 = CreateRxUdpSocket(5001);
    int maxfd = (sock1 > sock2) ? sock1 : sock2;

    fd_set readfds;
    uint8_t buffer[512];
    IPADDR srcIP;
    uint16_t lp, rp;

    while (1) {
        FD_ZERO(&readfds);
        FD_SET(sock1, &readfds);
        FD_SET(sock2, &readfds);

        if (select(maxfd + 1, &readfds, NULL, NULL, NULL) > 0) {
            if (FD_ISSET(sock1, &readfds)) {
                int n = recvfrom(sock1, buffer, sizeof(buffer),
                                &srcIP, &lp, &rp);
                printf("Port 5000: %d bytes\r\n", n);
            }

            if (FD_ISSET(sock2, &readfds)) {
                int n = recvfrom(sock2, buffer, sizeof(buffer),
                                &srcIP, &lp, &rp);
                printf("Port 5001: %d bytes\r\n", n);
            }
        }
    }
}
```

14.57.6.1.4 Error Handling

All socket creation functions return:

- `**>= 0**`: Valid socket file descriptor
- `**< 0**`: Error code (see `UDP_ERR_*` constants)

Common error codes:

- `UDP_ERR_NOSUCH_SOCKET (-1)`: Socket does not exist or was closed
- `UDP_ERR_NOTOPEN_TO_WRITE (-2)`: Socket not configured for sending
- `UDP_ERR_NOTOPEN_TO_READ (-3)`: Socket not configured for receiving

```
int sock = CreateRxUdpSocket(port);
if (sock < 0) {
    switch (sock) {
        case UDP_ERR_NOSUCH_SOCKET:
            printf("Socket does not exist\r\n");
            break;
        case UDP_ERR_NOTOPEN_TO_WRITE:
            printf("Cannot write to socket\r\n");
            break;
        case UDP_ERR_NOTOPEN_TO_READ:
            printf("Cannot read from socket\r\n");
            break;
        default:
            printf("Unknown error: %d\r\n", sock);
    }
    return;
}
```

14.57.6.1.5 Thread Safety

- Each socket should be used by a single task
- Multiple tasks can use different sockets
- Sockets are NOT thread-safe for concurrent access
- Use separate sockets per task for parallel operations

```
// Good: Each task uses its own socket
void Task1(void* pd) {
    int sock = CreateRxUdpSocket(5000);
    // ... use sock ...
}

void Task2(void* pd) {
    int sock = CreateRxUdpSocket(5001);
    // ... use sock ...
}
```

14.57.6.1.6 Performance Considerations

14.57.6.1.6.1 Buffer Sizing

- **Typical MTU:** 1500 bytes (Ethernet)
- **UDP header overhead:** 8 bytes
- **IP header overhead:** 20 bytes (IPv4), 40 bytes (IPv6)
- **Safe payload size:** 1472 bytes (IPv4), 1452 bytes (IPv6)
- **Larger packets:** Use [SendFragmentedUdpPacket\(\)](#) for automatic fragmentation

14.57.6.1.6.2 Optimization Tips

1. **Reuse sockets:** Create once, use many times
2. **Batch operations:** Send multiple packets before context switch
3. **Appropriate buffer sizes:** Match expected packet sizes
4. **Interface binding:** Reduce routing overhead in multi-homed systems
5. **Port reuse:** Close sockets when no longer needed

14.57.6.1.7 IPv4 vs IPv6

The API automatically handles both IPv4 and IPv6 based on IPADDR type:

```
#ifdef IPV6
// Dual-stack - works with both IPv4 and IPv6
IPADDR destIP = AsciiToIp("2001:db8::1"); // IPv6
// or
IPADDR destIP = AsciiToIp("192.168.1.100"); // IPv4-mapped

int sock = CreateTxUdpSocket(destIP, 8080, 0);
#else
// IPv4 only
IPADDR4 destIP = AsciiToIp4("192.168.1.100");
int sock = CreateTxUdpSocket(destIP, 8080, 0);
#endif
```

14.57.6.1.8 Multi-Homed Systems

For devices with multiple network interfaces:

14.57.6.1.8.1 Interface Number (Preferred)

```
// Send via interface 1 (second Ethernet, Wi-Fi, etc.)
int sock = CreateTxUdpSocketVia(destIP, 8080, 0, 1);
sendtovia(sock, data, len, destIP, 8080, 1);
```

14.57.6.1.8.2 Interface IP Address (Alternative)

```
// Send via specific interface IP
IPADDR myInterfaceIP = GetInterfaceIP(1);
int sock = CreateTxUdpSocketVia(destIP, 8080, 0, myInterfaceIP);
```

14.57.6.1.9 Comparison with UDPPacket Class

Feature	BSD Sockets	UDPPacket Class
API Style	Familiar C functions	C++ object-oriented
Memory Management	Manual	Automatic (RAII)
Buffer Access	Copy-based	Zero-copy available
Multi-destination	Multiple sends	SendAndKeep() support
Integration	select() / poll()	OS_FIFO queues
Portability	Standard POSIX	NetBurner-specific
Best For	Porting code, C code	New C++ projects

14.57.6.1.9.1 When to Use BSD Sockets:

- Porting existing socket-based code
- Writing in C (not C++)
- Need [select\(\)](#)/[poll\(\)](#) integration
- Familiar with Berkeley sockets
- Simple send/receive patterns

14.57.6.1.9.2 When to Use UDPPacket:

- New C++ development
- Need automatic memory management
- Direct buffer manipulation
- Multiple destination sends
- OS_FIFO-based architectures

14.57.6.1.10 Advanced Features

14.57.6.1.10.1 Large Packet Transmission (Fragmentation)

For packets exceeding MTU, use [SendFragmentedUdpPacket\(\)](#):

```
uint8_t largeData[5000];
// ... fill largeData ...

int result = SendFragmentedUdpPacket(
    destIP,
    0,      // source port (0 = auto-assign)
    8080,   // destination port
    largeData,
    sizeof(largeData)
);

if (result == sizeof(largeData)) {
    printf("Successfully sent %d bytes\r\n", result);
} else {
    printf("Send failed or partial: %d/%d bytes\r\n",
        result, sizeof(largeData));
}
```

Note: Fragmentation should be avoided when possible due to:

- Increased packet loss risk (one fragment lost = entire packet lost)
- Additional processing overhead
- Potential firewall/router issues
- Better to design protocol with smaller messages when feasible

14.57.6.1.10.2 Socket Lifecycle Management

```
// Create socket
int sock = CreateRxTxUdpSocket(serverIP, 8080, 0);
if (sock < 0) {
    // Handle error
    return;
}

// Use socket for communication
// ...

// Close socket when done
close(sock);

// Socket descriptor is now invalid
```

14.57.6.1.11 Common Pitfalls

14.57.6.1.11.1 [NO] Not checking return values

```
int sock = CreateRxUdpSocket(5000);
recvfrom(sock, buf, len, &ip, &lp, &rp); // BUG: sock might be negative!
```

14.57.6.1.11.2 [YES] Always validate

```
int sock = CreateRxUdpSocket(5000);
if (sock < 0) {
    printf("Socket creation failed: %d\r\n", sock);
    return;
}
recvfrom(sock, buf, len, &ip, &lp, &rp);
```

14.57.6.1.11.3 [NO] Buffer overflow

```
uint8_t buf[100];
recvfrom(sock, buf, 1000, &ip, &lp, &rp); // BUG: Buffer too small!
```

14.57.6.1.11.4 [YES] Match buffer size

```
uint8_t buf[1500];
int n = recvfrom(sock, buf, sizeof(buf), &ip, &lp, &rp);
if (n > 0 && n < sizeof(buf)) {
    // Process n bytes safely
}
```

14.57.6.1.11.5 [NO] Ignoring recvfrom() return value

```
uint8_t buf[512];
recvfrom(sock, buf, sizeof(buf), &ip, &lp, &rp);
printf("%s", buf); // BUG: Might print garbage or crash!
```

14.57.6.1.11.6 [YES] Check bytes received

```
uint8_t buf[512];
int n = recvfrom(sock, buf, sizeof(buf), &ip, &lp, &rp);
if (n > 0) {
    buf[n] = '\0'; // Safe null-termination
    printf("%s", buf);
}
```

14.57.6.1.11.7 [NO] Port conflicts

```
int sock1 = CreateRxUdpSocket(8080);
int sock2 = CreateRxUdpSocket(8080); // BUG: Port already in use!
```

14.57.6.1.11.8 [YES] Unique ports or reuse pattern

```
int sock1 = CreateRxUdpSocket(8080);
// ... use sock1 ...
close(sock1);

// Now safe to reuse port
int sock2 = CreateRxUdpSocket(8080);
```

14.57.6.1.12 Debugging Tips

1. **Enable diagnostics:** Call [EnableSystemDiagnostics\(\)](#) in UserMain()
2. **Check socket descriptor:** Valid sockets are ≥ 0
3. **Verify network connectivity:** Ensure interface is up and configured
4. **Monitor packet capture:** Use Wireshark to inspect actual traffic
5. **Test with known tools:** Use netcat (nc) for quick testing: ``bash

14.57.6.2 Send UDP packet from command line

```
echo "test" | nc -u 192.168.1.100 8080
```

14.57.6.3 Listen for UDP packets

```
nc -ul 8080 ``
```

1. **Check firewall rules:** Ensure UDP traffic is not blocked
2. **Validate addresses:** Use [AsciiIP\(\)](#) to print addresses for debugging

14.57.6.3.1 Related Topics

- [UDPPacket Class](#) - Object-oriented UDP interface
- [TCP Sockets](#) - For reliable, connection-oriented communication
- [IP Addressing](#) - IPADDR types and conversion functions
- File descriptor operations: [select\(\)](#), [close\(\)](#)

14.57.6.4 Function Documentation

14.57.6.4.1 CreateRxTxUdpSocket()

```
int CreateRxTxUdpSocket (
    const IPADDR & send_to_addr,
    uint16_t send_to_remote_port,
    uint16_t local_port) [inline]
```

```
#include <udp.h>
```

Create bidirectional UDP socket for both sending and receiving.

Creates a UDP socket that can both send packets to a specified destination and receive packets on a local port. This combines the functionality of [CreateTxUdpSocket\(\)](#) and [CreateRxUdpSocket\(\)](#) into a single socket, ideal for request/response patterns and client-server communication.

Socket Characteristics:

- Can send to pre-configured destination
- Can receive on bound local port
- Single socket for bidirectional communication
- Efficient for request/response protocols

Common Use Cases:

- Client applications expecting replies (DNS queries, SNMP, RPC)
- Request/response protocols
- Duplex communication patterns
- Protocols requiring both send and receive on same endpoint

Parameters

<i>send_to_addr</i>	Destination IP address for sending packets Use <code>AsciiToIp()</code> or <code>AsciiToIp4()</code> to convert from string
<i>send_to_remote_port</i>	Destination port number for sending (1-65535)
<i>local_port</i>	Local port to bind for receiving (0-65535) • 0: Auto-assign ephemeral port (common for clients) • 1-65535: Specific port (required if server must reply to known port)

Returns

Socket file descriptor (≥ 0) on success, negative error code on failure:

- `UDP_ERR_NOSUCH_SOCKET` (-1): Socket creation failed
- `UDP_ERR_NOTOPEN_TO_WRITE` (-2): Socket not writable
- `UDP_ERR_NOTOPEN_TO_READ` (-3): Socket not readable

Note

Port Selection: Use 0 for `local_port` if server response port doesn't matter. Use specific port only when protocol requires clients to listen on fixed ports.

Response Handling: Responses will arrive on the `local_port`. Use `recvfrom()` to receive them. The remote source may be different from `send_to_addr`.

Warning

Thread Safety: Socket should be used by single task. Do not call `sendto()` and `recvfrom()` concurrently from multiple tasks on same socket.

Expand for Example Usage

Examples

Example 1: DNS Query (Request/Response Pattern)

```
#include <udp.h>

bool DnsQuery(const char* hostname, IPADDR* result) {
    IPADDR dnsServer = AsciiToIp("8.8.8.8");

    // Create bidirectional socket for DNS query and response
    int sock = CreateRxTxUdpSocket(dnsServer, 53, 0);
    if (sock < 0) return false;

    // Build DNS query packet
    uint8_t query[512];
    int queryLen = BuildDnsQuery(query, hostname);

    // Send query
    sendto(sock, query, queryLen, dnsServer, 53);

    // Wait for response with timeout
    fd_set readfds;
    struct timeval tv = {5, 0}; // 5 second timeout
    FD_ZERO(&readfds);
    FD_SET(sock, &readfds);

    if (select(sock + 1, &readfds, NULL, NULL, &tv) > 0) {
        uint8_t response[512];
        IPADDR srcIP;
        uint16_t lp, rp;
        int n = recvfrom(sock, response, sizeof(response),
                        &srcIP, &lp, &rp);

        if (n > 0) {
            *result = ParseDnsResponse(response, n);
            close(sock);
            return true;
        }
    }
}
```

```

    close(sock);
    return false; // Timeout or error
}

```

Example 2: Echo Client with Response

```

void EchoClientTest() {
    IPADDR serverIP = AsciiToIp("192.168.1.100");

    // Create bidirectional socket
    int sock = CreateRxTxUdpSocket(serverIP, 5000, 0);
    if (sock < 0) {
        printf("Socket creation failed\r\n");
        return;
    }

    // Send message
    const char* msg = "Hello, Server!";
    sendto(sock, (uint8_t*)msg, strlen(msg), serverIP, 5000);
    printf("Sent: %s\r\n", msg);

    // Wait for echo response
    uint8_t buf[256];
    IPADDR respIP;
    uint16_t lp, rp;

    int n = recvfrom(sock, buf, sizeof(buf), &respIP, &lp, &rp);
    if (n > 0) {
        buf[n] = '\0';
        printf("Received echo: %s\r\n", buf);
    }

    close(sock);
}

```

Example 3: Persistent Connection Pattern

```

void RequestResponseLoop() {
    IPADDR serverIP = AsciiToIp("10.0.0.50");

    // Create persistent bidirectional socket
    int sock = CreateRxTxUdpSocket(serverIP, 6000, 0);
    if (sock < 0) return;

    uint8_t request[64];
    uint8_t response[256];
    IPADDR respIP;
    uint16_t lp, rp;

    for (int i = 0; i < 10; i++) {
        // Send request
        int reqLen = sprintf((char*)request, "REQ:%d", i);
        sendto(sock, request, reqLen, serverIP, 6000);

        // Wait for response
        int n = recvfrom(sock, response, sizeof(response),
                        &respIP, &lp, &rp);

        if (n > 0) {
            response[n] = '\0';
            printf("Response %d: %s\r\n", i, response);
        }

        OSTimeDly(TICKS_PER_SECOND);
    }

    close(sock);
}

```

See also

[CreateTxUdpSocket\(\)](#) - For send-only sockets (one-way communication)

[CreateRxUdpSocket\(\)](#) - For receive-only sockets (servers)

[CreateRxTxUdpSocketVia\(\)](#) - For multi-interface routing

[sendto\(\)](#) - Send data through socket

[recvfrom\(\)](#) - Receive data from socket

[select\(\)](#) - Wait for socket to be ready with timeout

[close\(\)](#) - Close socket when finished

14.57.6.4.2 CreateRxUdpSocket()

```

int CreateRxUdpSocket (

```



```
uint16_t listening_port)
#include <udp.h>
```

Create receive-only UDP socket bound to specified port.

Creates a UDP socket configured only for receiving packets. The socket is bound to the specified local port and can receive UDP packets from any source. This is the standard function for creating UDP server/listener sockets.

Socket Characteristics:

- Bound to specified local port
- Receives from any source IP/port
- Cannot send packets (receive-only)
- Suitable for servers and broadcast receivers

Common Use Cases:

- UDP servers listening for client requests
- Broadcast/multicast receivers
- Service discovery listeners
- Logging/monitoring receivers

Parameters

<i>listening_port</i>	Local port number to bind (1-65535) • Well-known ports (1-1023): Standard services (use with caution) • Registered ports (1024-49151): Custom applications (recommended) • Dynamic ports (49152-65535): Temporary/ephemeral use

Returns

Socket file descriptor (≥ 0) on success, negative error code on failure:

- `UDP_ERR_NOSUCH_SOCKET` (-1): Socket creation failed
- `UDP_ERR_NOTOPEN_TO_READ` (-3): Internal error - socket not readable

Note

Port Availability: The port must not already be in use by another socket. Use [UnregisterUDPFifo\(\)](#) or [close\(\)](#) existing sockets before reusing ports.

Thread Safety: The returned socket should only be used by a single task. For multi-task reception, create separate sockets on different ports or use the [UDPPacket](#) class with `OS_FIFO` for thread-safe queue-based reception.

Warning

Error Checking Required: Always check return value before using socket. Attempting to use a negative file descriptor will cause undefined behavior.

Expand for Example Usage

Examples

Example 1: Simple UDP Server

```
#include <udp.h>
#include <init.h>

void UdpServerTask(void* pd) {
    // Create receive-only socket on port 8080
    int sock = CreateRxUdpSocket(8080);
    if (sock < 0) {
        printf("Failed to create socket: error %d\r\n", sock);
        return;
    }

    printf("UDP server listening on port 8080\r\n");

    uint8_t buffer[1024];
    IPADDR clientIP;
    uint16_t localPort, remotePort;

    while (1) {
        // Wait for incoming packet
        int n = recvfrom(sock, buffer, sizeof(buffer),
                        &clientIP, &localPort, &remotePort);

        if (n > 0) {
            buffer[n] = '\0'; // Null-terminate for printing
            printf("Received %d bytes from %s:%d: %s\r\n",
                  n, AsciiIP(clientIP).c_str(), remotePort, buffer);
        }

        // Clean up (typically never reached in server loop)
        close(sock);
    }
}
```

Example 2: Echo Server

```
void UdpEchoServer() {
    int sock = CreateRxUdpSocket(5000);
    if (sock < 0) return;

    uint8_t buf[512];
    IPADDR srcIP;
    uint16_t lp, rp;

    while (1) {
        int n = recvfrom(sock, buf, sizeof(buf), &srcIP, &lp, &rp);
        if (n > 0) {
            // Note: Cannot echo back using this socket - it's receive-only!
            // Need CreateRxTxUdpSocket() for bidirectional operation
            printf("Echo server received: %d bytes\r\n", n);
        }
    }
}
```

Example 3: Broadcast Receiver

```
void BroadcastListener() {
    int sock = CreateRxUdpSocket(9999);
    if (sock < 0) {
        printf("Cannot create broadcast listener\r\n");
        return;
    }

    printf("Listening for broadcast packets on port 9999\r\n");

    uint8_t buf[256];
    IPADDR senderIP;
    uint16_t lp, rp;

    while (1) {
        int n = recvfrom(sock, buf, sizeof(buf), &senderIP, &lp, &rp);
        if (n > 0) {
            buf[n] = '\0';
            printf("Broadcast from %s: %s\r\n",
                  AsciiIP(senderIP).c_str(), buf);
        }
    }
}
```

See also

[CreateRxTxUdpSocket\(\)](#) - For bidirectional sockets (receive and send)

[CreateTxUdpSocket\(\)](#) - For transmit-only sockets

[CreateRxUdpSocketVia\(\)](#) - For multi-interface systems

[recvfrom\(\)](#) - Receive data from socket

[RegisterUDPFifo\(\)](#) - Alternative using [UDPPacket](#) class with OS_FIFO

[close\(\)](#) - Close socket when finished

14.57.6.4.3 CreateTxUdpSocket()

```
int CreateTxUdpSocket (
    const IPADDR & send_to_addr,
    uint16_t remote_port,
    uint16_t local_port) [inline]
#include <udp.h>
```

Create transmit-only UDP socket for sending to specified destination.

Creates a UDP socket configured only for sending packets to a specific remote address and port. The socket can optionally be bound to a specific local port, or use 0 to auto-assign an ephemeral port (recommended for most clients).

Socket Characteristics:

- Send-only (cannot receive packets)
- Pre-configured with destination address and port
- Optional local port binding (0=auto-assign)
- Suitable for client applications and sensors

Common Use Cases:

- UDP clients sending requests to servers
- Sensor data transmission
- Logging to remote syslog servers
- Periodic status updates to monitoring systems

Parameters

<i>send_to_addr</i>	Destination IP address (IPv4 or IPv6 depending on build config) Use AsciiToIp() or AsciiToIp4() to convert from string
<i>remote_port</i>	Destination port number (1-65535) Must match the listening port on remote system
<i>local_port</i>	Local port to bind (0-65535) • 0: Auto-assign ephemeral port (RECOMMENDED for clients) • 1-65535: Specific port (use only if required by protocol)

Returns

Socket file descriptor (≥ 0) on success, negative error code on failure:

- `UDP_ERR_NOSUCH_SOCKET` (-1): Socket creation failed
- `UDP_ERR_NOTOPEN_TO_WRITE` (-2): Internal error - socket not writable

Note

Local Port Recommendation: Use 0 for automatic port assignment unless your protocol specifically requires a fixed source port. Auto-assignment prevents port conflicts and is standard practice for UDP clients.

Destination Changes: This socket is bound to the specified destination. To send to different destinations, use [sendto\(\)](#) or create multiple sockets.

Warning

Send-Only: This socket cannot receive packets. For bidirectional communication (request/response patterns), use [CreateRxTxUdpSocket\(\)](#).

Expand for Example Usage**Examples****Example 1: Simple UDP Client**

```
#include <udp.h>

void SendMessage(const char* message) {
    IPADDR serverIP = AsciiToIp("192.168.1.100");

    // Create socket with auto-assigned local port (0)
    int sock = CreateTxUdpSocket(serverIP, 8080, 0);
    if (sock < 0) {
        printf("Failed to create socket\r\n");
        return;
    }

    // Send message
    int sent = sendto(sock, (uint8_t*)message, strlen(message),
                     serverIP, 8080);

    if (sent > 0) {
        printf("Sent %d bytes\r\n", sent);
    }

    close(sock);
}
```

Example 2: Periodic Sensor Data Transmission

```
void SensorTransmitTask(void* pd) {
    IPADDR serverIP = AsciiToIp("10.0.1.50");

    // Create persistent socket (reuse for efficiency)
    int sock = CreateTxUdpSocket(serverIP, 5000, 0);
    if (sock < 0) return;

    char msgBuf[64];

    while (1) {
        // Read sensor
        int temperature = ReadTemperatureSensor();
        int humidity = ReadHumiditySensor();

        // Format message
        int len = sprintf(msgBuf, "TEMP:%d,HUM:%d",
                          temperature, humidity);

        // Send to server
        sendto(sock, (uint8_t*)msgBuf, len, serverIP, 5000);

        // Wait before next transmission
        OSTimeDly(TICKS_PER_SECOND * 10); // Every 10 seconds
    }
}
```

Example 3: Syslog Client

```
void SendSyslogMessage(const char* msg, int severity) {
    IPADDR syslogServer = AsciiToIp("192.168.1.10");
    int sock = CreateTxUdpSocket(syslogServer, 514, 0); // Syslog port

    if (sock >= 0) {
        char syslogMsg[512];
        int len = sprintf(syslogMsg, "<%d>%s", severity, msg);
        sendto(sock, (uint8_t*)syslogMsg, len, syslogServer, 514);
        close(sock);
    }
}
```

Example 4: Multiple Destinations (Create Multiple Sockets)

```
void SendToMultipleServers() {
    IPADDR server1 = AsciiToIp("192.168.1.10");
    IPADDR server2 = AsciiToIp("192.168.1.20");

    int sock1 = CreateTxUdpSocket(server1, 8080, 0);
    int sock2 = CreateTxUdpSocket(server2, 8080, 0);

    uint8_t data[] = "Status update";

    if (sock1 >= 0) {
        sendto(sock1, data, sizeof(data), server1, 8080);
        close(sock1);
    }

    if (sock2 >= 0) {
```

```

        sendto(sock2, data, sizeof(data), server2, 8080);
        close(sock2);
    }
}

```

See also

[CreateRxTxUdpSocket\(\)](#) - For bidirectional communication (send and receive)

[CreateRxUdpSocket\(\)](#) - For receive-only sockets (servers)

[CreateTxUdpSocketVia\(\)](#) - For multi-interface routing

[sendto\(\)](#) - Send data through socket

[close\(\)](#) - Close socket when finished

14.57.6.4.4 recvfrom()

```

int recvfrom (
    int sock,
    uint8_t buffer,
    int len,
    IPADDR * pAddr,
    uint16_t * pLocal_port,
    uint16_t * pRemote_port) [inline]

```

#include <udp.h>

Receive UDP packet from socket with sender information.

Receives a UDP datagram from the specified socket and returns the sender's IP address and port. This is the standard BSD socket function for receiving UDP data. Works with sockets created by [CreateRxUdpSocket\(\)](#) or [CreateRxTxUdpSocket\(\)](#). Blocks until data arrives.

Parameters

<i>sock</i>	Socket file descriptor from CreateRxUdpSocket() or CreateRxTxUdpSocket()
<i>buffer</i>	Destination buffer for received data
<i>len</i>	Maximum bytes to read (buffer size, recommend 1500+ bytes)
<i>pAddr</i>	[OUT] Sender's IP address (can be NULL)
<i>pLocal_port</i>	[OUT] Local port packet arrived on (can be NULL)
<i>pRemote_port</i>	[OUT] Sender's port number (can be NULL)

Returns

Bytes received (0 to len), or negative error code:

- UDP_ERR_NOTOPEN_TO_READ (-3): Socket not configured for receiving
- UDP_ERR_NOSUCH_SOCKET (-1): Invalid socket

Note

Blocking: Waits indefinitely for data. Use [select\(\)](#) for timeout.

Datagram Boundaries: Each call returns one complete datagram.

Truncation: If datagram > buffer, excess data is discarded.

Warning

Buffer Size: Ensure buffer has at least len bytes.

Example: Echo Server

```
int sock = CreateRxUdpSocket(5000);
uint8_t buf[1500];
IPADDR clientIP;
uint16_t lp, rp;

while (1) {
    int n = recvfrom(sock, buf, sizeof(buf), &clientIP, &lp, &rp);
    if (n > 0) {
        printf("Received %d bytes from %s:%d\r\n",
            n, AsciiIP(clientIP).c_str(), rp);
        sendto(sock, buf, n, clientIP, rp); // Echo back
    }
}
```

See also

[CreateRxUdpSocket\(\)](#), [CreateRxTxUdpSocket\(\)](#), [sendto\(\)](#), [select\(\)](#)

14.57.6.4.5 SendFragmentedUdpPacket()

```
int SendFragmentedUdpPacket (
    const IPADDR & to,
    uint16_t source_port,
    uint16_t dest_port,
    uint8_t data,
    int length) [inline]
```

#include <udp.h>

Send large UDP packet (>MTU) with IP fragmentation.

Sends a UDP packet that exceeds the Maximum Transmission Unit (MTU) by automatically fragmenting it at the IP layer. The function handles both IPv4 and IPv6 addressing and reassembly is handled by the receiving host.

This is useful for sending large datagrams without implementing application-level fragmentation. However, note that IP fragmentation can reduce reliability since loss of any fragment requires retransmission of the entire datagram.

Parameters

<i>to</i>	Destination IP address (IPv4 or IPv6)
<i>source_port</i>	Source UDP port number. Use 0 for automatic port assignment (recommended)
<i>dest_port</i>	Destination UDP port number
<i>data</i>	Pointer to data buffer to send
<i>length</i>	Length of data in bytes (can exceed MTU)

Returns

Number of bytes sent on success, 0 on failure

Note

IP fragmentation may be blocked by some firewalls and routers

For maximum reliability, consider keeping packets under MTU size (typically 1500 bytes for Ethernet)

This function wraps `SendFragmentedUdpPacket6()` which handles both IPv4 and IPv6

Warning

Fragmented packets are more susceptible to loss and can impact performance

Example:

```
IPADDR destination = AsciiToIp("192.168.1.100");
uint8_t largeData[8000]; // Larger than typical MTU
// ... fill largeData ...

int result = SendFragmentedUdpPacket(destination, 0, 5000, largeData, sizeof(largeData));
if (result > 0) {
    printf("Sent %d bytes\n", result);
} else {
    printf("Send failed\n");
}
```

See also

[SendFragmentedUdpPacket6\(\)](#)

[SendUdpPacket\(\)](#) for non-fragmented sending

14.57.6.4.6 sendto()

```
int sendto (
    int sock,
    puint8_t what_to_send,
    int len_to_send,
    const IPADDR & to_addr,
    uint16_t remote_port) [inline]
```

```
#include <udp.h>
```

Send UDP packet through socket to specified destination.

Transmits a UDP datagram through the specified socket to the given IP address and port. This is the standard BSD socket function for sending UDP data. Works with sockets created by [CreateTxUdpSocket\(\)](#) or [CreateRxTxUdpSocket\(\)](#).

Operation:

- Sends datagram immediately (non-blocking)
- No connection establishment required
- No delivery guarantee (UDP is unreliable)
- Each call creates a new independent datagram

Parameters

<i>sock</i>	Socket file descriptor from CreateTxUdpSocket() or CreateRxTxUdpSocket() Must be a valid socket (≥ 0) or behavior is undefined
<i>what_to_send</i>	Pointer to data buffer to transmit Buffer content is copied during transmission
<i>len_to_send</i>	Number of bytes to send from buffer • Maximum practical size: 1472 bytes (IPv4) or 1452 bytes (IPv6) • Larger sizes may cause IP fragmentation (avoid if possible) • Use SendFragmentedUdpPacket() for data $>$ MTU
<i>to_addr</i>	Destination IP address (IPv4 or IPv6 based on build configuration) Use AsciiToIp() or AsciiToIp4() to convert from string format
<i>remote_port</i>	Destination port number (1-65535) Must match listening port on remote system

Returns

Number of bytes successfully queued for transmission (should equal `len_to_send`), or negative error code on failure:

- `UDP_ERR_NOTOPEN_TO_WRITE` (-2): Socket not configured for sending
- `UDP_ERR_NOSUCH_SOCKET` (-1): Invalid socket descriptor
- Other negative values: System-specific errors

Note

Non-Blocking: This function returns immediately after queuing data. The datagram may not have been physically transmitted yet.

No Delivery Confirmation: UDP provides no acknowledgment. Successfully sending does NOT mean the remote host received the packet. For reliable delivery, implement application-level acknowledgments or use TCP.

Maximum Transfer Unit (MTU): Ethernet MTU is typically 1500 bytes. After UDP (8 bytes) and IP headers (20-40 bytes), safe payload is ~1472 bytes. Exceeding this causes fragmentation, increasing packet loss risk.

Warning

Buffer Validity: Ensure `what_to_send` buffer remains valid and contains at least `len_to_send` bytes. Buffer overruns cause undefined behavior.

Socket Type: Using [sendto\(\)](#) on receive-only socket ([CreateRxUdpSocket\(\)](#)) will fail with `UDP_ERR_NOTOPEN_TO_WRITE`.

Expand for Example Usage

Examples

Example 1: Simple Send

```
IPADDR serverIP = AsciiToIp("192.168.1.100");
int sock = CreateTxUdpSocket(serverIP, 8080, 0);

if (sock >= 0) {
    const char* msg = "Hello, Server!";
    int sent = sendto(sock, (uint8_t*)msg, strlen(msg), serverIP, 8080);

    if (sent == strlen(msg)) {
        printf("Successfully sent %d bytes\r\n", sent);
    } else if (sent < 0) {
        printf("Send failed: error %d\r\n", sent);
    }
    close(sock);
}
```

Example 2: Send Binary Data

```
struct SensorData {
    uint32_t timestamp;
    int16_t temperature;
    int16_t humidity;
} data;

data.timestamp = Secs;
data.temperature = ReadTemperature();
data.humidity = ReadHumidity();

IPADDR collector = AsciiToIp("10.0.1.50");
int sock = CreateTxUdpSocket(collector, 5000, 0);

if (sock >= 0) {
    int result = sendto(sock, (uint8_t*)&data, sizeof(data),
                        collector, 5000);
    if (result != sizeof(data)) {
        printf("Warning: Send incomplete\r\n");
    }
    close(sock);
}
```

See also

- [CreateTxUdpSocket\(\)](#) - Create transmit socket
- [CreateRxTxUdpSocket\(\)](#) - Create bidirectional socket
- [sendtovia\(\)](#) - Send via specific network interface
- [recvfrom\(\)](#) - Receive UDP packets
- [SendFragmentedUdpPacket\(\)](#) - For sending large datagrams

14.58 User Authorization Manager

Topics

- [Authorization Responses](#)
- [Authorization Types](#)

Classes

- struct [UserAuthRecord](#)
A stored record of a user's authorization credentials. The value is hashed when saved so it can't be read directly. User's can currently only have one authorization record per user name.
- class [UserAuthManager](#)
The user authorization manager class allows application developers the ability to manage user authorization records. The can be loaded and saved to any storage space, including the config system or UserParams. Authorization values are hashed before being saved. Validation compares both the hash as well as the authorization type. Adding, updating, and removing records will automatically call the user devined save functions. For usage, please see the example found in `examples/SSH/sshServerUserAuth`.

Typedefs

- typedef int(* [SaveAuthRecordsFn](#)) (const [UserAuthRecord](#) *authRec)
User provided function for saving user authorization records. This allows the users to dictate where their authorization records are stored.
- typedef int(* [LoadAuthRecordsFn](#)) ([UserAuthRecord](#) *authRec)
User provided function for loading user authorization records. This allows the users to dictate where their authorization records are stored.

14.58.1 Detailed Description

The NetBurner User Authorization Manager

14.58.2 Typedef Documentation

14.58.2.1 LoadAuthRecordsFn

```
typedef int (* LoadAuthRecordsFn) (UserAuthRecord *authRec)
#include <UserAuthManager.h>
```

User provided function for loading user authorization records. This allows the users to dictate where their authorization records are stored.

Parameters

<i>authRec</i>	A pointer to an array of UserAuthRecords where the data will be loaded.

Return values

0	If the authorization records were loaded correctly.
!0	If there was an error loading the authorization records. Error codes are left to application developers to define.

14.58.2.2 SaveAuthRecordsFn

```
typedef int (* SaveAuthRecordsFn) (const UserAuthRecord *authRec)
#include <UserAuthManager.h>
```

User provided function for saving user authorization records. This allows the users to dictate where their authorization records are stored.

Parameters

<i>authRec</i>	A pointer to an array of UserAuthRecords that should be saved.

Return values

0	If the authorization records were saved correctly.
/0	If there was an error saving the authorization records. Error codes are left to application developers to define.

14.58.3 Authorization Responses

Enumerations

- enum `AuthResponse` : `int16_t` {
`eAuthSuccess` = 1 , `eAuthErrorUserExists` = -1 , `eAuthErrorUserDoesNotExist` = -2 , `eAuthErrorNoEmptyUserAuthRecords` = -3 ,
`eAuthErrorUnableToCreateHash` = -4 , `eAuthErrorAuthCheckFailed` = -5 , `eAuthErrorAuthTypeMismatch` = -6
, `eAuthErrorFailedRecordUpdate` = -7 ,
`eAuthErrorUnableToAddUser` = -8 , `eAuthErrorSaveFailed` = -9 }

Response return codes when checking for the authorization status of a user.

14.58.3.1 Detailed Description

14.58.3.2 Enumeration Type Documentation

14.58.3.2.1 AuthResponse

```
enum AuthResponse : int16_t
```

```
#include <UserAuthManager.h>
```

Response return codes when checking for the authorization status of a user.

Enumerator

<code>eAuthSuccess</code>	The authorization request or function was successful.
<code>eAuthErrorUserExists</code>	The user already exists.
<code>eAuthErrorUserDoesNotExist</code>	The user does not exist.
<code>eAuthErrorNoEmptyUserAuthRecords</code>	The authorization records are full.
<code>eAuthErrorUnableToCreateHash</code>	There was an error hashing the authorization value.
<code>eAuthErrorAuthCheckFailed</code>	The authorization check failed.
<code>eAuthErrorAuthTypeMismatch</code>	The authorization type didn't match the request.
<code>eAuthErrorFailedRecordUpdate</code>	The record update failed.
<code>eAuthErrorUnableToAddUser</code>	The user was not added successfully.
<code>eAuthErrorSaveFailed</code>	Unable to save user authorization records.

14.58.4 Authorization Types

Enumerations

- enum `AuthType` : `int8_t` { `eAuthTypeDefault` = 0 , `eAuthTypePassword` = 1 , `eAuthTypeKey` = 2 }

The types of authorization requests that are managed. These just indicate what the has value is, and don't provide any specific processing logic.

14.58.4.1 Detailed Description

14.58.4.2 Enumeration Type Documentation

14.58.4.2.1 AuthType

```
enum AuthType : int8_t
#include <UserAuthManager.h>
```

The types of authorization requests that are managed. These just indicate what the has value is, and don't provide any specific processing logic.

Enumerator

eAuthTypeDefault	Default value, should be considered an invalid type.
eAuthTypePassword	Password.
eAuthTypeKey	Key.

14.59 Utilities

Macros

- #define **DB_TCPIP** (1)
Debug output control flags for NetBurner system components.
- #define **DB_HTTP** (2)
HTTP debug messages.
- #define **DB_ETHER** (4)
Ethernet debug messages.
- #define **DB_RTOS** (8)
RTOS debug messages.
- #define **DB_BUFFER** (16)
Buffer debug messages.
- #define **DB_PPP** (32)
PPP debug messages.
- #define **DB_AU** (64)
Authentication debug messages.
- #define **DB_MAIL** (128)
Mail debug messages.
- #define **DB_IP** (256)
IP debug messages.
- #define **DB_TCPDATA** (512)
TCP data debug messages.
- #define **DB_SSL** (1024)
SSL debug messages.
- #define **DB_SNMP** (2048)
SNMP debug messages.
- #define **DB_IPV6_ND** (4096)
IPv6 Neighbor Discovery debug messages.
- #define **DB_IPV6_ICMP** (8192)
IPv6 ICMP debug messages.
- #define **DB_IPV6_ROUTE** (16384)
IPv6 routing debug messages.
- #define **DB_IPV6_FRAG** (32768)
IPv6 fragmentation debug messages.
- #define **DB_IPV6_ERR** (65536)
IPv6 error debug messages.

Functions

- [IPADDR AsciiToIp6](#) (const char *p)
Convert an ASCII IPv6 string to an IP address.
- void **ShowCounters** ()
Dump all system counters to stdio.
- void [sShowCounters](#) (char *buffer, int slen)
Dump all system counters to a memory buffer.
- void [FdShowRingData](#) (int fd, const uint8_t *ringBuf, uint32_t bufLen, uint32_t start, uint32_t end, const char *indent=NULL)
Dump part of a ring buffer to file descriptor.
- void [ShowRingData](#) (const uint8_t *ringBuf, uint32_t bufLen, uint32_t start, uint32_t end, const char *indent=NULL)
Dump part of a ring buffer to stdio.
- void [FdShowData](#) (int fd, const uint8_t *fromptr, uint16_t len)
Dump a block of data to file descriptor and show in ASCII and hex.
- void [ShowData](#) (const uint8_t *fromptr, uint16_t len)
Dump a block of data to stdio and show in ASCII and hex.
- void [ShowMac](#) (const [MACADR](#) *ma)
Dump a MAC address to stdio.
- void [fdShowMac](#) (int fd, const [MACADR](#) *ma)
Dump a MAC address to file descriptor.
- void [ShowMac](#) (const [MACADR](#) &ma)
Dump a MAC address to stdio (reference version)
- void [snShowMac](#) (char *buf, size_t maxlen, const [MACADR](#) *ma)
Dump a MAC address to character buffer.
- void [snShowMac](#) (char *buf, size_t maxlen, const [MACADR](#) &ma)
Dump a MAC address to character buffer (reference version)
- void [MacToId](#) ([MACADR](#) *ma, char *IDBuf)
Write 6 character ID string based on unique portion of MAC.
- void [outbyte](#) (char c)
Write out a single, unbuffered byte to stdio.
- void [print](#) (const char *str)
Write out a zero-terminated, unbuffered string.
- void [putnum](#) (int i)
Write out a hexadecimal, unbuffered number to stdio.
- void [putbytenum](#) (unsigned char c)
Write out a hexadecimal, unbuffered byte to stdio.
- [IPADDR4 AsciiToIp4](#) (const char *p)
Convert an ASCII IPv4 string to an IP address.
- [MACADR AsciiToMac](#) (const char *p)
Convert an ASCII MAC address string to a MAC address.
- BOOL [ValidIPv4](#) (const char *p)
Validate if a string contains a valid IPv4 address.
- void [ShowIP4](#) (const [IPADDR4](#) ia)
Dump an IPv4 address in ASCII IP string format to stdio.
- int [snShowIP4](#) (char *buf, size_t maxlen, const [IPADDR4](#) ia)
Dump an IPv4 address in ASCII IP string format to character buffer.
- void [ShowIP6](#) (const [IPADDR](#) &ia)
Dump an IPv6 address in ASCII IP string format to stdio.
- int [snShowIP6](#) (char *buf, size_t maxlen, const [IPADDR](#) &ia)

- Dump an IPv6 address in ASCII IP string format to character buffer.*

 - uint32_t `GetPreciseTime` (void)

Gets the time tick since system start at a higher resolution.
- unsigned long long `Get_msec` ()

Returns the number of milliseconds as a 64-bit value.
- int `kill` (int pid, int sig)

Send a signal to a process.
- void `_exit` (int i)

Terminate the calling process.
- int `_fini` (void)

Finalization function.
- unsigned char * `convertBinaryToHexAscii` (unsigned char *fromBufferPtr, unsigned int fromByteCount, unsigned char *toBufferPtr, unsigned int toByteCount)

Convert binary data to hexadecimal ASCII representation.
- unsigned char * `convertHexAsciiToBinary` (unsigned char *fromBufferPtr, unsigned int fromByteCount, unsigned char *toBufferPtr, unsigned int toByteCount)

Convert hexadecimal ASCII string to binary data.
- const char * `bfnstr` (const char *search, const char *target, size_t len)

Search for a C string in an arbitrary memory blob that may contain NULL chars.

14.59.1 Detailed Description

```
#include< utils.h>
```

14.59.2 Macro Definition Documentation

14.59.2.1 DB_TCPIP

```
#define DB_TCPIP (1)
```

```
#include <utils.h>
```

Debug output control flags for NetBurner system components.

These flags control diagnostic output from various NetBurner subsystems. Set the global DB_FLAGS variable to enable debug messages from specific components. Flags can be combined using bitwise OR to enable multiple debug outputs simultaneously.

Use requires the declaration of unsigned int DB_FLAGS = NB_SSL, usually in main.cpp. At least one flag or a value of 0 must be set in the DB_FLAGS declaration.

Note

Debug output should be disabled in production code as it adds overhead and can flood the console with messages.

```
#include <utils.h>
```

```
void UserMain(void *pd) {
    init();
```

```
    // Enable TCP/IP debug messages only
    unsigned int DB_FLAGS = DB_TCPIP | DB_SSL;
```

```
    // Now you'll see TCP/IP and SSL related debug output in the console
```

```
    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

TCP/IP debug messages

14.59.3 Function Documentation

14.59.3.1 `_exit()`

```
void _exit (
    int i)
#include <utils.h>
Terminate the calling process.
```

Parameters

<i>i</i>	Exit status

14.59.3.2 `_fini()`

```
int _fini (
    void )
#include <utils.h>
Finalization function.
```

Returns

Status code

14.59.3.3 `AsciiToIp4()`

```
IPADDR4 AsciiToIp4 (
    const char * p)
#include <utils.h>
Convert an ASCII IPv4 string to an IP address.
```

Parameters

<i>p</i>	Pointer to null-terminated ASCII IPv4 string

Returns

[IPADDR4](#) object containing the converted IPv4 address

14.59.3.4 `AsciiToIp6()`

```
IPADDR AsciiToIp6 (
    const char * p)
#include <utils.h>
Convert an ASCII IPv6 string to an IP address.
```

Parameters

<i>p</i>	Pointer to null-terminated ASCII IPv6 string

Returns

IPADDR object containing the converted IPv6 address

14.59.3.5 `AsciiToMac()`

```
MACADR AsciiToMac (
    const char * p)
#include <utils.h>
Convert an ASCII MAC address string to a MAC address.
```

Parameters

<i>p</i>	Pointer to null-terminated ASCII MAC address string

Returns

MACADR object containing the converted MAC address

14.59.3.6 bufnstr()

```
const char * bufnstr (
    const char * search,
    const char * target,
    size_t len)
```

```
#include <utils.h>
```

Search for a C string in an arbitrary memory blob that may contain NULL chars.

Parameters

<i>search</i>	Pointer to the buffer to search
<i>target</i>	The C string being searched for
<i>len</i>	Length of the search buffer

Returns

Pointer to the start of the target string in the search buffer, or NULL if target not found

14.59.3.7 convertBinaryToHexAscii()

```
unsigned char * convertBinaryToHexAscii (
    unsigned char * fromBufferPtr,
    unsigned int fromByteCount,
    unsigned char * toBufferPtr,
    unsigned int toByteCount)
```

```
#include <utils.h>
```

Convert binary data to hexadecimal ASCII representation.

Converts raw binary data into a hexadecimal ASCII string where each byte is represented by two hexadecimal characters (0-9, A-F). For example, the byte value 0xFF becomes the two-character string "FF".

This function is commonly used for:

- Displaying binary data in human-readable format
- Creating hexadecimal dumps of memory or packet contents
- Encoding binary data for text-based protocols
- Debugging network packets or cryptographic operations

Parameters

<i>fromBufferPtr</i>	Pointer to the source buffer containing binary data to convert
<i>fromByteCount</i>	Number of bytes to convert from the source buffer
<i>toBufferPtr</i>	Pointer to the destination buffer for hexadecimal ASCII output
<i>toByteCount</i>	Size of the destination buffer in bytes

Returns

Pointer to the next free byte in the destination buffer after conversion. Returns NULL if the destination buffer is too small.

Note

The destination buffer must be at least $(\text{fromByteCount} * 2)$ bytes to hold the complete hexadecimal representation, plus 1 byte if null-termination is desired.

The output is NOT automatically null-terminated. Add a null terminator manually if you need a C-string: `*result = '\0';`

Uses uppercase hexadecimal characters (A-F) not lowercase (a-f)

If $\text{toByteCount} < (\text{fromByteCount} * 2)$, conversion will stop when the destination buffer is full and return NULL to indicate buffer overflow

Warning

Always ensure the destination buffer is large enough to prevent buffer overflow: $\text{toByteCount} \geq (\text{fromByteCount} * 2)$

This function does not add any separators (spaces, colons, etc.) between bytes

Expand for Example Usage**Examples****Example 1: Basic Binary to Hex Conversion**

```
unsigned char binaryData[] = {0x48, 0x65, 0x6C, 0x6C, 0x6F}; // "Hello"
unsigned char hexBuffer[20]; // Need at least 10 bytes (5 * 2)

unsigned char *nextPtr = convertBinaryToHexAscii(
    binaryData,
    sizeof(binaryData),
    hexBuffer,
    sizeof(hexBuffer)
);

if (nextPtr != NULL) {
    *nextPtr = '\0'; // Null-terminate for printing
    iprintf("Hex: %s\r\n", hexBuffer); // Output: "48656C6C6F"
}
```

Example 2: Display MAC Address in Hex

```
void DisplayMACAddress(unsigned char *macAddr) {
    unsigned char hexBuffer[13]; // 6 bytes * 2 = 12, plus null terminator

    unsigned char *result = convertBinaryToHexAscii(
        macAddr,
        6, // MAC address is 6 bytes
        hexBuffer,
        sizeof(hexBuffer) - 1 // Leave room for null terminator
    );

    if (result != NULL) {
        *result = '\0';
        // Format with colons for readability
        iprintf("MAC: %c%c:%c%c:%c%c:%c%c:%c%c\r\n",
            hexBuffer[0], hexBuffer[1], // First byte
            hexBuffer[2], hexBuffer[3], // Second byte
            hexBuffer[4], hexBuffer[5], // Third byte
            hexBuffer[6], hexBuffer[7], // Fourth byte
            hexBuffer[8], hexBuffer[9], // Fifth byte
            hexBuffer[10], hexBuffer[11]); // Sixth byte
    }
}
```


Example 3: Hex Dump of Network Packet

```

void HexDumpPacket(unsigned char *packet, unsigned int length) {
    const int bytesPerLine = 16;
    unsigned char hexBuffer[64]; // 16 bytes * 2 = 32, with extra space

    iprintf("Packet hex dump (%u bytes):\r\n", length);

    for (unsigned int offset = 0; offset < length; offset += bytesPerLine) {
        unsigned int lineBytes = (length - offset < bytesPerLine) ?
                                (length - offset) : bytesPerLine;

        // Convert this line to hex
        unsigned char *result = convertBinaryToHexAscii(
            packet + offset,
            lineBytes,
            hexBuffer,
            sizeof(hexBuffer)
        );

        if (result != NULL) {
            *result = '\0';
            iprintf("%04X: %s\r\n", offset, hexBuffer);
        }
    }
}

```

Example 4: Convert Hash to Hex String

```

#include <md5.h>

void DisplayMD5Hash(const char *data, unsigned int length) {
    unsigned char hash[16]; // MD5 produces 16 bytes
    unsigned char hexHash[33]; // 16 * 2 = 32, plus null terminator

    // Calculate MD5 hash (example function)
    MD5_CTX ctx;
    MD5Init(&ctx);
    MD5Update(&ctx, (unsigned char *)data, length);
    MD5Final(hash, &ctx);

    // Convert to hex string
    unsigned char *result = convertBinaryToHexAscii(
        hash,
        16,
        hexHash,
        sizeof(hexHash) - 1
    );

    if (result != NULL) {
        *result = '\0';
        iprintf("MD5: %s\r\n", hexHash);
    }
}

```

Example 5: Error Handling for Buffer Overflow

```

bool ConvertToHexSafely(unsigned char *binary, unsigned int binLen,
                        char *hexStr, unsigned int hexLen) {
    // Check if destination buffer is large enough
    if (hexLen < (binLen * 2 + 1)) {
        iprintf("ERROR: Hex buffer too small. Need %u bytes, have %u\r\n",
            binLen * 2 + 1, hexLen);
        return false;
    }

    unsigned char *result = convertBinaryToHexAscii(
        binary,
        binLen,
        (unsigned char *)hexStr,
        hexLen - 1 // Leave room for null terminator
    );

    if (result == NULL) {
        iprintf("ERROR: Conversion failed\r\n");
        return false;
    }

    *result = '\0';
    return true;
}

```

```

}

// Usage
unsigned char data[] = {0xDE, 0xAD, 0xBE, 0xEF};
char hexStr[20];

if (ConvertToHexSafely(data, sizeof(data), hexStr, sizeof(hexStr))) {
    iprintf("Success: %s\r\n", hexStr); // Output: "DEADBEEF"
}

```

Example 6: Encode Binary Data for HTTP/JSON

```

void SendBinaryDataAsJSON(int sock, unsigned char *data, unsigned int length) {
    unsigned char hexBuffer[512];

    // Check if data fits
    if (length * 2 >= sizeof(hexBuffer)) {
        iprintf("ERROR: Data too large for hex encoding\r\n");
        return;
    }

    unsigned char *result = convertBinaryToHexAscii(
        data,
        length,
        hexBuffer,
        sizeof(hexBuffer) - 1
    );

    if (result != NULL) {
        *result = '\0';

        // Send as JSON
        NBString json;
        json.sprintf("{\"data\":\"%s\",\"length\":%u}", hexBuffer, length);

        writestring(sock, "HTTP/1.1 200 OK\r\n");
        writestring(sock, "Content-Type: application/json\r\n\r\n");
        writestring(sock, json.c_str());
    }
}

```

Example 7: Convert Serial Number to Hex String

```

void DisplayDeviceSerialNumber() {
    unsigned char serialNum[16]; // Example: 16-byte serial number
    unsigned char hexSerial[33]; // 16 * 2 = 32, plus null

    // Get serial number from device (example function)
    GetDeviceSerialNumber(serialNum, sizeof(serialNum));

    unsigned char *result = convertBinaryToHexAscii(
        serialNum,
        sizeof(serialNum),
        hexSerial,
        sizeof(hexSerial) - 1
    );

    if (result != NULL) {
        *result = '\0';
        iprintf("Device Serial: %s\r\n", hexSerial);
    }
}

```

Example 8: Format with Spaces Between Bytes

```

void ConvertToHexWithSpaces(unsigned char *binary, unsigned int length) {
    // Need 2 chars per byte + 1 space per byte (except last)
    unsigned char hexBuffer[256];
    unsigned char *ptr = hexBuffer;

    for (unsigned int i = 0; i < length; i++) {
        // Convert one byte at a time
        unsigned char *result = convertBinaryToHexAscii(
            &binary[i],
            1, // One byte
            ptr,
            3 // Two hex chars + space
        );

        if (result == NULL) break;
    }
}

```

```

    ptr = result;

    // Add space between bytes (except after last byte)
    if (i < length - 1) {
        *ptr++ = ' ';
    }

    *ptr = '\0';
    iprintf("Hex: %s\r\n", hexBuffer);
    // Example output: "48 65 6C 6C 6F"
}

```

See also

`convertHexAsciiToBinary()` - Reverse operation: hex ASCII to binary

14.59.3.8 `convertHexAsciiToBinary()`

```

unsigned char * convertHexAsciiToBinary (
    unsigned char * fromBufferPtr,
    unsigned int fromByteCount,
    unsigned char * toBufferPtr,
    unsigned int toByteCount)

```

#include <utils.h>

Convert hexadecimal ASCII string to binary data.

Converts a hexadecimal ASCII string (where each pair of characters represents one byte) into raw binary data. This is the reverse operation of [convertBinaryToHexAscii\(\)](#). For example, the string "FF" becomes the byte value 0xFF, and "48656C6C6F" becomes the five bytes {0x48, 0x65, 0x6C, 0x6C, 0x6F} which represents "Hello".

This function is commonly used for:

- Parsing hexadecimal data from configuration files or user input
- Decoding hex-encoded network protocols
- Converting hex strings from web forms or JSON
- Processing hexadecimal MAC addresses, hashes, or serial numbers

Parameters

<i>fromBufferPtr</i>	Pointer to the source buffer containing hexadecimal ASCII string
<i>fromByteCount</i>	Number of ASCII characters to convert (must be even for complete bytes)
<i>toBufferPtr</i>	Pointer to the destination buffer for binary output
<i>toByteCount</i>	Size of the destination buffer in bytes

Returns

Pointer to the next free byte in the destination buffer after conversion. Returns NULL if conversion fails (invalid hex characters, odd length, or buffer too small).

Note

The source string should contain only valid hexadecimal characters (0-9, A-F, a-f)

`fromByteCount` should be even (two hex chars = one binary byte). If odd, the last character may be ignored or cause an error.

The destination buffer must be at least (`fromByteCount / 2`) bytes in size

Both uppercase (A-F) and lowercase (a-f) hex characters are accepted

The source string does NOT need to be null-terminated; conversion uses `fromByteCount`

Warning

If fromByteCount is odd, behavior depends on implementation - may ignore last char or return NULL. Always provide an even number of hex characters.

Invalid hex characters (G-Z, special chars, etc.) will cause conversion to fail

Always ensure the destination buffer is large enough: toByteCount >= (fromByteCount / 2)

Expand for Example Usage**Examples****Example 1: Basic Hex String to Binary Conversion**

```
const char *hexString = "48656C6C6F"; // "Hello" in hex
unsigned char binaryData[10];

unsigned char *result = convertHexAsciiToBinary(
    (unsigned char *)hexString,
    strlen(hexString), // 10 characters = 5 bytes
    binaryData,
    sizeof(binaryData)
);

if (result != NULL) {
    unsigned int bytesConverted = result - binaryData;
    iprintf("Converted %u bytes\r\n", bytesConverted);
    iprintf("Text: %.*s\r\n", bytesConverted, binaryData); // Output: "Hello"
} else {
    iprintf("Conversion failed\r\n");
}
```

Example 2: Parse MAC Address from Hex String

```
bool ParseMACAddress(const char *hexMAC, unsigned char *macBytes) {
    // Remove colons/dashes if present, or use hex string directly
    // Example input: "001122334455" (12 hex chars = 6 bytes)

    if (strlen(hexMAC) != 12) {
        iprintf("ERROR: MAC address must be 12 hex characters\r\n");
        return false;
    }

    unsigned char *result = convertHexAsciiToBinary(
        (unsigned char *)hexMAC,
        12,
        macBytes,
        6
    );

    if (result == NULL) {
        iprintf("ERROR: Invalid hex characters in MAC address\r\n");
        return false;
    }

    iprintf("MAC: %02X:%02X:%02X:%02X:%02X:%02X\r\n",
        macBytes[0], macBytes[1], macBytes[2],
        macBytes[3], macBytes[4], macBytes[5]);
    return true;
}

// Usage
unsigned char mac[6];
if (ParseMACAddress("001122AABBCC", mac)) {
    // Use MAC address
}
```

Example 3: Decode Hex Data from HTTP POST

```
void HandleHexDataPost(int sock, const char *hexData) {
    unsigned char binaryData[256];
    unsigned int hexLen = strlen(hexData);

    // Validate hex string length
    if (hexLen % 2 != 0) {
        writestring(sock, "HTTP/1.1 400 Bad Request\r\n\r\n");
        writestring(sock, "Hex string must have even length");
        return;
    }

    if (hexLen / 2 > sizeof(binaryData)) {
```

```

        writestring(sock, "HTTP/1.1 413 Payload Too Large\r\n\r\n");
        return;
    }

    // Convert hex to binary
    unsigned char *result = convertHexAsciiToBinary(
        (unsigned char *)hexData,
        hexLen,
        binaryData,
        sizeof(binaryData)
    );

    if (result == NULL) {
        writestring(sock, "HTTP/1.1 400 Bad Request\r\n\r\n");
        writestring(sock, "Invalid hex characters");
        return;
    }

    unsigned int byteCount = result - binaryData;

    // Process binary data
    ProcessBinaryData(binaryData, byteCount);

    writestring(sock, "HTTP/1.1 200 OK\r\n\r\n");
    writestring(sock, "Data processed successfully");
}

```

Example 4: Parse Configuration Value in Hex

```

bool LoadEncryptionKey(const char *hexKey, unsigned char *keyBytes,
    unsigned int expectedKeySize) {
    unsigned int hexLen = strlen(hexKey);

    // Check if hex string represents correct key size
    if (hexLen != expectedKeySize * 2) {
        fprintf("ERROR: Key must be %u bytes (%u hex chars)\r\n",
            expectedKeySize, expectedKeySize * 2);
        return false;
    }

    unsigned char *result = convertHexAsciiToBinary(
        (unsigned char *)hexKey,
        hexLen,
        keyBytes,
        expectedKeySize
    );

    if (result == NULL) {
        fprintf("ERROR: Invalid hex characters in encryption key\r\n");
        return false;
    }

    fprintf("Encryption key loaded successfully\r\n");
    return true;
}

// Usage: Load 128-bit (16 byte) AES key
unsigned char aesKey[16];
if (LoadEncryptionKey("0123456789ABCDEF0123456789ABCDEF", aesKey, 16)) {
    // Use key for encryption
}

```

Example 5: Convert JSON Hex Field to Binary

```

#include <json_lexer.h>

bool ParseHexFromJSON(const char *jsonStr, unsigned char *output,
    unsigned int maxOutputSize) {
    ParsedJsonDataSet json(jsonStr);

    // Get hex string from JSON field
    const char *hexValue = json["data"].GetStr();
    if (hexValue == NULL) {
        fprintf("ERROR: 'data' field not found in JSON\r\n");
        return false;
    }

    unsigned int hexLen = strlen(hexValue);

    if (hexLen / 2 > maxOutputSize) {
        fprintf("ERROR: Hex data too large for buffer\r\n");
        return false;
    }
}

```

```

    unsigned char *result = convertHexAsciiToBinary(
        (unsigned char *)hexValue,
        hexLen,
        output,
        maxOutputSize
    );

    return (result != NULL);
}

// Usage with JSON: {"data":"DEADBEEF"}
unsigned char binaryData[256];
if (ParseHexFromJSON(jsonString, binaryData, sizeof(binaryData))) {
    // Process binary data
}

```

Example 6: Validate and Convert Hex String

```

bool IsValidHexString(const char *str) {
    // Check for even length
    unsigned int len = strlen(str);
    if (len % 2 != 0) return false;

    // Check all characters are valid hex
    for (unsigned int i = 0; i < len; i++) {
        char c = str[i];
        if (!((c >= '0' && c <= '9') ||
            (c >= 'A' && c <= 'F') ||
            (c >= 'a' && c <= 'f')))) {
            return false;
        }
    }
    return true;
}

bool SafeHexToBinary(const char *hexStr, unsigned char *binary,
                    unsigned int binarySize) {
    // Pre-validate before conversion
    if (!IsValidHexString(hexStr)) {
        iprintf("ERROR: Invalid hex string format\r\n");
        return false;
    }

    unsigned int hexLen = strlen(hexStr);
    if (hexLen / 2 > binarySize) {
        iprintf("ERROR: Output buffer too small\r\n");
        return false;
    }

    unsigned char *result = convertHexAsciiToBinary(
        (unsigned char *)hexStr,
        hexLen,
        binary,
        binarySize
    );

    return (result != NULL);
}

```

Example 7: Round-Trip Conversion Test

```

void TestHexConversions() {
    // Original binary data
    unsigned char original[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xCA, 0xFE};

    // Convert to hex
    unsigned char hexBuffer[20];
    unsigned char *hexResult = convertBinaryToHexAscii(
        original,
        sizeof(original),
        hexBuffer,
        sizeof(hexBuffer) - 1
    );

    if (hexResult == NULL) {
        iprintf("Hex conversion failed\r\n");
        return;
    }
    *hexResult = '\0';

    iprintf("Hex: %s\r\n", hexBuffer); // "DEADBEEFCAFE"
}

```

```

// Convert back to binary
unsigned char restored[10];
unsigned char *binResult = convertHexAsciiToBinary(
    hexBuffer,
    strlen((char *)hexBuffer),
    restored,
    sizeof(restored)
);

if (binResult == NULL) {
    iprintf("Binary conversion failed\r\n");
    return;
}

// Verify round-trip
if (memcmp(original, restored, sizeof(original)) == 0) {
    iprintf("Round-trip conversion successful!\r\n");
} else {
    iprintf("ERROR: Round-trip conversion mismatch\r\n");
}
}

```

Example 8: Parse Hex String with Mixed Case

```

void DemoMixedCaseHex() {
    // Both uppercase and lowercase hex chars are valid
    const char *mixedHex = "DeAdBeEf";
    unsigned char binary[4];

    unsigned char *result = convertHexAsciiToBinary(
        (unsigned char *)mixedHex,
        strlen(mixedHex),
        binary,
        sizeof(binary)
    );

    if (result != NULL) {
        iprintf("Converted: ");
        for (int i = 0; i < 4; i++) {
            iprintf("%02X ", binary[i]);
        }
        iprintf("\r\n"); // Output: "DE AD BE EF"
    }
}

```

Example 9: Parse Serial Number from User Input

```

bool SetDeviceSerialNumber(const char *hexSerial) {
    const unsigned int SERIAL_SIZE = 16; // 16 bytes = 128 bits
    unsigned char serialBytes[SERIAL_SIZE];

    // Validate input length
    if (strlen(hexSerial) != SERIAL_SIZE * 2) {
        iprintf("ERROR: Serial number must be %u hex characters\r\n",
            SERIAL_SIZE * 2);
        return false;
    }

    // Convert to binary
    unsigned char *result = convertHexAsciiToBinary(
        (unsigned char *)hexSerial,
        SERIAL_SIZE * 2,
        serialBytes,
        SERIAL_SIZE
    );

    if (result == NULL) {
        iprintf("ERROR: Invalid hex characters in serial number\r\n");
        return false;
    }

    // Store serial number (example function)
    StoreSerialNumberToFlash(serialBytes, SERIAL_SIZE);

    iprintf("Serial number set successfully\r\n");
    return true;
}

```

See also

[convertBinaryToHexAscii\(\)](#) - Reverse operation: binary to hex ASCII

14.59.3.9 FdShowData()

```
void FdShowData (
    int fd,
    const uint8_t * fromptr,
    uint16_t len)
#include <utils.h>
```

Dump a block of data to file descriptor and show in ASCII and hex.

Parameters

<i>fd</i>	File descriptor to write to
<i>fromptr</i>	Pointer to data to display
<i>len</i>	Number of bytes to display

14.59.3.10 fdShowMac()

```
void fdShowMac (
    int fd,
    const MACADDR * ma)
#include <utils.h>
```

Dump a MAC address to file descriptor.

Parameters

<i>fd</i>	File descriptor to write to
<i>ma</i>	Pointer to MACADDR structure to display

14.59.3.11 FdShowRingData()

```
void FdShowRingData (
    int fd,
    const uint8_t * ringBuf,
    uint32_t bufLen,
    uint32_t start,
    uint32_t end,
    const char * indent = NULL)
#include <utils.h>
```

Dump part of a ring buffer to file descriptor.

Parameters

<i>fd</i>	File descriptor to write to
<i>ringBuf</i>	Pointer to ring buffer data
<i>bufLen</i>	Total length of ring buffer
<i>start</i>	Start offset in ring buffer
<i>end</i>	End offset in ring buffer
<i>indent</i>	Optional indentation string for formatting (can be NULL)

14.59.3.12 Get_msec()

```
unsigned long long Get_msec ()
#include <utils.h>
```

Returns the number of milliseconds as a 64-bit value.

Return values

<i>Number</i>	of milliseconds

14.59.3.13 GetPreciseTime()

```
uint32_t GetPreciseTime (
    void ) [extern]
#include <utils.h>
```

Gets the time tick since system start at a higher resolution.

Resolution depends on the platform: 0.868-us for MOD5234/70, and 1.929-us for MOD5282

Returns

High resolution time tick as uint32_t

14.59.3.14 kill()

```
int kill (
    int pid,
    int sig)
#include <utils.h>
```

Send a signal to a process.

Parameters

<i>pid</i>	Process ID
<i>sig</i>	Signal number

Returns

0 on success, -1 on error

14.59.3.15 MacToID()

```
void MacToID (
    MACADR * ma,
    char * IDBuf)
#include <utils.h>
```

Write 6 character ID string based on unique portion of MAC.

Parameters

<i>ma</i>	Pointer to MACADR structure
<i>IDBuf</i>	Buffer to write 6-character ID string to (must be at least 7 bytes for null terminator)

14.59.3.16 outbyte()

```
void outbyte (
    char c)
#include <utils.h>
```

Write out a single, unbuffered byte to stdio.

Parameters

<i>c</i>	Character to output

14.59.3.17 print()

```
void print (
    const char * str)
#include <utils.h>
Write out a zero-terminated, unbuffered string.
```

Parameters

<i>str</i>	Pointer to null-terminated string to output

14.59.3.18 putbytenum()

```
void putbytenum (
    unsigned char c)
#include <utils.h>
Write out a hexadecimal, unbuffered byte to stdio.
```

Parameters

<i>c</i>	Byte value to output in hexadecimal format

14.59.3.19 putnum()

```
void putnum (
    int i)
#include <utils.h>
Write out a hexadecimal, unbuffered number to stdio.
```

Parameters

<i>i</i>	Integer value to output in hexadecimal format

14.59.3.20 ShowData()

```
void ShowData (
    const uint8_t * fromptr,
    uint16_t len)
#include <utils.h>
Dump a block of data to stdio and show in ASCII and hex.
```

Parameters

<i>fromptr</i>	Pointer to data to display
<i>len</i>	Number of bytes to display

14.59.3.21 ShowIP4()

```
void ShowIP4 (
    const IPADDR4 ia)
#include <utils.h>
Dump an IPv4 address in ASCII IP string format to stdio.
```

Parameters

<i>ia</i>	IPv4 address to display

14.59.3.22 ShowIP6()

```
void ShowIP6 (
    const IPADDR & ia) [inline]
#include <utils.h>
Dump an IPv6 address in ASCII IP string format to stdio.
```

Parameters

<i>ia</i>	IPv6 address to display

14.59.3.23 ShowMac() [1/2]

```
void ShowMac (
    const MACADR & ma) [inline]
#include <utils.h>
Dump a MAC address to stdio (reference version)
```

Parameters

<i>ma</i>	Reference to MACADR structure to display

14.59.3.24 ShowMac() [2/2]

```
void ShowMac (
    const MACADR * ma)
#include <utils.h>
Dump a MAC address to stdio.
```

Parameters

<i>ma</i>	Pointer to MACADR structure to display

14.59.3.25 ShowRingData()

```
void ShowRingData (
    const uint8_t * ringBuf,
    uint32_t bufLen,
    uint32_t start,
    uint32_t end,
    const char * indent = NULL)
#include <utils.h>
Dump part of a ring buffer to stdio.
```

Parameters

<i>ringBuf</i>	Pointer to ring buffer data
<i>bufLen</i>	Total length of ring buffer
<i>start</i>	Start offset in ring buffer

<i>end</i>	End offset in ring buffer
<i>indent</i>	Optional indentation string for formatting (can be NULL)

14.59.3.26 snShowIP4()

```
int snShowIP4 (
    char * buf,
    size_t maxlen,
    const IPADDR4 ia)
```

#include <utils.h>

Dump an IPv4 address in ASCII IP string format to character buffer.

Parameters

<i>buf</i>	Buffer to write IP address string to
<i>maxlen</i>	Maximum length of buffer
<i>ia</i>	IPv4 address to convert

Returns

Number of characters written to buffer

14.59.3.27 snShowIP6()

```
int snShowIP6 (
    char * buf,
    size_t maxlen,
    const IPADDR & ia) [inline]
```

#include <utils.h>

Dump an IPv6 address in ASCII IP string format to character buffer.

Parameters

<i>buf</i>	Buffer to write IP address string to
<i>maxlen</i>	Maximum length of buffer
<i>ia</i>	IPv6 address to convert

Returns

Number of characters written to buffer

14.59.3.28 snShowMac() [1/2]

```
void snShowMac (
    char * buf,
    size_t maxlen,
    const MACADDR & ma) [inline]
```

#include <utils.h>

Dump a MAC address to character buffer (reference version)

Parameters

<i>buf</i>	Buffer to write MAC address string to
<i>maxlen</i>	Maximum length of buffer
<i>ma</i>	Reference to MACADDR structure to convert

14.59.3.29 snShowMac() [2/2]

```
void snShowMac (
    char * buf,
    size_t maxlen,
    const MACADR * ma)
#include <utils.h>
Dump a MAC address to character buffer.
```

Parameters

<i>buf</i>	Buffer to write MAC address string to
<i>maxlen</i>	Maximum length of buffer
<i>ma</i>	Pointer to MACADR structure to convert

14.59.3.30 sShowCounters()

```
void sShowCounters (
    char * buffer,
    int slen)
#include <utils.h>
Dump all system counters to a memory buffer.
```

Parameters

<i>buffer</i>	Pointer to buffer to store counter information
<i>slen</i>	Size of the buffer in bytes

14.59.3.31 ValidIPv4()

```
BOOL ValidIPv4 (
    const char * p)
#include <utils.h>
Validate if a string contains a valid IPv4 address.
```

Parameters

<i>p</i>	Pointer to null-terminated string to validate

Returns

TRUE if string contains valid IPv4 address, FALSE otherwise

14.60 Web Client**Topics**

- [Web Client Error Codes](#)

Classes

- class [ParsedURI](#)
Parsed Uniform Resource Identifier Class (URI)
- class [buffer_object](#)
Base class for web client response buffers.

- class [SimpleBufferObject](#)
Class for web client response buffers.
- class [StringBufferObject](#)
Class for NBString-based web client response buffers.

Functions

- void [SetHttpDiag](#) (bool b)
Enable/disable Web Client HTTP diagnostics to the console port.
- void [SetHttpDiag](#) (int level)
Enable/disable Web Client HTTP diagnostics to the console port.
- int [DoMultipartStartHttp](#) (HttpMethod method, [ParsedURI](#) &TheUri, const char *separator, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#), uint32_t contentLength=0)
Initiates a multipart HTTP request using a pre-parsed URI and returns a connection file descriptor.
- int [DoMultipartStartHttp](#) (HttpMethod method, const char *pUrl, const char *separator, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#), uint32_t contentLength=0)
Initiates a multipart HTTP request using a URL string and returns a connection file descriptor.
- int [DoMultipartStartPost](#) ([ParsedURI](#) &TheUri, const char *separator, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#), uint32_t contentLength=0)
Initiates a multipart HTTP POST request using a pre-parsed URI and returns a connection file descriptor.
- int [DoMultipartStartPost](#) (const char *pUrl, const char *separator, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#), uint32_t contentLength=0)
Initiates a multipart HTTP POST request using a URL string and returns a connection file descriptor.
- void [DoMultipartItem](#) (int tcpfd, const char *Disposition, const char *separator, const unsigned char *data, int len)
Sends a multipart data item within an active multipart HTTP request.
- void [DoMultipartBoundary](#) (int tcpfd, const char *Disposition, const char *separator)
Sends a multipart boundary section within an active multipart HTTP request.
- bool [DoMultipartFinished](#) (int tcpfd, const char *separator, [buffer_object](#) &result_buffer, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))
Completes a multipart HTTP request and retrieves the server response in a buffer object.
- bool [DoMultipartFinished](#) (int tcpfd, const char *separator, [NBString](#) &result_string, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))
Completes a multipart HTTP request and retrieves the server response in an NBString. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. This is a convenience overload that finalizes a multipart HTTP request and stores the server response in an NBString reference. The function uses a StringBufferObject internally to handle the response data collection and forwards the call to the underlying DoMultipartFinished implementation. This provides string-based response handling for multipart operations where text-based responses are expected.
- bool [DoUrlEncodedFormHttp](#) (HttpMethod method, [ParsedURI](#) &TheUri, char *headers, char *form_data, [buffer_object](#) &result_buffer, uint16_t TIMEOUT_WAIT)
Executes an HTTP request with URL-encoded form data using a pre-parsed URI and stores the response in a buffer object.
- bool [DoUrlEncodedFormPost](#) ([ParsedURI](#) &TheUri, char *headers, char *form_data, [buffer_object](#) &result_buffer, uint16_t TIMEOUT_WAIT)
Executes an HTTP POST request with URL-encoded form data using a pre-parsed URI and stores the response in a buffer object.
- bool [DoUrlEncodedFormHttp](#) (HttpMethod method, [ParsedURI](#) &TheUri, char *headers, char *form_data, [NBString](#) &result_string, uint16_t TIMEOUT_WAIT)
Executes an HTTP request with URL-encoded form data using a pre-parsed URI and stores the response in an NBString. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. This is a convenience overload that performs an HTTP request with the specified method using URL-encoded form data and a pre-parsed URI object. The complete response body is stored in the provided NBString reference. This function uses a StringBufferObject internally to handle the response data collection and forwards the call to the underlying DoUrlEncodedFormHttp implementation. This provides flexibility for form-based HTTP operations with pre-parsed URIs and string-based response handling.

- **bool DoUrlEncodedFormPost** (**ParsedURI** &TheUri, char *headers, char *form_data, **NBString** &result_string, uint16_t TIMEOUT_WAIT)

*Executes an HTTP POST request with URL-encoded form data using a pre-parsed URI and stores the response in an **NBString**. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.*

*This is a convenience overload that performs an HTTP POST request with URL-encoded form data using a pre-parsed URI object. The complete response body is stored in the provided **NBString** reference. This function uses a **StringBufferObject** internally to handle the response data collection and forwards the call to the underlying DoUrlEncodedFormHttp implementation. Using a pre-parsed URI can improve performance when making multiple form submissions to the same host by avoiding repeated URL parsing operations.*
- **bool DoUrlEncodedFormHttp** (HttpMethod method, const char *pUrl, char *headers, char *form_data, **buffer_object** &result_buffer, uint16_t TIMEOUT_WAIT)

Executes an HTTP request with URL-encoded form data using a URL string and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is the core URL-encoded form HTTP function that performs an HTTP request with the specified method using URL-encoded form data and a URL string. The complete response body is stored in the provided buffer object. This function provides the foundation for form-based HTTP operations, supporting all standard HTTP methods while handling URL parsing and DNS resolution automatically. The form data is sent with the appropriate Content-Type header for proper server processing of web form submissions.
- **bool DoUrlEncodedFormPost** (const char *pUrl, char *headers, char *form_data, **buffer_object** &result_buffer, uint16_t TIMEOUT_WAIT)

Executes an HTTP POST request with URL-encoded form data using a URL string and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request with URL-encoded form data using a URL string. The complete response body is stored in the provided buffer object. This function is a simple wrapper around the generic DoUrlEncodedFormHttp function, specifically optimized for POST operations with form data. This provides the most straightforward interface for web form submissions when you have URL-encoded form data and want to use automatic URL parsing with buffer-based response handling.
- **bool DoUrlEncodedFormPost** (const char *pUrl, char *headers, char *form_data, **NBString** &result_string, uint16_t TIMEOUT_WAIT)

*Executes an HTTP POST request with URL-encoded form data using a URL string and stores the response in an **NBString**. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.*

*This is a convenience overload that performs an HTTP POST request with URL-encoded form data using a URL string. The complete response body is stored in the provided **NBString** reference. This function uses a **StringBufferObject** internally to handle the response data collection and forwards the call to the underlying DoUrlEncodedFormHttp implementation. This provides a simple interface for form-based POST operations with string-based response handling.*
- **bool DoJsonHttp** (HttpMethod method, const char *pUrl, const char *Json_Data_To_Post, **buffer_object** &result_buffer, const char *AdditionalHeaders=NULL, uint16_t TIMEOUT_WAIT=10 ***TICKS_PER_SECOND**)

Executes an HTTP request with raw JSON data using a URL string and stores the response in a buffer object.
- **bool DoJsonPost** (const char *pUrl, const char *Json_Data_To_Post, **buffer_object** &result_buffer, const char *AdditionalHeaders=NULL, uint16_t TIMEOUT_WAIT=10 ***TICKS_PER_SECOND**)

Executes an HTTP POST request with raw JSON data using a URL string and stores the response in a buffer object.
- **bool DoJsonHttp** (HttpMethod method, const char *pUrl, const char *Json_Data_To_Post, **NBString** &result_string, const char *AdditionalHeaders=NULL, uint16_t TIMEOUT_WAIT=10 ***TICKS_PER_SECOND**)

*Executes an HTTP request with raw JSON data using a URL string and stores the response in an **NBString**. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.*

*This is a convenience overload that performs an HTTP request with the specified method using raw JSON data provided as a string and a URL string. The complete response body is stored in the provided **NBString** reference. This function uses a **StringBufferObject** internally to handle the response data collection and forwards the call to the underlying DoJsonHttp implementation. The URL string will be parsed internally and DNS resolution will be performed automatically if needed, making this ideal for simple JSON operations with pre-formatted data.*
- **bool DoJsonPost** (const char *pUrl, const char *Json_Data_To_Post, **NBString** &result_string, const char *AdditionalHeaders=NULL, uint16_t TIMEOUT_WAIT=10 ***TICKS_PER_SECOND**)

*Executes an HTTP POST request with raw JSON data using a URL string and stores the response in an **NBString**. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.*

This is a convenience overload that performs an HTTP POST request using raw JSON data provided as a string and a URL string. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoJsonHttp implementation. This provides the simplest interface for JSON POST operations when you have pre-formatted JSON data and want string-based response handling with automatic URL parsing.

- bool [DoJsonHttp](#) (HttpMethod method, [ParsedURI](#) &TheUri, const char *Json_Data_To_Post, [buffer_object](#) &result_buffer, const char *AdditionalHeaders=NULL, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request with raw JSON data using a pre-parsed URI and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is the core JSON HTTP function for raw JSON string data that performs an HTTP request with the specified method using raw JSON data provided as a string and a pre-parsed URI object. The complete response body is stored in the provided buffer object. This function provides the foundation for JSON HTTP operations with pre-formatted JSON strings, offering optimal performance through pre-parsed URI components while supporting all standard HTTP methods.

- bool [DoJsonPost](#) ([ParsedURI](#) &TheUri, const char *Json_Data_To_Post, [buffer_object](#) &result_buffer, const char *AdditionalHeaders=NULL, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP POST request with raw JSON data using a pre-parsed URI and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using raw JSON data provided as a string and a pre-parsed URI object. The complete response body is stored in the provided buffer object. This function is a simple wrapper around the generic DoJsonHttp function, specifically optimized for POST operations with pre-formatted JSON strings, providing a streamlined interface when you already have JSON data in string format.

- bool [DoJsonHttp](#) (HttpMethod method, [ParsedURI](#) &TheUri, const char *Json_Data_To_Post, [NBString](#) &result_string, const char *AdditionalHeaders=NULL, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request with raw JSON data using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using raw JSON data provided as a string and a pre-parsed URI object. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoJsonHttp implementation. This provides flexibility for JSON operations with pre-formatted JSON strings and string-based response handling.

- bool [DoJsonPost](#) ([ParsedURI](#) &TheUri, const char *Json_Data_To_Post, [NBString](#) &result_string, const char *AdditionalHeaders=NULL, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP POST request with raw JSON data using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using raw JSON data provided as a string and a pre-parsed URI object. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoJsonHttp implementation. This provides a simple interface for JSON POST operations with pre-formatted JSON strings and string-based response handling.

- bool [DoJsonHttp](#) (HttpMethod method, const char *pUrl, [ParsedJsonDataSet](#) &jsonout, [buffer_object](#) &result_buffer, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request with JSON data from a [ParsedJsonDataSet](#) using a URL string and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is the core JSON HTTP function that performs an HTTP request with the specified method using JSON data from a [ParsedJsonDataSet](#) object and a URL string. The complete response body is stored in the provided buffer object. This function provides the foundation for programmatic JSON HTTP operations with string-based URLs, allowing dynamic construction and manipulation of JSON data before transmission. The URL string will be parsed internally and DNS resolution will be performed automatically if needed.

- bool [DoJsonPost](#) (const char *pUrl, [ParsedJsonDataSet](#) &jsonout, [buffer_object](#) &result_buffer, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP POST request with JSON data from a [ParsedJsonDataSet](#) using a URL string and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using JSON data from a [ParsedJsonDataSet](#) object and a URL string. The complete response body is stored in the provided buffer object. This function is a simple wrapper around the generic `DoJsonHttp` function, specifically optimized for POST operations with programmatically constructed JSON data, providing a streamlined interface for common JSON API interactions.

- `bool DoJsonHttp (HttpMethod method, const char *pUrl, ParsedJsonDataSet &jsonout, NBString &result_string, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *TICKS\_PER\_SECOND)`

Executes an HTTP request with JSON data from a [ParsedJsonDataSet](#) using a URL string and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using JSON data from a [ParsedJsonDataSet](#) object and a URL string. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying `DoJsonHttp` implementation. This provides flexibility for programmatic JSON operations with string-based response handling and simplified URL specification.

- `bool DoJsonPost (const char *pUrl, ParsedJsonDataSet &jsonout, NBString &result_string, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *TICKS\_PER\_SECOND)`

Executes an HTTP POST request with JSON data from a [ParsedJsonDataSet](#) using a URL string and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using JSON data from a [ParsedJsonDataSet](#) object and a URL string. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying `DoJsonHttp` implementation. This provides a simple interface for JSON POST operations with programmatically constructed data and string-based response handling.

- `bool DoJsonHttp (HttpMethod method, ParsedURI &TheUri, ParsedJsonDataSet &jsonout, buffer\_object &result_buffer, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *TICKS\_PER\_SECOND)`

Executes an HTTP request with JSON data from a [ParsedJsonDataSet](#) using a pre-parsed URI and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is the core JSON HTTP function that performs an HTTP request with the specified method using JSON data from a [ParsedJsonDataSet](#) object and a pre-parsed URI. The complete response body is stored in the provided buffer object. This function provides the foundation for programmatic JSON HTTP operations, allowing dynamic construction and manipulation of JSON data before transmission. The [ParsedJsonDataSet](#) will be automatically serialized to JSON format for the request body.

- `bool DoJsonPost (ParsedURI &TheUri, ParsedJsonDataSet &jsonout, buffer\_object &result_buffer, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *TICKS\_PER\_SECOND)`

Executes an HTTP POST request with JSON data from a [ParsedJsonDataSet](#) using a pre-parsed URI and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using JSON data from a [ParsedJsonDataSet](#) object and a pre-parsed URI. The complete response body is stored in the provided buffer object. This function is a simple wrapper around the generic `DoJsonHttp` function, specifically optimized for POST operations with programmatically constructed JSON data, providing a streamlined interface for common JSON API interactions.

- `bool DoJsonHttp (HttpMethod method, ParsedURI &TheUri, ParsedJsonDataSet &jsonout, NBString &result_string, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *TICKS\_PER\_SECOND)`

Executes an HTTP request with JSON data from a [ParsedJsonDataSet](#) using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using JSON data from a [ParsedJsonDataSet](#) object and a pre-parsed URI. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying `DoJsonHttp` implementation. This provides flexibility for programmatic JSON operations with string-based response handling.

- `bool DoJsonPost (ParsedURI &TheUri, ParsedJsonDataSet &jsonout, NBString &result_string, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *TICKS\_PER\_SECOND)`

Executes an HTTP POST request with JSON data from a [ParsedJsonDataSet](#) using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using JSON data from a [ParsedJsonDataSet](#) object and a pre-parsed URI. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying

DoJsonHttp implementation. This approach allows for dynamic JSON POST requests with programmatically constructed data.

- bool [DoJsonHttpFile](#) (HttpMethod method, const char *pUrl, const char *FragmentName, [buffer_object](#) &result_buffer, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request with JSON data from a file fragment using a URL string and stores the response in a buffer object.

- bool [DoJsonPostHttpFile](#) (const char *pUrl, const char *FragmentName, [buffer_object](#) &result_buffer, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP POST request with JSON data from a file fragment using a URL string and stores the response in a buffer object.

- bool [DoJsonHttpFile](#) (HttpMethod method, const char *pUrl, const char *FragmentName, [NBString](#) &result_string, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Performs an HTTP request and stores JSON response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that wraps the result [NBString](#) in a [StringBufferObject](#) and calls the main [DoJsonHttpFile](#) function.

- bool [DoJsonPostHttpFile](#) (const char *pUrl, const char *FragmentName, [NBString](#) &result_string, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Performs an HTTP POST request and stores JSON response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience wrapper function that simplifies HTTP POST operations with JSON response handling. It internally calls [DoJsonHttpFile](#) with [HttpMethod::Post](#) and automatically manages the buffer conversion from [NBString](#) to [StringBufferObject](#). The function handles the complete HTTP transaction including connection establishment, request transmission, response reception, and JSON parsing. Only the specified JSON fragment is extracted from the response and stored in the result string, making it efficient for retrieving specific data from larger JSON responses. The function is particularly useful for REST API interactions where you need to POST data and extract a specific piece of information from the JSON response.

- bool [DoJsonHttpFile](#) (HttpMethod method, [ParsedURI](#) &TheUri, const char *FragmentName, [buffer_object](#) &result_buffer, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Sends an HTTP request and processes JSON response using a pre-parsed URI.

- bool [DoJsonPostHttpFile](#) ([ParsedURI](#) &TheUri, const char *FragmentName, [buffer_object](#) &result_buffer, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP POST request with JSON data from a file fragment using a pre-parsed URI and stores the response in a buffer object.

- bool [DoJsonHttpFile](#) (HttpMethod method, [ParsedURI](#) &TheUri, const char *FragmentName, [NBString](#) &result_string, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request with JSON data from a file fragment using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using JSON data loaded from a file fragment and a pre-parsed URI object. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying [DoJsonHttpFile](#) implementation. This approach provides maximum flexibility for JSON-based HTTP operations with pre-stored data fragments.

- bool [DoJsonPostHttpFile](#) ([ParsedURI](#) &TheUri, const char *FragmentName, [NBString](#) &result_string, const char *AdditionalHeaders, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP POST request with JSON data from a file fragment using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using JSON data loaded from a specified file fragment and a pre-parsed URI object. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying [DoJsonHttpFile](#) implementation. This approach allows for efficient JSON POST requests with pre-defined data stored in file fragments.

- bool [DoHttp](#) (HttpMethod method, [ParsedURI](#) &TheUri, [buffer_object](#) &result_buffer, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request using a pre-parsed URI and stores the response in a buffer object.

- bool [DoGet](#) ([ParsedURI](#) &TheUri, [buffer_object](#) &result_buffer, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP GET request using a pre-parsed URI and stores the response in a buffer object.

- bool **DoHttp** (HttpMethod method, [ParsedURI](#) &TheUri, [NSString](#) &result_string, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request using a pre-parsed URI and stores the response in an [NSString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using a pre-parsed URI object and stores the complete response body in the provided [NSString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoHttp implementation that accepts a [buffer_object](#). Using a pre-parsed URI can improve performance when making multiple requests to the same host by avoiding repeated URL parsing operations.

- bool **DoGet** ([ParsedURI](#) &TheUri, [NSString](#) &result_string, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP GET request using a pre-parsed URI and stores the response in an [NSString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP GET request using a pre-parsed URI object and stores the complete response body in the provided [NSString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoHttp implementation. Using a pre-parsed URI can improve performance when making multiple requests to the same host by avoiding repeated URL parsing operations.

- bool **DoHttpEx** (HttpMethod method, [ParsedURI](#) &TheUri, const char *headers, [buffer_object](#) &result_buffer, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request with custom headers using a pre-parsed URI and stores the response in a buffer object.

- bool **DoGetEx** ([ParsedURI](#) &TheUri, const char *headers, [buffer_object](#) &result_buffer, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP GET request with custom headers using a pre-parsed URI and stores the response in a buffer object.

- bool **DoHttpEx** (HttpMethod method, [ParsedURI](#) &TheUri, const char *headers, [NSString](#) &result_string, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request with custom headers using a pre-parsed URI and stores the response in an [NSString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using a pre-parsed URI object and allows custom headers to be included in the request. The complete response body is stored in the provided [NSString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoHttpEx implementation that accepts a [buffer_object](#).

- bool **DoGetEx** ([ParsedURI](#) &TheUri, const char *headers, [NSString](#) &result_string, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP GET request with custom headers using a pre-parsed URI and stores the response in an [NSString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP GET request using a pre-parsed URI object and allows custom headers to be included in the request. The complete response body is stored in the provided [NSString](#) reference.

This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoHttpEx implementation.

- bool **DoHttp** (HttpMethod method, const char *pUrl, [buffer_object](#) &result_buffer, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP/HTTPS request using the specified method and stores the response in a buffer object.

- bool **DoGet** (const char *pUrl, [buffer_object](#) &result_buffer, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP/HTTPS GET request and stores the response in a buffer object.

- bool **DoHttp** (HttpMethod method, const char *pUrl, [NSString](#) &result_string, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Performs an HTTP request using the specified method and stores the response in an [NSString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the given method to the specified URL and stores the complete response body in the provided [NSString](#) reference. The function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoHttp implementation.

- bool **DoGet** (const char *pUrl, [NSString](#) &result_string, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Performs an HTTP GET request and stores the result in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overload provides a convenient way to perform HTTP GET requests when you want to store the response directly in an [NBString](#) object. The function wraps the [NBString](#) in a [StringBufferObject](#) and delegates to the main [DoHttp](#) function.

- bool [DoHttpEx](#) (HttpMethod method, const char *pUrl, const char *headers, [buffer_object](#) &result_buffer, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Execute a HTTP/HTTPS request using a pointer to a URL string.

- bool [DoGetEx](#) (const char *pUrl, const char *headers, [buffer_object](#) &result_buffer, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP GET request with custom headers using a URL string and stores the response in a buffer object.

- bool [DoHttpEx](#) (HttpMethod method, const char *pUrl, const char *headers, [NBString](#) &result_string, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request with custom headers using a URL string and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method and custom headers using a URL string. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying [DoHttpEx](#) implementation. This provides maximum flexibility for HTTP operations with custom headers and string-based response handling.

- bool [DoGetEx](#) (const char *pUrl, const char *headers, [NBString](#) &result_string, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP GET request with custom headers using a URL string and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP GET request with custom headers using a URL string. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying [DoHttpEx](#) implementation. This provides a simple interface for GET requests that require custom headers with string-based response handling.

- int [DoHttp](#) (HttpMethod method, [ParsedURI](#) &TheUri, unsigned char *result, int maxl, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request using a pre-parsed URI and stores the response in a raw buffer.

- int [DoHttpEx](#) (HttpMethod method, [ParsedURI](#) &TheUri, const char *headers, unsigned char *result, int maxl, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request with custom headers using a pre-parsed URI and stores the response in a raw buffer.

- int [DoHttp](#) (HttpMethod method, const char *pUrl, unsigned char *result, int maxl, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request using a URL string and stores the response in a raw buffer.

- int [DoHttpEx](#) (HttpMethod method, const char *pUrl, const char *headers, unsigned char *result, int maxl, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP request with custom headers using a URL string and stores the response in a raw buffer.

- int [DoGet](#) ([ParsedURI](#) &TheUri, unsigned char *result, int maxl, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP GET request using a pre-parsed URI and stores the response in a raw buffer.

- int [DoGetEx](#) ([ParsedURI](#) &TheUri, const char *headers, unsigned char *result, int maxl, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP GET request with custom headers using a pre-parsed URI and stores the response in a raw buffer.

- int [DoGet](#) (const char *pUrl, unsigned char *result, int maxl, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP GET request using a URL string and stores the response in a raw buffer.

- int [DoGetEx](#) (const char *pUrl, const char *headers, unsigned char *result, int maxl, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Executes an HTTP GET request with custom headers using a URL string and stores the response in a raw buffer.

- int [DoGetUpdate](#) ([ParsedURI](#) &TheUri, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Execute a firmware update from the specified URI.

- int [DoGetUpdate](#) (const char *pUrl, uint16_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))

Execute a firmware update from the specified URI.

- int [PopulateAuthHeader](#) (const char *user, const char *password, char *buffer, int maxlen)
Generates a Base64-encoded HTTP Basic Authentication header string for inclusion in HTTP requests.
- int [StartConnection](#) ([ParsedURI](#) &TheUri, uint32_t timeout=[TICKS_PER_SECOND](#) *10)
Establishes a TCP/TLS connection to a host using a pre-parsed URI with timeout specified in ticks.
- int [WriteHttpRequestHeaders](#) (int fd, [ParsedURI](#) &uri, const char *methodStr, int content_len=0, const char *content_type=0, const char *extra_header=0)
Writes HTTP request headers to an established connection file descriptor.
- int [StartTransaction](#) ([ParsedURI](#) &TheUri, const char *headers=0, const char *action="GET", uint32_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))
Initiates an HTTP transaction using a pre-parsed URI by establishing a connection and sending request headers.
- int [StartTransaction](#) (const char *pUrl, const char *headers=0, const char *action="GET", uint32_t TIMEOUT_WAIT=10 *[TICKS_PER_SECOND](#))
Initiates an HTTP transaction using a URL string by establishing a connection and sending request headers.
- bool [DoActualClientRequest](#) ([ParsedURI](#) &TheUri, uint16_t &next_time_delay)
Callback to specify for performing and handle a WebClient request.
- bool [StartWebClient](#) (int prio, const char *url1, const char *url2=NULL, bool bDoNtp=false)
Start the web client using a URL string.
- bool [StartWebClient](#) (int prio, const [NBString](#) &url1, const [NBString](#) &url2, bool bDoNtp=false)
Start the web client using a [NBString](#).
- bool [StartWebClient](#) (int prio, const [NBString](#) &url1, bool bDoNtp=false)
Start the web client using a [NBString](#).

Variables

- WebErrorReporterFunc * [pWebErrorReporter](#)
Set this variable to a callback function to specify a handler for errors.

14.60.1 Detailed Description

```
#include< http_funcs.h>
```

Functions for interacting with the web, like HTTP GET/POST requests.

An easy way to get web data (from our [GetNonJsonFromServer](#) example):

```
const char *purl = "http://www.example.com/";
int result = 0;
static unsigned char ResultBuffer[5000];

result = DoGet(purl, ResultBuffer, 5000, 60 * TICKS\_PER\_SECOND);

if (result > 0)
{
    printf("Result = %d\r\n", result);
    for (int i = 0; i < result; i++)
    {
        printf("%c", ResultBuffer[i]);
    }
    printf("]\r\n");
}
else
{
    iprintf("Failed to contact server\r\n");
}

#include< web_client.h>
```

Functions for instantiating a periodic web client

From our WebClient examples:

```
const char *url = "http://www.example.com";

void HandleErrorState(int state)
{
    if (state == WEB_CLIENT_ERROR_NO_ERROR) return;
    if ((state <= WEB_CLIENT_ERROR_LAST_STATE) && (state > 0))
        iprintf("Set Error state [%d] [%s]\r\n", state, web_error_state_text[state]);
    else
        iprintf("Error state unknown %d\r\n", state);
}

bool DoActualClientRequest(ParsedURI &TheUri, uint16_t &next_time_delay)
{
    static unsigned char ResultBuffer[5000];

    bool result = DoGet(TheUri, ResultBuffer);
    if (result)
    {
        printf("Result = %d\r\n", result);
        for (int i = 0; i < result; i++)
        {
            printf("%c", ResultBuffer[i]);
        }
        printf("]\r\n");
        next_time_delay = 10 * TICKS_PER_SECOND;
        return true;
    }

    next_time_delay = 10 * TICKS_PER_SECOND;
    return false;
}

void UserMain(void *pd)
{
    init();
    WaitForActiveNetwork(TICKS_PER_SECOND * 5);

    pWebErrorReporter = HandleErrorState;
    StartWebClient(MAIN_PRIO - 2, url);

    while (1)
    {
        OSTimeDly(TICKS_PER_SECOND * 1);
    }
}
```

14.60.2 Function Documentation

14.60.2.1 DoActualClientRequest()

```
bool DoActualClientRequest (
    ParsedURI & TheUri,
    uint16_t & next_time_delay)
#include <web_client.h>
Callback to specify for performing and handle a WebClient request.
```

Parameters

<i>TheUri</i>	Pointer to a ParsedURI object (representing the URL)
<i>next_time_delay</i>	Pointer to the number of ticks to wait before the next request

Returns

True on success, false on failure.

14.60.2.2 DoGet() [1/6]

```
bool DoGet (
    const char * pUrl,
```

```

        buffer_object & result_buffer,
        uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>

```

Executes an HTTP/HTTPS GET request and stores the response in a buffer object.

This function performs an HTTP GET request to the specified URL and stores the complete response body in the provided buffer object. The URL is parsed to extract the host information, and DNS resolution is performed automatically if a hostname is provided instead of an IP address. This is a convenience wrapper around the generic DoHttp function.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the GET request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, or HTTP error)

Note

The result_buffer contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size.

See also

[DoHttp\(\)](#) for the underlying HTTP implementation

[buffer_object](#) for response buffer management

14.60.2.3 DoGet() [2/6]

```

bool DoGet (
    const char * pUrl,
    NSString & result_string,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>

```

Performs an HTTP GET request and stores the result in an [NSString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overload provides a convenient way to perform HTTP GET requests when you want to store the response directly in an [NSString](#) object. The function wraps the [NSString](#) in a [StringBufferObject](#) and delegates to the main DoHttp function.

Parameters

<i>pUrl</i>	Pointer to null-terminated string containing the URL to request
<i>result_string</i>	Reference to NSString in which to store the HTTP response
<i>TIMEOUT_WAIT</i>	Timeout value in system ticks (default: 10 seconds)

Returns

true if the HTTP request was successful, false otherwise

See also

[DoHttp\(\)](#)
[StringBufferObject](#)
[NBString](#)

14.60.2.4 DoGet() [3/6]

```
int DoGet (
    const char * pUrl,
    unsigned char * result,
    int maxl,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP GET request using a URL string and stores the response in a raw buffer.

This is a low-level HTTP GET function that performs an HTTP GET request using a URL string and stores the response body in a provided raw buffer. The function handles URL parsing and DNS resolution automatically if needed. The response is written directly to the buffer without any additional processing or wrapper objects, providing direct access to the raw HTTP response data with minimal overhead.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>result</i>	Pointer to raw buffer where the HTTP response body will be stored Buffer must be pre-allocated with sufficient space for the response
<i>maxl</i>	Maximum length of the result buffer in bytes Response will be truncated if it exceeds this length
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

Number of bytes received and stored in the result buffer on success (≥ 0), or a negative error code on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The result buffer will contain the raw response body data. No null termination is added - treat as binary data. The response may be truncated if it exceeds the maxl parameter.

See also

[DoHttp\(\)](#) for the underlying HTTP implementation
[DoGet\(ParsedURI&, unsigned char*, int, uint16_t\)](#) for pre-parsed URI version
[DoGet\(const char*, buffer_object&, uint16_t\)](#) for [buffer_object](#) version

14.60.2.5 DoGet() [4/6]

```
bool DoGet (
    ParsedURI & TheUri,
    buffer_object & result_buffer,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP GET request using a pre-parsed URI and stores the response in a buffer object.

This function performs an HTTP GET request using a pre-parsed URI object and stores the complete response body in the provided buffer object. This is a convenience wrapper around the generic DoHttp function, providing a simple interface for GET requests while leveraging pre-parsed URI components for improved performance when making multiple requests to the same host by avoiding repeated URL parsing operations.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use <code>TICKS_PER_SECOND</code> for conversion

Returns

true if the GET request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The *result_buffer* contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. Using pre-parsed URIs improves performance for repeated requests to the same host.

See also

[DoHttp\(\)](#) for the underlying HTTP implementation
[ParsedURI](#) for URL parsing and component access
[buffer_object](#) for response buffer management
[DoGet\(const char*, buffer_object&, uint16_t\)](#) for string URL overload
[DoGetEx\(\)](#) for GET requests with custom headers support

14.60.2.6 DoGet() [5/6]

```
bool DoGet (
    ParsedURI & TheUri,
    NBString & result_string,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP GET request using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP GET request using a pre-parsed URI object and stores the complete response body in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying [DoHttp](#) implementation. Using a pre-parsed URI can improve performance when making multiple requests to the same host by avoiding repeated URL parsing operations.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use <code>TICKS_PER_SECOND</code> for conversion

Returns

true if the GET request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The result_string contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Using pre-parsed URIs improves performance for repeated requests to the same host.

See also

[DoHttp\(\)](#) for the underlying HTTP implementation
[ParsedURI](#) for URL parsing and component access
[StringBufferObject](#) for response handling details
[DoGet\(const char*, NBString&, uint16_t\)](#) for string URL overload
[DoGetEx\(\)](#) for GET requests with custom headers support

14.60.2.7 DoGet() [6/6]

```
int DoGet (
    ParsedURI & TheUri,
    unsigned char * result,
    int maxl,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP GET request using a pre-parsed URI and stores the response in a raw buffer.

This is a low-level HTTP GET function that performs an HTTP GET request using a pre-parsed URI object and stores the response body in a provided raw buffer. This function provides optimal performance by using pre-parsed URI components, eliminating URL parsing and DNS lookup overhead when making multiple GET requests to the same host. The response is written directly to the buffer without any additional processing or wrapper objects.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>result</i>	Pointer to raw buffer where the HTTP response body will be stored Buffer must be pre-allocated with sufficient space for the response
<i>maxl</i>	Maximum length of the result buffer in bytes Response will be truncated if it exceeds this length
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

Number of bytes received and stored in the result buffer on success (≥ 0), or a negative error code on failure (network error, timeout, connection error, or HTTP error response)

Note

The result buffer will contain the raw response body data. No null termination is added - treat as binary data. Using pre-parsed URIs improves performance for repeated requests to the same host. The response may be truncated if it exceeds the maxl parameter.

See also

[DoHttp\(\)](#) for the underlying HTTP implementation
[ParsedURI](#) for URL parsing and component access
[DoGet\(const char*, unsigned char*, int, uint16_t\)](#) for string URL version
[DoGet\(ParsedURI&, buffer_object&, uint16_t\)](#) for [buffer_object](#) version

14.60.2.8 DoGetEx() [1/6]

```
bool DoGetEx (
    const char * pUrl,
    const char * headers,
    buffer_object & result_buffer,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP GET request with custom headers using a URL string and stores the response in a buffer object. This function performs an HTTP GET request with custom headers using a URL string and stores the complete response body in the provided buffer object. This is a convenience wrapper around the generic DoHttpEx function, specifically optimized for GET operations with custom header support. The URL string will be parsed internally and DNS resolution will be performed automatically if needed, making this ideal for GET requests that require additional headers such as authentication, user-agent, or content negotiation.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>headers</i>	Pointer to null-terminated string containing custom HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr or empty string if no custom headers are needed
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the GET request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The `result_buffer` contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. Custom headers must be properly formatted without trailing CRLF on the final header.

See also

[DoHttpEx\(\)](#) for the underlying HTTP implementation with full method support

[buffer_object](#) for response buffer management

[DoGet\(const char*, buffer_object&, uint16_t\)](#) for GET requests without custom headers

[DoGetEx\(ParsedURI&, const char*, buffer_object&, uint16_t\)](#) for pre-parsed URI version

14.60.2.9 DoGetEx() [2/6]

```
bool DoGetEx (
    const char * pUrl,
    const char * headers,
    NBString & result_string,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP GET request with custom headers using a URL string and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP GET request with custom headers using a URL string. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoHttpEx implementation. This provides a simple interface for GET requests that require custom headers with string-based response handling.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>headers</i>	Pointer to null-terminated string containing custom HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr or empty string if no custom headers are needed
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the GET request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The result_string contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Custom headers must be properly formatted without trailing CRLF on the final header.

See also

[DoHttpEx\(HttpMethod, const char*, const char*, NBString&, uint16_t\)](#) for the underlying generic method implementation

[StringBufferObject](#) for response handling details

[DoGet\(const char*, NBString&, uint16_t\)](#) for GET requests without custom headers

[DoGetEx\(ParsedURI&, const char*, NBString&, uint16_t\)](#) for pre-parsed URI version

14.60.2.10 DoGetEx() [3/6]

```
int DoGetEx (
    const char * pUrl,
    const char * headers,
    unsigned char * result,
    int maxl,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
```

```
#include <http_funcs.h>
```

Executes an HTTP GET request with custom headers using a URL string and stores the response in a raw buffer. This is a low-level extended HTTP GET function that performs an HTTP GET request with custom headers using a URL string and stores the response body in a provided raw buffer. The function handles URL parsing and DNS resolution automatically while supporting custom headers for authentication, content negotiation, or other advanced GET operations. The response is written directly to the buffer without any additional processing.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>headers</i>	Pointer to null-terminated string containing custom HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr or empty string if no custom headers are needed
<i>result</i>	Pointer to raw buffer where the HTTP response body will be stored Buffer must be pre-allocated with sufficient space for the response
<i>maxl</i>	Maximum length of the result buffer in bytes Response will be truncated if it exceeds this length

<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use <i>TICKS_PER_SECOND</i> for conversion

Returns

Number of bytes received and stored in the result buffer on success (≥ 0), or a negative error code on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The result buffer will contain the raw response body data. No null termination is added - treat as binary data. Custom headers must be properly formatted without trailing CRLF on the final header. The response may be truncated if it exceeds the *maxl* parameter.

See also

[DoHttpEx\(\)](#) for the underlying HTTP implementation

[DoGetEx\(ParsedURI&, const char*, unsigned char*, int, uint16_t\)](#) for pre-parsed URI version

[DoGetEx\(const char*, const char*, buffer_object&, uint16_t\)](#) for [buffer_object](#) version

14.60.2.11 DoGetEx() [4/6]

```
bool DoGetEx (
    ParsedURI & TheUri,
    const char * headers,
    buffer_object & result_buffer,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP GET request with custom headers using a pre-parsed URI and stores the response in a buffer object.

This function performs an HTTP GET request using a pre-parsed URI object and allows custom headers to be included in the request. The complete response body is stored in the provided buffer object. This is a convenience wrapper around the generic [DoHttpEx](#) function, providing enhanced control over the request headers while using pre-parsed URI components for improved performance when making multiple requests to the same host.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>headers</i>	Pointer to null-terminated string containing custom HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr or empty string if no custom headers are needed
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use <i>TICKS_PER_SECOND</i> for conversion

Returns

true if the GET request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The `result_buffer` contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. Headers must be properly formatted without trailing CRLF on the final header. Using pre-parsed URIs can improve performance for repeated requests to the same host.

See also

[DoHttpEx\(\)](#) for the underlying HTTP implementation with full method support
[ParsedURI](#) for URL parsing and component access
[buffer_object](#) for response buffer management
[DoGet\(\)](#) for simpler GET requests without custom headers

14.60.2.12 DoGetEx() [5/6]

```
bool DoGetEx (
    ParsedURI & TheUri,
    const char * headers,
    NBString & result_string,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP GET request with custom headers using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP GET request using a pre-parsed URI object and allows custom headers to be included in the request. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying `DoHttpEx` implementation.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, etc.)
<i>headers</i>	Pointer to null-terminated string containing custom HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no custom headers are needed
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use <code>TICKS_PER_SECOND</code> for conversion

Returns

true if the GET request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The `result_string` contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Custom headers must be properly formatted with CRLF line endings.

See also

[DoHttpEx\(\)](#) for the underlying HTTP implementation with header support
[ParsedURI](#) for URL parsing and component access
[StringBufferObject](#) for response handling details
[DoGet\(\)](#) for simpler GET requests without custom headers

14.60.2.13 DoGetEx() [6/6]

```
int DoGetEx (
    ParsedURI & TheUri,
    const char * headers,
    unsigned char * result,
    int maxl,
    uint16_t TIMEOUT_WAIT = 10 * TICKS\_PER\_SECOND) [inline]
```

```
#include <http_funcs.h>
```

Executes an HTTP GET request with custom headers using a pre-parsed URI and stores the response in a raw buffer.

This is a low-level extended HTTP GET function that performs an HTTP GET request with custom headers using a pre-parsed URI object and stores the response body in a provided raw buffer. This function provides optimal performance through pre-parsed URI components while supporting custom headers for authentication, content negotiation, or other advanced GET operations. The response is written directly to the buffer without any additional processing.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>headers</i>	Pointer to null-terminated string containing custom HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr or empty string if no custom headers are needed
<i>result</i>	Pointer to raw buffer where the HTTP response body will be stored Buffer must be pre-allocated with sufficient space for the response
<i>maxl</i>	Maximum length of the result buffer in bytes Response will be truncated if it exceeds this length
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

Number of bytes received and stored in the result buffer on success (≥ 0), or a negative error code on failure (network error, timeout, connection error, or HTTP error response)

Note

The result buffer will contain the raw response body data. No null termination is added - treat as binary data. Custom headers must be properly formatted without trailing CRLF on the final header. Using pre-parsed URIs improves performance for repeated requests to the same host. The response may be truncated if it exceeds the *maxl* parameter.

See also

[DoHttpEx\(\)](#) for the underlying HTTP implementation

[ParsedURI](#) for URL parsing and component access

[DoGetEx\(const char*, const char*, unsigned char*, int, uint16_t\)](#) for string URL version

[DoGetEx\(ParsedURI&, const char*, buffer_object&, uint16_t\)](#) for *buffer_object* version

14.60.2.14 DoGetUpdate() [1/2]

```
int DoGetUpdate (
    const char * pUrl,
    uint16_t TIMEOUT_WAIT = 10 * TICKS\_PER\_SECOND)
```

```
#include <http_funcs.h>
```

Execute a firmware update from the specified URI.

Parameters

<i>pUrl</i>	Pointer to a URL string. A DNS lookup will be done if necessary.
<i>TIMEOUT_WAIT</i>	Timeout in in system time ticks to wait for a response.

Returns

14.60.2.15 DoGetUpdate() [2/2]

```
int DoGetUpdate (
    ParsedURI & TheUri,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
#include <http_funcs.h>
Execute a firmware update from the specified URI.
```

Parameters

<i>TheUri</i>	Reference to the ParsedURI object.
<i>TIMEOUT_WAIT</i>	Timeout in in system time ticks to wait for a response.

Returns

14.60.2.16 DoHttp() [1/6]

```
bool DoHttp (
    HttpMethod method,
    const char * pUrl,
    buffer_object & result_buffer,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
#include <http_funcs.h>
```

Executes an HTTP/HTTPS request using the specified method and stores the response in a buffer object. This function performs an HTTP request with the given method to the specified URL and stores the complete response body in the provided buffer object. The URL is parsed to extract the host information, port, and path components. DNS resolution is performed automatically if a hostname is provided instead of an IP address. This is the core HTTP client function that supports all standard HTTP methods.

Parameters

<i>method</i>	HTTP method to use for the request (GET, POST, PUT, DELETE, etc.) See HttpMethod enumeration for all supported methods
<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, invalid URL, connection refused, or HTTP error response)

Note

The `result_buffer` contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. For HTTPS URLs, SSL/TLS connection will be established automatically.

See also

`HttpMethod` for available HTTP methods

[buffer_object](#) for response buffer management

[DoGet\(\)](#) for a GET-specific convenience wrapper

`DoPost()` for a POST-specific convenience wrapper

14.60.2.17 DoHttp() [2/6]

```
bool DoHttp (
    HttpMethod method,
    const char * pUrl,
    NBString & result_string,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Performs an HTTP request using the specified method and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the given method to the specified URL and stores the complete response body in the provided [NBString](#) reference. The function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying `DoHttp` implementation.

Parameters

<i>method</i>	HTTP method to use for the request (GET, POST, PUT, DELETE, etc.) See <code>HttpMethod</code> enumeration for available options
<i>pUrl</i>	Pointer to null-terminated string containing the target URL
<i>result_string</i>	Reference to NBString in which to store the HTTP response body
<i>TIMEOUT_WAIT</i>	Optional timeout in system ticks (default: 10 seconds) Use <code>TICKS_PER_SECOND</code> to convert from seconds to ticks

Returns

true if the HTTP request completed successfully, false on failure (timeout, network error, or HTTP error response)

Note

The `result_string` will be cleared and populated with the response data. For large responses, ensure sufficient memory is available.

See also

`DoHttp(HttpMethod, const char*, StringBufferObject&, uint16_t)` for the underlying implementation

[DoGet\(\)](#) for a GET-specific convenience function

`DoPost()` for a POST-specific convenience function

[StringBufferObject](#) for response handling details

`HttpMethod` for available HTTP methods

14.60.2.18 DoHttp() [3/6]

```
int DoHttp (
    HttpMethod method,
    const char * pUrl,
    unsigned char * result,
    int maxl,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
```

```
#include <http_funcs.h>
```

Executes an HTTP request using a URL string and stores the response in a raw buffer.

This is a low-level HTTP client function that performs an HTTP request with the specified method using a URL string and stores the response body in a provided raw buffer. The function handles URL parsing and DNS resolution automatically if needed. The response is written directly to the buffer without any additional processing or wrapper objects, providing direct access to the raw HTTP response data.

Parameters

<i>method</i>	HTTP method to use for the request (GET, POST, PUT, DELETE, etc.) See HttpMethod enumeration for all supported methods
<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>result</i>	Pointer to raw buffer where the HTTP response body will be stored Buffer must be pre-allocated with sufficient space for the response
<i>maxl</i>	Maximum length of the result buffer in bytes Response will be truncated if it exceeds this length
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

Number of bytes received and stored in the result buffer on success (≥ 0), or a negative error code on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The result buffer will contain the raw response body data. No null termination is added - treat as binary data. The response may be truncated if it exceeds the maxl parameter.

See also

HttpMethod for available HTTP methods

[DoHttp\(HttpMethod, ParsedURI&, unsigned char*, int, uint16_t\)](#) for pre-parsed URI version

[DoHttp\(HttpMethod, const char*, buffer_object&, uint16_t\)](#) for [buffer_object](#) version

14.60.2.19 DoHttp() [4/6]

```
bool DoHttp (
    HttpMethod method,
    ParsedURI & TheUri,
    buffer_object & result_buffer,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
```

```
#include <http_funcs.h>
```

Executes an HTTP request using a pre-parsed URI and stores the response in a buffer object.

This is the core HTTP client function that performs an HTTP request with the specified method using a pre-parsed URI object and stores the complete response body in the provided buffer object. This function provides enhanced performance by using pre-parsed URI components, eliminating the overhead of repeated URL parsing when making multiple requests to the same host. It supports all standard HTTP methods and handles the complete request/response cycle.

Parameters

<i>method</i>	HTTP method to use for the request (GET, POST, PUT, DELETE, etc.) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, fragment, etc.)
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, connection error, invalid method, or HTTP error response)

Note

The result_buffer contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. For HTTPS URLs, SSL/TLS connection will be established automatically. Using pre-parsed URIs improves performance for repeated requests to the same host.

See also

HttpMethod for available HTTP methods

[ParsedURI](#) for URL parsing and component access

[buffer_object](#) for response buffer management

[DoHttp\(HttpMethod, const char*, buffer_object&, uint16_t\)](#) for string URL overload

[DoHttpEx\(\)](#) for requests with custom headers support

14.60.2.20 DoHttp() [5/6]

```
bool DoHttp (
    HttpMethod method,
    ParsedURI & TheUri,
    NBString & result_string,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP request using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. This is a convenience overload that performs an HTTP request with the specified method using a pre-parsed URI object and stores the complete response body in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoHttp implementation that accepts a [buffer_object](#). Using a pre-parsed URI can improve performance when making multiple requests to the same host by avoiding repeated URL parsing operations.

Parameters

<i>method</i>	HTTP method to use for the request (GET, POST, PUT, DELETE, etc.) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The `result_string` contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Using pre-parsed URIs improves performance for repeated requests to the same host.

See also

[DoHttp\(HttpMethod, ParsedURI&, buffer_object&, uint16_t\)](#) for the underlying implementation

[HttpMethod](#) for available HTTP methods

[ParsedURI](#) for URL parsing and component access

[StringBufferObject](#) for response handling details

[DoHttp\(HttpMethod, const char*, NBString&, uint16_t\)](#) for string URL overload

[DoHttpEx\(\)](#) for requests with custom headers support

14.60.2.21 DoHttp() [6/6]

```
int DoHttp (
    HttpMethod method,
    ParsedURI & TheUri,
    unsigned char * result,
    int maxl,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
#include <http_funcs.h>
```

Executes an HTTP request using a pre-parsed URI and stores the response in a raw buffer.

This is a low-level HTTP client function that performs an HTTP request with the specified method using a pre-parsed URI object and stores the response body in a provided raw buffer. This function provides optimal performance by using pre-parsed URI components, eliminating URL parsing and DNS lookup overhead when making multiple requests to the same host. The response is written directly to the buffer without any additional processing or wrapper objects.

Parameters

<i>method</i>	HTTP method to use for the request (GET, POST, PUT, DELETE, etc.) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>result</i>	Pointer to raw buffer where the HTTP response body will be stored Buffer must be pre-allocated with sufficient space for the response
<i>maxl</i>	Maximum length of the result buffer in bytes Response will be truncated if it exceeds this length
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

Number of bytes received and stored in the result buffer on success (≥ 0), or a negative error code on failure (network error, timeout, connection error, or HTTP error response)

Note

The result buffer will contain the raw response body data. No null termination is added - treat as binary data. Using pre-parsed URIs improves performance for repeated requests to the same host. The response may be truncated if it exceeds the `maxl` parameter.

See also

`HttpMethod` for available HTTP methods

[ParsedURI](#) for URL parsing and component access

[DoHttp\(HttpMethod, const char*, unsigned char*, int, uint16_t\)](#) for string URL version

[DoHttp\(HttpMethod, ParsedURI&, buffer_object&, uint16_t\)](#) for `buffer_object` version

14.60.2.22 DoHttpEx() [1/6]

```
bool DoHttpEx (
    HttpMethod method,
    const char * pUrl,
    const char * headers,
    buffer_object & result_buffer,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
```

```
#include <http_funcs.h>
```

Execute a HTTP/HTTPS request using a pointer to a URL string.

This function will parse the URL string to connect to the host.

Parameters

<i>method</i>	HTTP Method to use
<i>pUrl</i>	Pointer to a URL string. A DNS lookup will be done if necessary.
<i>headers</i>	Additional HTTP header fields. Multiple header fields should be concatenated together and separated by by <code>\r\n</code> Do not put <code>\r\n</code> for last header (no trailing <code>\r\n</code>)
<i>&result_buffer</i>	Reference to the memory buffer in which to store the result.
<i>TIMEOUT_WAIT</i>	Timeout in in system time ticks to wait for a response.

Returns

True on success, false on failure.

14.60.2.23 DoHttpEx() [2/6]

```
bool DoHttpEx (
    HttpMethod method,
    const char * pUrl,
    const char * headers,
    NBString & result_string,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]

#include <http_funcs.h>
```

Executes an HTTP request with custom headers using a URL string and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method and custom headers using a URL string. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying `DoHttpEx` implementation. This provides maximum flexibility for HTTP operations with custom headers and string-based response handling.

Parameters

<i>method</i>	HTTP method to use for the request (GET, POST, PUT, DELETE, etc.) See HttpMethod enumeration for all supported methods
<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>headers</i>	Pointer to null-terminated string containing custom HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr or empty string if no custom headers are needed
<i>result_string</i>	Reference to NSString in which to store the HTTP response body The string will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The result_string contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Custom headers must be properly formatted without trailing CRLF on the final header.

See also

[DoHttpEx\(HttpMethod, const char*, const char*, buffer_object&, uint16_t\)](#) for the underlying implementation

[HttpMethod](#) for available HTTP methods

[StringBufferObject](#) for response handling details

[DoHttp\(HttpMethod, const char*, NSString&, uint16_t\)](#) for requests without custom headers

14.60.2.24 DoHttpEx() [3/6]

```
int DoHttpEx (
    HttpMethod method,
    const char * pUrl,
    const char * headers,
    unsigned char * result,
    int maxl,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
```

```
#include <http_funcs.h>
```

Executes an HTTP request with custom headers using a URL string and stores the response in a raw buffer.

This is a low-level extended HTTP client function that performs an HTTP request with the specified method and custom headers using a URL string. The response body is stored in a provided raw buffer. The function handles URL parsing and DNS resolution automatically if needed while supporting custom headers for authentication, content negotiation, or other advanced HTTP operations. The response is written directly to the buffer without any additional processing.

Parameters

<i>method</i>	HTTP method to use for the request (GET, POST, PUT, DELETE, etc.) See HttpMethod enumeration for all supported methods
<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>headers</i>	Pointer to null-terminated string containing custom HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr or empty string if no custom headers are needed

Parameters

<i>result</i>	Pointer to raw buffer where the HTTP response body will be stored Buffer must be pre-allocated with sufficient space for the response
<i>maxl</i>	Maximum length of the result buffer in bytes Response will be truncated if it exceeds this length
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

Number of bytes received and stored in the result buffer on success (≥ 0), or a negative error code on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The result buffer will contain the raw response body data. No null termination is added - treat as binary data. Custom headers must be properly formatted without trailing CRLF on the final header. The response may be truncated if it exceeds the *maxl* parameter.

See also

HttpMethod for available HTTP methods

[DoHttpEx\(HttpMethod, ParsedURI&, const char*, unsigned char*, int, uint16_t\)](#) for pre-parsed URI version

[DoHttpEx\(HttpMethod, const char*, const char*, buffer_object&, uint16_t\)](#) for [buffer_object](#) version

14.60.2.25 DoHttpEx() [4/6]

```
bool DoHttpEx (
    HttpMethod method,
    ParsedURI & TheUri,
    const char * headers,
    buffer_object & result_buffer,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
#include <http_funcs.h>
```

Executes an HTTP request with custom headers using a pre-parsed URI and stores the response in a buffer object. This is the core extended HTTP client function that performs an HTTP request with the specified method using a pre-parsed URI object and allows custom headers to be included in the request. The complete response body is stored in the provided buffer object. This function provides maximum flexibility and control over HTTP requests, supporting all standard HTTP methods, custom headers, and pre-parsed URIs for optimal performance when making multiple requests.

Parameters

<i>method</i>	HTTP method to use for the request (GET, POST, PUT, DELETE, etc.) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, fragment, etc.)
<i>headers</i>	Pointer to null-terminated string containing custom HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr or empty string if no custom headers are needed
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, connection error, invalid method, malformed headers, or HTTP error response)

Note

The `result_buffer` contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. Headers must be properly formatted without trailing CRLF on the final header. This is the most comprehensive HTTP client function providing full control. For HTTPS URLs, SSL/TLS connection will be established automatically. Using pre-parsed URIs improves performance for repeated requests to the same host.

See also

[HttpMethod](#) for available HTTP methods

[ParsedURI](#) for URL parsing and component access

[buffer_object](#) for response buffer management

[DoHttp\(\)](#) for simpler requests without custom headers

[DoGetEx\(\)](#) for GET-specific convenience wrapper

[DoHttpEx\(HttpMethod, ParsedURI&, const char*, NBString&, uint16_t\)](#) for [NBString](#) overload

14.60.2.26 DoHttpEx() [5/6]

```
bool DoHttpEx (
    HttpMethod method,
    ParsedURI & TheUri,
    const char * headers,
    NBString & result_string,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP request with custom headers using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using a pre-parsed URI object and allows custom headers to be included in the request. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying [DoHttpEx](#) implementation that accepts a [buffer_object](#).

Parameters

<i>method</i>	HTTP method to use for the request (GET, POST, PUT, DELETE, etc.) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>headers</i>	Pointer to null-terminated string containing custom HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no custom headers are needed
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The `result_string` contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Custom headers must be properly formatted with CRLF line endings. This is the most flexible HTTP client function supporting all methods and custom headers.

See also

[DoHttpEx\(HttpMethod, ParsedURI&, const char*, buffer_object&, uint16_t\)](#) for the underlying implementation
[ParsedURI](#) for URL parsing and component access
[StringBufferObject](#) for response handling details
[HttpMethod](#) for available HTTP methods
[DoHttp\(\)](#) for simpler requests without custom headers
[DoGetEx\(\)](#) for GET-specific convenience function

14.60.2.27 DoHttpEx() [6/6]

```
int DoHttpEx (
    HttpMethod method,
    ParsedURI & TheUri,
    const char * headers,
    unsigned char * result,
    int maxl,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
#include <http_funcs.h>
```

Executes an HTTP request with custom headers using a pre-parsed URI and stores the response in a raw buffer. This is a low-level extended HTTP client function that performs an HTTP request with the specified method and custom headers using a pre-parsed URI object. The response body is stored in a provided raw buffer. This function provides optimal performance through pre-parsed URI components while supporting custom headers for authentication, content negotiation, or other advanced HTTP operations. The response is written directly to the buffer without any additional processing.

Parameters

<i>method</i>	HTTP method to use for the request (GET, POST, PUT, DELETE, etc.) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>headers</i>	Pointer to null-terminated string containing custom HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr or empty string if no custom headers are needed
<i>result</i>	Pointer to raw buffer where the HTTP response body will be stored Buffer must be pre-allocated with sufficient space for the response
<i>maxl</i>	Maximum length of the result buffer in bytes Response will be truncated if it exceeds this length
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

Number of bytes received and stored in the result buffer on success (≥ 0), or a negative error code on failure (network error, timeout, connection error, or HTTP error response)

Note

The result buffer will contain the raw response body data. No null termination is added - treat as binary data. Custom headers must be properly formatted without trailing CRLF on the final header. Using pre-parsed URIs improves performance for repeated requests to the same host. The response may be truncated if it exceeds the `maxl` parameter.

See also

`HttpMethod` for available HTTP methods

`ParsedURI` for URL parsing and component access

`DoHttpEx(HttpMethod, const char*, const char*, unsigned char*, int, uint16_t)` for string URL version

`DoHttpEx(HttpMethod, ParsedURI&, const char*, buffer_object&, uint16_t)` for `buffer_object` version

14.60.2.28 DoJsonHttp() [1/8]

```
bool DoJsonHttp (
    HttpMethod method,
    const char * pUrl,
    const char * Json_Data_To_Post,
    buffer_object & result_buffer,
    const char * AdditionalHeaders = NULL,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
#include <http_funcs.h>
```

Executes an HTTP request with raw JSON data using a URL string and stores the response in a buffer object. This is the core JSON HTTP function for raw JSON string data that performs an HTTP request with the specified method using raw JSON data provided as a string and a URL string. The complete response body is stored in the provided buffer object. This function provides the foundation for JSON HTTP operations with pre-formatted JSON strings, supporting all standard HTTP methods while handling URL parsing and DNS resolution automatically. The JSON data is sent with the appropriate Content-Type header for proper server processing.

Parameters

<i>method</i>	HTTP method to use for the request (POST, PUT, PATCH recommended for JSON data) See <code>HttpMethod</code> enumeration for all supported methods
<i>pUrl</i>	Pointer to null-terminated URL string (<code>http://</code> or <code>https://</code>) DNS lookup will be performed automatically for hostnames
<i>Json_Data_To_Post</i>	Pointer to null-terminated string containing the raw JSON data to be sent in the request body. Must be valid, serialized JSON format
<i>result_buffer</i>	Reference to <code>buffer_object</code> where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Multiple headers should be concatenated and separated by <code>"\r\n"</code> Do not include trailing <code>"\r\n"</code> after the last header Pass <code>nullptr</code> if no additional headers are needed (default: <code>NULL</code>)
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use <code>TICKS_PER_SECOND</code> for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The `result_buffer` contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. Content-Type header is automatically set to "application/json" for the request. The provided JSON string must be properly formatted, serialized, and null-terminated.

See also

[HttpMethod](#) for available HTTP methods

[buffer_object](#) for response buffer management

[DoJsonHttp\(HttpMethod, ParsedURI&, const char*, buffer_object&, const char*, uint16_t\)](#) for pre-parsed URI version

[DoJsonPost\(const char*, const char*, buffer_object&, const char*, uint16_t\)](#) for POST-specific convenience wrapper

14.60.2.29 DoJsonHttp() [2/8]

```
bool DoJsonHttp (
    HttpMethod method,
    const char * pUrl,
    const char * Json_Data_To_Post,
    NSString & result_string,
    const char * AdditionalHeaders = NULL,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
```

```
#include <http_funcs.h>
```

Executes an HTTP request with raw JSON data using a URL string and stores the response in an [NSString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using raw JSON data provided as a string and a URL string. The complete response body is stored in the provided [NSString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoJsonHttp implementation. The URL string will be parsed internally and DNS resolution will be performed automatically if needed, making this ideal for simple JSON operations with pre-formatted data.

Parameters

<i>method</i>	HTTP method to use for the request (POST, PUT, PATCH recommended for JSON data) See HttpMethod enumeration for all supported methods
<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>Json_Data_To_Post</i>	Pointer to null-terminated string containing the raw JSON data to be sent in the request body. Must be valid JSON format
<i>result_string</i>	Reference to NSString in which to store the HTTP response body The string will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr if no additional headers are needed (default: NULL)
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The result_string contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Content-Type header is automatically set to "application/json" for the request. The provided JSON string must be properly formatted and null-terminated.

See also

[DoJsonHttp\(HttpMethod, const char*, const char*, buffer_object&, const char*, uint16_t\)](#) for the underlying implementation

[HttpMethod](#) for available HTTP methods

[StringBufferObject](#) for response handling details

[DoJsonHttp\(HttpMethod, ParsedUri&, const char*, NString&, const char*, uint16_t\)](#) for pre-parsed URI version

14.60.2.30 DoJsonHttp() [3/8]

```
bool DoJsonHttp (
    HttpMethod method,
    const char * pUrl,
    ParsedJsonDataSet & jsonout,
    buffer_object & result_buffer,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
```

```
#include <http_funcs.h>
```

Executes an HTTP request with JSON data from a [ParsedJsonDataSet](#) using a URL string and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is the core JSON HTTP function that performs an HTTP request with the specified method using JSON data from a [ParsedJsonDataSet](#) object and a URL string. The complete response body is stored in the provided buffer object. This function provides the foundation for programmatic JSON HTTP operations with string-based URLs, allowing dynamic construction and manipulation of JSON data before transmission. The URL string will be parsed internally and DNS resolution will be performed automatically if needed.

Parameters

<i>method</i>	HTTP method to use for the request (POST, PUT, PATCH recommended for JSON data) See HttpMethod enumeration for all supported methods
<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>jsonout</i>	Reference to ParsedJsonDataSet containing the JSON data to be serialized and sent in the request body
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no additional headers are needed
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, JSON serialization error, or HTTP error response)

Note

The result_buffer contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. The JSON data will be automatically serialized with Content-Type "application/json".

See also

[HttpMethod](#) for available HTTP methods

[ParsedJsonDataSet](#) for JSON data construction and management

[buffer_object](#) for response buffer management

[DoJsonHttp\(HttpMethod, ParsedUri&, ParsedJsonDataSet&, buffer_object&, const char*, uint16_t\)](#) for pre-parsed URI version

[JSON Lexer](#)

14.60.2.31 DoJsonHttp() [4/8]

```
bool DoJsonHttp (
    HttpMethod method,
    const char * pUrl,
    ParsedJsonDataSet & jsonout,
    NBString & result_string,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP request with JSON data from a [ParsedJsonDataSet](#) using a URL string and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using JSON data from a [ParsedJsonDataSet](#) object and a URL string. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoJsonHttp implementation. This provides flexibility for programmatic JSON operations with string-based response handling and simplified URL specification.

Parameters

<i>method</i>	HTTP method to use for the request (POST, PUT, PATCH recommended for JSON data) See HttpMethod enumeration for all supported methods
<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>jsonout</i>	Reference to ParsedJsonDataSet containing the JSON data to be serialized and sent in the request body
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no additional headers are needed
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, JSON serialization error, or HTTP error response)

Note

The result_string contents will be replaced with the response data. For large responses, ensure sufficient memory is available. The JSON data will be automatically serialized with Content-Type "application/json".

See also

[DoJsonHttp\(HttpMethod, const char*, ParsedJsonDataSet&, buffer_object&, const char*, uint16_t\)](#) for the underlying implementation

[HttpMethod](#) for available HTTP methods

[ParsedJsonDataSet](#) for JSON data construction and management

[StringBufferObject](#) for response handling details

14.60.2.32 DoJsonHttp() [5/8]

```
bool DoJsonHttp (
    HttpMethod method,
    ParsedURI & TheUri,
    const char * Json_Data_To_Post,
    buffer_object & result_buffer,
    const char * AdditionalHeaders = NULL,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
```

```
#include <http_funcs.h>
```

Executes an HTTP request with raw JSON data using a pre-parsed URI and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is the core JSON HTTP function for raw JSON string data that performs an HTTP request with the specified method using raw JSON data provided as a string and a pre-parsed URI object. The complete response body is stored in the provided buffer object. This function provides the foundation for JSON HTTP operations with pre-formatted JSON strings, offering optimal performance through pre-parsed URI components while supporting all standard HTTP methods.

Parameters

<i>method</i>	HTTP method to use for the request (POST, PUT, PATCH recommended for JSON data) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>Json_Data_To_Post</i>	Pointer to null-terminated string containing the raw JSON data to be sent in the request body. Must be valid JSON format
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr if no additional headers are needed (default: NULL)
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The result_buffer contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. Content-Type header is automatically set to "application/json" for the request. The provided JSON string must be properly formatted and null-terminated. Using pre-parsed URIs improves performance for repeated requests to the same host.

See also

[HttpMethod](#) for available HTTP methods

[ParsedURI](#) for URL parsing and component access

[buffer_object](#) for response buffer management

[DoJsonHttp\(HttpMethod, const char*, const char*, buffer_object&, const char*, uint16_t\)](#) for string URL version

14.60.2.33 DoJsonHttp() [6/8]

```
bool DoJsonHttp (
    HttpMethod method,
    ParsedURI & TheUri,
    const char * Json_Data_To_Post,
    NBString & result_string,
    const char * AdditionalHeaders = NULL,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
```

```
#include <http_funcs.h>
```

Executes an HTTP request with raw JSON data using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using raw JSON data provided as a string and a pre-parsed URI object. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoJsonHttp implementation. This provides flexibility for JSON operations with pre-formatted JSON strings and string-based response handling.

Parameters

<i>method</i>	HTTP method to use for the request (POST, PUT, PATCH recommended for JSON data) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>Json_Data_To_Post</i>	Pointer to null-terminated string containing the raw JSON data to be sent in the request body. Must be valid JSON format
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr if no additional headers are needed (default: NULL)
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The result_string contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Content-Type header is automatically set to "application/json" for the request. The provided JSON string must be properly formatted and null-terminated.

See also

[DoJsonHttp\(HttpMethod, ParsedUri&, const char*, buffer_object&, const char*, uint16_t\)](#) for the underlying implementation

[HttpMethod](#) for available HTTP methods

[ParsedUri](#) for URL parsing and component access

[StringBufferObject](#) for response handling details

14.60.2.34 DoJsonHttp() [7/8]

```
bool DoJsonHttp (
    HttpMethod method,
    ParsedUri & TheUri,
    ParsedJsonDataSet & jsonout,
    buffer_object & result_buffer,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
```

```
#include <http_funcs.h>
```

Executes an HTTP request with JSON data from a [ParsedJsonDataSet](#) using a pre-parsed URI and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is the core JSON HTTP function that performs an HTTP request with the specified method using JSON data from a [ParsedJsonDataSet](#) object and a pre-parsed URI. The complete response body is stored in the provided buffer object. This function provides the foundation for programmatic JSON HTTP operations, allowing dynamic construction and manipulation of JSON data before transmission. The [ParsedJsonDataSet](#) will be automatically serialized to JSON format for the request body.

Parameters

<i>method</i>	HTTP method to use for the request (POST, PUT, PATCH recommended for JSON data) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedUri object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>jsonout</i>	Reference to ParsedJsonDataSet containing the JSON data to be serialized and sent in the request body
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no additional headers are needed
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, connection error, JSON serialization error, or HTTP error response)

Note

The *result_buffer* contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. The JSON data will be automatically serialized with Content-Type "application/json". Using pre-parsed URIs improves performance for repeated requests to the same host.

See also

[HttpMethod](#) for available HTTP methods

[ParsedUri](#) for URL parsing and component access

[ParsedJsonDataSet](#) for JSON data construction and management

[buffer_object](#) for response buffer management

14.60.2.35 DoJsonHttp() [8/8]

```
bool DoJsonHttp (
    HttpMethod method,
    ParsedUri & TheUri,
    ParsedJsonDataSet & jsonout,
    NBString & result_string,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP request with JSON data from a [ParsedJsonDataSet](#) using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using JSON data from a [ParsedJsonDataSet](#) object and a pre-parsed URI. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoJsonHttp implementation. This provides flexibility for programmatic JSON operations with string-based response handling.

Parameters

<i>method</i>	HTTP method to use for the request (POST, PUT, PATCH recommended for JSON data) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedUri object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>jsonout</i>	Reference to ParsedJsonDataSet containing the JSON data to be serialized and sent in the request body
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no additional headers are needed
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, connection error, JSON serialization error, or HTTP error response)

Note

The result_string contents will be replaced with the response data. For large responses, ensure sufficient memory is available. The JSON data will be automatically serialized with Content-Type "application/json".

See also

[DoJsonHttp\(HttpMethod, ParsedURI&, ParsedJsonDataSet&, buffer_object&, const char*, uint16_t\)](#) for the underlying implementation

[HttpMethod](#) for available HTTP methods

[ParsedURI](#) for URL parsing and component access

[ParsedJsonDataSet](#) for JSON data construction and management

[StringBufferObject](#) for response handling details

14.60.2.36 DoJsonHttpFile() [1/4]

```
bool DoJsonHttpFile (
    HttpMethod method,
    const char * pUrl,
    const char * FragmentName,
    buffer_object & result_buffer,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
#include <http_funcs.h>
```

Executes an HTTP request with JSON data from a file fragment using a URL string and stores the response in a buffer object.

This is the core JSON HTTP file function that performs an HTTP request with the specified method using JSON data loaded from a file fragment and a URL string. The complete response body is stored in the provided buffer object. This function supports JSON file templates with dynamic variable substitution, function callbacks, and other advanced features similar to the NetBurner web server system. The URL string will be parsed internally and DNS resolution will be performed automatically if needed.

Parameters

<i>method</i>	HTTP method to use for the request (POST, PUT, PATCH recommended for JSON data) See HttpMethod enumeration for all supported methods
<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>FragmentName</i>	Pointer to null-terminated string specifying the name of the file fragment containing the JSON template data with optional variable substitutions
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr or empty string if no additional headers are needed
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, fragment not found, template processing error, or HTTP error response)

Note

The result_buffer contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. The JSON file fragment supports template processing with variable substitution. Content-Type header is automatically set to "application/json" for the request.

See also

[HttpMethod](#) for available HTTP methods

[buffer_object](#) for response buffer management

[DoJsonHttpFile\(HttpMethod, ParsedURI&, const char*, buffer_object&, const char*, uint16_t\)](#) for pre-parsed URI version

14.60.2.37 DoJsonHttpFile() [2/4]

```
bool DoJsonHttpFile (
    HttpMethod method,
    const char * pUrl,
    const char * FragmentName,
    NBString & result_string,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Performs an HTTP request and stores JSON response in an [NBString](#) This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that wraps the result [NBString](#) in a [StringBufferObject](#) and calls the main DoJsonHttpFile function.

Parameters

<i>method</i>	HTTP method to use (GET, POST, PUT, DELETE, etc.)
<i>pUrl</i>	URL to send the HTTP request to
<i>FragmentName</i>	Name of the JSON fragment to extract from response
<i>result_string</i>	Reference to NBString in which to store the result
<i>AdditionalHeaders</i>	Additional HTTP headers to include in the request (can be NULL)
<i>TIMEOUT_WAIT</i>	Timeout value in system ticks (default: 10 seconds)

Returns

true if the HTTP request was successful and JSON was parsed, false otherwise

See also

[DoJsonHttpFile\(HttpMethod, const char*, const char*, StringBufferObject&, const char*, uint16_t\)](#)

14.60.2.38 DoJsonHttpFile() [3/4]

```
bool DoJsonHttpFile (
    HttpMethod method,
    ParsedURI & TheUri,
    const char * FragmentName,
    buffer_object & result_buffer,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND)
#include <http_funcs.h>
```

Sends an HTTP request and processes JSON response using a pre-parsed URI.

This function performs HTTP operations with JSON response processing using a pre-parsed URI object, providing more control and efficiency when the same URI will be used multiple times. It supports various HTTP methods (Post, Put, Patch) commonly used in REST API operations for creating, updating, and modifying resources. The function establishes an HTTP connection to the specified URI, sends the request with any additional headers, waits for the server response within the specified timeout period, and then parses the JSON response to extract only the requested fragment. This selective extraction reduces memory usage and processing overhead when dealing

with large JSON responses. The use of a pre-parsed URI object eliminates repeated URI parsing overhead and provides better performance for repeated requests to the same endpoint. The function handles all aspects of the HTTP transaction including error conditions, timeouts, and malformed responses.

Parameters

<i>method</i>	HTTP method to use (Post, Put, or Patch)
<i>TheUri</i>	Reference to the pre-parsed URI object containing host, port, and path information
<i>FragmentName</i>	Name of the JSON fragment to extract from the response
<i>result_buffer</i>	Reference to the buffer object where the result will be stored
<i>AdditionalHeaders</i>	Additional HTTP headers to include in the request (may be NULL)
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for response (default: 10 seconds)

Returns

true on success, false on failure

14.60.2.39 DoJsonHttpFile() [4/4]

```
bool DoJsonHttpFile (
    HttpMethod method,
    ParsedURI & TheUri,
    const char * FragmentName,
    NBString & result_string,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP request with JSON data from a file fragment using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using JSON data loaded from a file fragment and a pre-parsed URI object. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoJsonHttpFile implementation. This approach provides maximum flexibility for JSON-based HTTP operations with pre-stored data fragments.

Parameters

<i>method</i>	HTTP method to use for the request (GET, POST, PUT, DELETE, etc.) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>FragmentName</i>	Pointer to null-terminated string specifying the name of the file fragment containing the JSON data to be sent in the request body
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no additional headers are needed
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, connection error, fragment not found, invalid method, or HTTP error response)

Note

The `result_string` contents will be replaced with the response data. For large responses, ensure sufficient memory is available. The specified file fragment must exist and contain valid JSON data. Content-Type header is automatically set to "application/json" for the request.

See also

[DoJsonHttpFile\(HttpMethod, ParsedURI&, const char*, buffer_object&, const char*, uint16_t\)](#) for the underlying implementation

[HttpMethod](#) for available HTTP methods

[ParsedURI](#) for URL parsing and component access

[StringBufferObject](#) for response handling details

[DoJsonPostHttpFile\(\)](#) for POST-specific convenience wrapper

14.60.2.40 DoJsonPost() [1/8]

```
bool DoJsonPost (
    const char * pUrl,
    const char * Json_Data_To_Post,
    buffer_object & result_buffer,
    const char * AdditionalHeaders = NULL,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP POST request with raw JSON data using a URL string and stores the response in a buffer object. This function performs an HTTP POST request using raw JSON data provided as a string and a URL string. The complete response body is stored in the provided buffer object. This is a convenience wrapper around the generic `DoJsonHttp` function, specifically optimized for POST operations with pre-formatted JSON strings. This provides the most straightforward interface for JSON POST operations when you have serialized JSON data and want to use automatic URL parsing with buffer-based response handling.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>Json_Data_To_Post</i>	Pointer to null-terminated string containing the raw JSON data to be sent in the POST request body. Must be valid, serialized JSON format
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr if no additional headers are needed (default: NULL)
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use <code>TICKS_PER_SECOND</code> for conversion

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The `result_buffer` contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. Content-Type header is automatically set to "application/json" for the request. The provided JSON string must be properly formatted, serialized, and null-terminated.

See also

[DoJsonHttp\(\)](#) for the underlying generic method implementation

[buffer_object](#) for response buffer management

[DoJsonPost\(ParsedURI&, const char*, buffer_object&, const char*, uint16_t\)](#) for pre-parsed URI version

[DoJsonPost\(const char*, const char*, NBString&, const char*, uint16_t\)](#) for [NBString](#) version

14.60.2.41 DoJsonPost() [2/8]

```
bool DoJsonPost (
    const char * pUrl,
    const char * Json_Data_To_Post,
    NBString & result_string,
    const char * AdditionalHeaders = NULL,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP POST request with raw JSON data using a URL string and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using raw JSON data provided as a string and a URL string. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying [DoJsonHttp](#) implementation. This provides the simplest interface for JSON POST operations when you have pre-formatted JSON data and want string-based response handling with automatic URL parsing.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>Json_Data_To_Post</i>	Pointer to null-terminated string containing the raw JSON data to be sent in the POST request body. Must be valid JSON format
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr if no additional headers are needed (default: NULL)
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The *result_string* contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Content-Type header is automatically set to "application/json" for the request. The provided JSON string must be properly formatted and null-terminated.

See also

[DoJsonHttp\(HttpMethod, const char*, const char*, NBString&, const char*, uint16_t\)](#) for the underlying generic method implementation

[StringBufferObject](#) for response handling details

[DoJsonPost\(ParsedURI&, const char*, NBString&, const char*, uint16_t\)](#) for pre-parsed URI version

14.60.2.42 DoJsonPost() [3/8]

```
bool DoJsonPost (
    const char * pUrl,
    ParsedJsonDataSet & jsonout,
    buffer_object & result_buffer,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP POST request with JSON data from a [ParsedJsonDataSet](#) using a URL string and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using JSON data from a [ParsedJsonDataSet](#) object and a URL string. The complete response body is stored in the provided buffer object. This function is a simple wrapper around the generic DoJsonHttp function, specifically optimized for POST operations with programmatically constructed JSON data, providing a streamlined interface for common JSON API interactions.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>jsonout</i>	Reference to ParsedJsonDataSet containing the JSON data to be serialized and sent in the POST request body
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no additional headers are needed
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, JSON serialization error, or HTTP error response)

Note

The result_buffer contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. The JSON data will be automatically serialized with Content-Type "application/json".

See also

[DoJsonHttp\(\)](#) for the underlying generic method implementation

[ParsedJsonDataSet](#) for JSON data construction and management

[buffer_object](#) for response buffer management

[DoJsonPost\(ParsedURI&, ParsedJsonDataSet&, buffer_object&, const char*, uint16_t\)](#) for pre-parsed URI version

[JSON Lexer](#)

14.60.2.43 DoJsonPost() [4/8]

```
bool DoJsonPost (
    const char * pUrl,
    ParsedJsonDataSet & jsonout,
```

```

        NBString & result_string,
        const char * AdditionalHeaders,
        uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>

```

Executes an HTTP POST request with JSON data from a [ParsedJsonDataSet](#) using a URL string and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using JSON data from a [ParsedJsonDataSet](#) object and a URL string. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoJsonHttp implementation. This provides a simple interface for JSON POST operations with programmatically constructed data and string-based response handling.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>jsonout</i>	Reference to ParsedJsonDataSet containing the JSON data to be serialized and sent in the POST request body
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no additional headers are needed
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, JSON serialization error, or HTTP error response)

Note

The result_string contents will be replaced with the response data. For large responses, ensure sufficient memory is available. The JSON data will be automatically serialized with Content-Type "application/json".

See also

[DoJsonHttp\(HttpMethod, const char*, ParsedJsonDataSet&, NBString&, const char*, uint16_t\)](#) for the underlying generic method implementation

[ParsedJsonDataSet](#) for JSON data construction and management

[StringBufferObject](#) for response handling details

[DoJsonPost\(ParsedURI&, ParsedJsonDataSet&, NBString&, const char*, uint16_t\)](#) for pre-parsed URI version

[JSON Lexer](#)

14.60.2.44 DoJsonPost() [5/8]

```

bool DoJsonPost (
    ParsedURI & TheUri,
    const char * Json_Data_To_Post,
    buffer_object & result_buffer,
    const char * AdditionalHeaders = NULL,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>

```

Executes an HTTP POST request with raw JSON data using a pre-parsed URI and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using raw JSON data provided as a string and a pre-parsed URI object. The complete response body is stored in the provided buffer object. This function is a simple wrapper around the generic DoJsonHttp function, specifically optimized for POST operations with pre-formatted JSON strings, providing a streamlined interface when you already have JSON data in string format.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>Json_Data_To_Post</i>	Pointer to null-terminated string containing the raw JSON data to be sent in the POST request body. Must be valid JSON format
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr if no additional headers are needed (default: NULL)
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The result_buffer contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. Content-Type header is automatically set to "application/json" for the request. The provided JSON string must be properly formatted and null-terminated.

See also

[DoJsonHttp\(\)](#) for the underlying generic method implementation

[ParsedURI](#) for URL parsing and component access

[buffer_object](#) for response buffer management

[DoJsonPost\(ParsedURI&, ParsedJsonDataSet&, buffer_object&, const char*, uint16_t\)](#) for [ParsedJsonDataSet](#) version

14.60.2.45 DoJsonPost() [6/8]

```
bool DoJsonPost (
    ParsedURI & TheUri,
    const char * Json_Data_To_Post,
    NBString & result_string,
    const char * AdditionalHeaders = NULL,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
```

```
#include <http_funcs.h>
```

Executes an HTTP POST request with raw JSON data using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using raw JSON data provided as a string and a pre-parsed URI object. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoJsonHttp implementation. This provides a simple interface for JSON POST operations with pre-formatted JSON strings and string-based response handling.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>Json_Data_To_Post</i>	Pointer to null-terminated string containing the raw JSON data to be sent in the POST request body. Must be valid JSON format
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr if no additional headers are needed (default: NULL)
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The *result_string* contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Content-Type header is automatically set to "application/json" for the request. The provided JSON string must be properly formatted and null-terminated.

See also

[DoJsonHttp\(HttpMethod, ParsedURI&, const char*, NBString&, const char*, uint16_t\)](#) for the underlying generic method implementation

[ParsedURI](#) for URL parsing and component access

[StringBufferObject](#) for response handling details

[DoJsonPost\(ParsedURI&, ParsedJsonDataSet&, NBString&, const char*, uint16_t\)](#) for [ParsedJsonDataSet](#) version

14.60.2.46 DoJsonPost() [7/8]

```
bool DoJsonPost (
    ParsedURI & TheUri,
    ParsedJsonDataSet & jsonout,
    buffer_object & result_buffer,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP POST request with JSON data from a [ParsedJsonDataSet](#) using a pre-parsed URI and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using JSON data from a [ParsedJsonDataSet](#) object and a pre-parsed URI. The complete response body is stored in the provided buffer object. This function is a simple wrapper around the generic DoJsonHttp function, specifically optimized for POST operations with programmatically constructed JSON data, providing a streamlined interface for common JSON API interactions.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>jsonout</i>	Reference to ParsedJsonDataSet containing the JSON data to be serialized and sent in the POST request body

Parameters

<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no additional headers are needed
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, connection error, JSON serialization error, or HTTP error response)

Note

The *result_buffer* contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. The JSON data will be automatically serialized with Content-Type "application/json".

See also

[DoJsonHttp\(\)](#) for the underlying generic method implementation
[ParsedUri](#) for URL parsing and component access
[ParsedJsonDataSet](#) for JSON data construction and management
[buffer_object](#) for response buffer management

14.60.2.47 DoJsonPost() [8/8]

```
bool DoJsonPost (
    ParsedUri & TheUri,
    ParsedJsonDataSet & jsonout,
    NBString & result_string,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP POST request with JSON data from a [ParsedJsonDataSet](#) using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using JSON data from a [ParsedJsonDataSet](#) object and a pre-parsed URI. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoJsonHttp implementation. This approach allows for dynamic JSON POST requests with programmatically constructed data.

Parameters

<i>TheUri</i>	Reference to ParsedUri object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>jsonout</i>	Reference to ParsedJsonDataSet containing the JSON data to be sent in the POST request body
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no additional headers are needed

<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use <i>TICKS_PER_SECOND</i> for conversion

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, connection error, JSON serialization error, or HTTP error response)

Note

The *result_string* contents will be replaced with the response data. For large responses, ensure sufficient memory is available. The JSON data will be automatically serialized and sent with Content-Type "application/json".

See also

[DoJsonHttp\(\)](#) for the underlying JSON HTTP implementation

[ParsedUri](#) for URL parsing and component access

[ParsedJsonDataSet](#) for JSON data management

[StringBufferObject](#) for response handling details

14.60.2.48 DoJsonPostHttpFile() [1/4]

```
bool DoJsonPostHttpFile (
    const char * pUrl,
    const char * FragmentName,
    buffer_object & result_buffer,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Executes an HTTP POST request with JSON data from a file fragment using a URL string and stores the response in a buffer object.

This function performs an HTTP POST request using JSON data loaded from a specified file fragment and a URL string. The complete response body is stored in the provided buffer object. This is a convenience wrapper around the generic [DoJsonHttpFile](#) function, providing a simple interface for JSON POST operations with file-based templates that support dynamic variable substitution, function callbacks, and other advanced processing features similar to the NetBurner web server system.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>FragmentName</i>	Pointer to null-terminated string specifying the name of the file fragment containing the JSON template data with optional variable substitutions
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr or empty string if no additional headers are needed
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use <i>TICKS_PER_SECOND</i> for conversion

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, fragment not found, template processing error, or HTTP error response)

Note

The `result_buffer` contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. The JSON file fragment supports template processing with variable substitution. Content-Type header is automatically set to "application/json" for the request.

See also

[DoJsonHttpFile\(\)](#) for the underlying generic method implementation

[buffer_object](#) for response buffer management

[DoJsonPostHttpFile\(ParsedURI&, const char*, buffer_object&, const char*, uint16_t\)](#) for pre-parsed URI version

14.60.2.49 DoJsonPostHttpFile() [2/4]

```
bool DoJsonPostHttpFile (
    const char * pUrl,
    const char * FragmentName,
    NBString & result_string,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Performs an HTTP POST request and stores JSON response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience wrapper function that simplifies HTTP POST operations with JSON response handling. It internally calls `DoJsonHttpFile` with `HttpMethod::Post` and automatically manages the buffer conversion from [NBString](#) to [StringBufferObject](#). The function handles the complete HTTP transaction including connection establishment, request transmission, response reception, and JSON parsing. Only the specified JSON fragment is extracted from the response and stored in the result string, making it efficient for retrieving specific data from larger JSON responses. The function is particularly useful for REST API interactions where you need to POST data and extract a specific piece of information from the JSON response.

Parameters

<i>pUrl</i>	URL to send the HTTP POST request to
<i>FragmentName</i>	Name of the JSON fragment to extract from response
<i>result_string</i>	Reference to NBString where the result will be stored
<i>AdditionalHeaders</i>	Additional HTTP headers to include in the request (may be NULL)
<i>TIMEOUT_WAIT</i>	Timeout value in system ticks (default: 10 seconds)

Returns

true if the HTTP POST request succeeded and JSON was parsed successfully, false otherwise

See also

[DoJsonHttpFile\(HttpMethod, const char*, const char*, NBString&, const char*, uint16_t\)](#)

14.60.2.50 DoJsonPostHttpFile() [3/4]

```
bool DoJsonPostHttpFile (
    ParsedURI & TheUri,
    const char * FragmentName,
    buffer_object & result_buffer,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
```

```
#include <http_funcs.h>
```

Executes an HTTP POST request with JSON data from a file fragment using a pre-parsed URI and stores the response in a buffer object.

This function performs an HTTP POST request using JSON data loaded from a specified file fragment and a pre-parsed URI object. The complete response body is stored in the provided buffer object. This is a convenience wrapper around the generic DoJsonHttpFile function, specifically optimized for POST operations with pre-defined JSON data stored in file fragments, eliminating the need to construct JSON payloads programmatically.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>FragmentName</i>	Pointer to null-terminated string specifying the name of the file fragment containing the JSON data to be sent in the POST request body
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no additional headers are needed
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, connection error, fragment not found, or HTTP error response)

Note

The result_buffer contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. The specified file fragment must exist and contain valid JSON data. Content-Type header is automatically set to "application/json" for the request.

See also

[DoJsonHttpFile\(\)](#) for the underlying JSON HTTP implementation

[ParsedURI](#) for URL parsing and component access

[buffer_object](#) for response buffer management

[DoJsonPostHttpFile\(ParsedURI&, const char*, NBString&, const char*, uint16_t\)](#) for NBString overload

14.60.2.51 DoJsonPostHttpFile() [4/4]

```
bool DoJsonPostHttpFile (
    ParsedURI & TheUri,
    const char * FragmentName,
    NBString & result_string,
    const char * AdditionalHeaders,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```


Executes an HTTP POST request with JSON data from a file fragment using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request using JSON data loaded from a specified file fragment and a pre-parsed URI object. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoJsonHttpFile implementation. This approach allows for efficient JSON POST requests with pre-defined data stored in file fragments.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>FragmentName</i>	Pointer to null-terminated string specifying the name of the file fragment containing the JSON data to be sent in the POST request body
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>AdditionalHeaders</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr or empty string if no additional headers are needed
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, connection error, fragment not found, or HTTP error response)

Note

The result_string contents will be replaced with the response data. For large responses, ensure sufficient memory is available. The specified file fragment must exist and contain valid JSON data. Content-Type header is automatically set to "application/json" for the request.

See also

[DoJsonHttpFile\(\)](#) for the underlying JSON HTTP implementation

[ParsedURI](#) for URL parsing and component access

[StringBufferObject](#) for response handling details

[DoJsonPostHttpFile\(ParsedURI&, const char*, buffer_object&, const char*, uint16_t\)](#) for [buffer_object](#) overload

14.60.2.52 DoMultipartBoundary()

```
void DoMultipartBoundary (
    int tcpfd,
    const char * Disposition,
    const char * separator)
#include <http_funcs.h>
```

Sends a multipart boundary section within an active multipart HTTP request.

This function sends a multipart boundary with content disposition headers but without any data payload as part of a multipart HTTP request. This is useful for creating empty form fields or marking section boundaries within the multipart stream according to the multipart/form-data specification. The boundary is properly formatted with the specified separator and disposition information.

Parameters

<i>tcpfd</i>	TCP file descriptor of the active HTTP connection Must be a valid descriptor returned from DoMultipartStartPost()
<i>Disposition</i>	Content-Disposition header value for this multipart section Should follow RFC format (e.g., "form-data; name=\"fieldname\"") Used to identify this section even though no data follows
<i>separator</i>	Multipart boundary separator string used to delimit sections Must be the same separator used throughout the entire multipart request

Note

This function must be called between [DoMultipartStartPost\(\)](#) and [DoMultipartFinished\(\)](#). The TCP connection must remain active throughout the entire multipart sequence. The separator string must be consistent across all multipart operations in the same request. This creates a boundary marker without associated data content.

See also

[DoMultipartStartPost\(\)](#) to initiate a multipart request
[DoMultipartItem\(\)](#) to send boundary sections with data
[DoMultipartFinished\(\)](#) to complete the request and receive the response

14.60.2.53 DoMultipartFinished() [1/2]

```
bool DoMultipartFinished (
    int tcpfd,
    const char * separator,
    buffer_object & result_buffer,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)

#include <http_funcs.h>
```

Completes a multipart HTTP request and retrieves the server response in a buffer object.

This function finalizes a multipart HTTP request by sending the closing boundary, waiting for the server response, and parsing the complete response into the provided buffer object. This must be called after all multipart items and boundaries have been sent using [DoMultipartItem\(\)](#) and [DoMultipartBoundary\(\)](#) functions. The function handles the proper termination of the multipart stream and manages the HTTP response parsing.

Parameters

<i>tcpfd</i>	TCP file descriptor of the active HTTP connection Must be a valid descriptor returned from DoMultipartStartPost()
<i>separator</i>	Multipart boundary separator string used throughout the request Must match the separator used in all previous multipart operations
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete server response
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the server response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the request completed successfully with a 200 OK response containing a body, false on failure (network error, timeout, non-200 response, or parsing error)

Note

This function closes the multipart request and invalidates the TCP file descriptor. The *result_buffer* contents will be replaced with the server response data. Ensure the buffer object has sufficient capacity for the expected response size. A successful return indicates both proper multipart transmission and a valid HTTP response.

See also

[DoMultipartStartPost\(\)](#) to initiate the multipart request sequence
[DoMultipartItem\(\)](#) and [DoMultipartBoundary\(\)](#) for sending multipart data
[buffer_object](#) for response buffer management

14.60.2.54 DoMultipartFinished() [2/2]

```
bool DoMultipartFinished (
    int tcpfd,
    const char * separator,
    NBString & result_string,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND) [inline]
#include <http_funcs.h>
```

Completes a multipart HTTP request and retrieves the server response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. This is a convenience overload that finalizes a multipart HTTP request and stores the server response in an [NBString](#) reference. The function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoMultipartFinished implementation. This provides string-based response handling for multipart operations where text-based responses are expected.

Parameters

<i>tcpfd</i>	TCP file descriptor of the active HTTP connection Must be a valid descriptor returned from DoMultipartStartPost()
<i>separator</i>	Multipart boundary separator string used throughout the request Must match the separator used in all previous multipart operations
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete server response
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the server response (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

true if the request completed successfully with a 200 OK response containing a body, false on failure (network error, timeout, non-200 response, or parsing error)

Note

This function closes the multipart request and invalidates the TCP file descriptor. The result_string contents will be replaced with the server response data. For large responses, ensure sufficient memory is available. A successful return indicates both proper multipart transmission and a valid HTTP response.

See also

[DoMultipartFinished\(int, const char*, buffer_object&, uint16_t\)](#) for the underlying implementation
[StringBufferObject](#) for response handling details
[DoMultipartStartPost\(\)](#) to initiate the multipart request sequence

14.60.2.55 DoMultipartItem()

```
void DoMultipartItem (
    int tcpfd,
    const char * Disposition,
    const char * separator,
```

```

        const unsigned char * data,
        int len)
#include <http_funcs.h>

```

Sends a multipart data item within an active multipart HTTP request.

This function sends a single data item as part of a multipart HTTP request that was previously initiated with [DoMultipartStartPost\(\)](#). The function handles the proper formatting of the multipart boundary, content disposition headers, and the actual data payload according to the multipart/form-data specification. This allows for sending binary data, files, or form fields as individual parts of a larger multipart submission.

Parameters

<i>tcpfd</i>	TCP file descriptor of the active HTTP connection Must be a valid descriptor returned from DoMultipartStartPost()
<i>Disposition</i>	Content-Disposition header value for this multipart section Should follow RFC format (e.g., "form-data; name=\"fieldname\"" or "form-data; name=\"file\"; filename=\"example.txt\"")
<i>separator</i>	Multipart boundary separator string used to delimit sections Must be the same separator used throughout the entire multipart request
<i>data</i>	Pointer to the binary data to be sent in this multipart section Can contain any type of data including binary files or text
<i>len</i>	Length of the data in bytes Must accurately represent the size of the data buffer

Note

This function must be called between [DoMultipartStartPost\(\)](#) and [DoMultipartFinished\(\)](#). The TCP connection must remain active throughout the entire multipart sequence. The separator string must be consistent across all multipart operations in the same request. Content-Type headers for individual parts can be included in the Disposition parameter.

See also

- [DoMultipartStartPost\(\)](#) to initiate a multipart request
- [DoMultipartBoundary\(\)](#) to send boundary-only sections
- [DoMultipartFinished\(\)](#) to complete the request and receive the response

14.60.2.56 DoMultipartStartHttp() [1/2]

```

int DoMultipartStartHttp (
    HttpMethod method,
    const char * pUrl,
    const char * separator,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND,
    uint32_t contentLength = 0)
#include <http_funcs.h>

```

Initiates a multipart HTTP request using a URL string and returns a connection file descriptor.

This is the core multipart HTTP function that establishes an HTTP connection and begins a multipart request with the specified method using a URL string. The function performs URL parsing, DNS resolution if needed, establishes the TCP/TLS connection, and sends the initial HTTP headers including the multipart/form-data content type with the specified boundary separator. The returned file descriptor is used for subsequent multipart operations.

Parameters

<i>method</i>	HTTP method to use for the multipart request (POST, PUT, PATCH recommended) See HttpMethod enumeration for all supported methods
<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames For HTTPS URLs, SSL/TLS connection will be established automatically

Parameters

<i>separator</i>	Multipart boundary separator string used to delimit sections Must be unique and not appear in the actual data being transmitted Commonly generated as a random string or UUID
<i>TIMEOUT_WAIT</i>	Timeout in system ticks for connection establishment and initial request (default: 10 seconds). Use TICKS_PER_SECOND for conversion
<i>contentLength</i>	Total content length for the entire multipart request in bytes If 0, chunked transfer encoding will be used (default: 0) If specified, must accurately represent the total payload size

Returns

TCP/TLS file descriptor (> 0) if connection was successfully established and headers sent, or a negative tcpSocketStatus error code on failure (connection error, DNS resolution failure, timeout, or HTTP protocol error)

Note

The returned file descriptor must be used with [DoMultipartItem\(\)](#), [DoMultipartBoundary\(\)](#), and [DoMultipartFinished\(\)](#) to complete the multipart request sequence. The connection remains active until [DoMultipartFinished\(\)](#) is called. If contentLength is specified, the total data sent must match exactly. The separator string must be consistent throughout all subsequent multipart operations.

See also

[HttpMethod](#) for available HTTP methods

[DoMultipartItem\(\)](#) to send data items

[DoMultipartBoundary\(\)](#) to send boundary markers

[DoMultipartFinished\(\)](#) to complete the request and receive response

[TCP Socket Status](#) for error code definitions

14.60.2.57 DoMultipartStartHttp() [2/2]

```
int DoMultipartStartHttp (
    HttpMethod method,
    ParsedURI & TheUri,
    const char * separator,
    uint16_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND,
    uint32_t contentLength = 0)
#include <http_funcs.h>
```

Initiates a multipart HTTP request using a pre-parsed URI and returns a connection file descriptor.

This is the core multipart HTTP function that establishes an HTTP connection and begins a multipart request with the specified method using a pre-parsed URI object. The function uses the pre-parsed URI components to establish the TCP/TLS connection directly, eliminating URL parsing overhead and providing optimal performance for repeated multipart requests to the same host. The function sends the initial HTTP headers including the multipart/form-data content type with the specified boundary separator.

Parameters

<i>method</i>	HTTP method to use for the multipart request (POST, PUT, PATCH recommended) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.) For HTTPS schemes, SSL/TLS connection will be established automatically
<i>separator</i>	Multipart boundary separator string used to delimit sections Must be unique and not appear in the actual data being transmitted Commonly generated as a random string or UUID

Parameters

<i>TIMEOUT_WAIT</i>	Timeout in system ticks for connection establishment and initial request (default: 10 seconds). Use TICKS_PER_SECOND for conversion
<i>contentLength</i>	Total content length for the entire multipart request in bytes. If 0, chunked transfer encoding will be used (default: 0). If specified, must accurately represent the total payload size

Returns

TCP/TLS file descriptor (> 0) if connection was successfully established and headers sent, or a negative tcpSocketStatus error code on failure (connection error, timeout, or HTTP protocol error)

Note

The returned file descriptor must be used with [DoMultipartItem\(\)](#), [DoMultipartBoundary\(\)](#), and [DoMultipartFinished\(\)](#) to complete the multipart request sequence. Using pre-parsed URLs improves performance for repeated requests to the same host. The connection remains active until [DoMultipartFinished\(\)](#) is called. If contentLength is specified, the total data sent must match exactly. The separator string must be consistent throughout all subsequent multipart operations.

See also

[HttpMethod](#) for available HTTP methods

[ParsedUri](#) for URL parsing and component access

[DoMultipartItem\(\)](#) to send data items

[DoMultipartBoundary\(\)](#) to send boundary markers

[DoMultipartFinished\(\)](#) to complete the request and receive response

[TCP Socket Status](#) for error code definitions

[DoMultipartStartHttp\(HttpMethod, const char*, const char*, uint16_t, uint32_t\)](#) for string URL version

14.60.2.58 DoMultipartStartPost() [1/2]

```
int DoMultipartStartPost (
    const char * pUrl,
    const char * separator,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND,
    uint32_t contentLength = 0) [inline]
#include <http_funcs.h>
```

Initiates a multipart HTTP POST request using a URL string and returns a connection file descriptor.

This is a convenience function that establishes an HTTP POST connection and begins a multipart request using a URL string. This function provides the simplest interface for starting multipart POST operations, handling URL parsing, DNS resolution, connection establishment, and sending the initial HTTP headers with multipart/form-data content type. This is ideal for straightforward multipart file uploads or form submissions.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames. For HTTPS URLs, SSL/TLS connection will be established automatically.
<i>separator</i>	Multipart boundary separator string used to delimit sections. Must be unique and not appear in the actual data being transmitted. Commonly generated as a random string or UUID.
<i>TIMEOUT_WAIT</i>	Timeout in system ticks for connection establishment and initial request (default: 10 seconds). Use TICKS_PER_SECOND for conversion.
<i>contentLength</i>	Total content length for the entire multipart request in bytes. If 0, chunked transfer encoding will be used (default: 0). If specified, must accurately represent the total payload size.

Returns

TCP/TLS file descriptor (> 0) if connection was successfully established and headers sent, or a negative `tcpSocketStatus` error code on failure (connection error, DNS resolution failure, timeout, or HTTP protocol error)

Note

The returned file descriptor must be used with [DoMultipartItem\(\)](#), [DoMultipartBoundary\(\)](#), and [DoMultipartFinished\(\)](#) to complete the multipart request sequence. If `contentLength` is specified, the total data sent must match exactly. The separator string must be consistent throughout all subsequent multipart operations.

See also

[DoMultipartStartHttp\(\)](#) for the underlying generic method implementation

[DoMultipartItem\(\)](#) to send data items

[DoMultipartBoundary\(\)](#) to send boundary markers

[DoMultipartFinished\(\)](#) to complete the request and receive response

[TCP Socket Status](#) for error code definitions

14.60.2.59 DoMultipartStartPost() [2/2]

```
int DoMultipartStartPost (
    ParsedURI & TheUri,
    const char * separator,
    uint16_t TIMEOUT_WAIT = 10 * TICKS_PER_SECOND,
    uint32_t contentLength = 0) [inline]
```

```
#include <http_funcs.h>
```

Initiates a multipart HTTP POST request using a pre-parsed URI and returns a connection file descriptor.

This is a convenience function that establishes an HTTP POST connection and begins a multipart request using a pre-parsed URI object. This function provides enhanced performance by using pre-parsed URI components, eliminating URL parsing overhead when making multiple multipart requests to the same host. The function establishes the connection and sends the initial HTTP headers with multipart/form-data content type.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>separator</i>	Multipart boundary separator string used to delimit sections. Must be unique and not appear in the actual data being transmitted. Commonly generated as a random string or UUID.
<i>TIMEOUT_WAIT</i>	Timeout in system ticks for connection establishment and initial request (default: 10 seconds). Use <code>TICKS_PER_SECOND</code> for conversion.
<i>contentLength</i>	Total content length for the entire multipart request in bytes. If 0, chunked transfer encoding will be used (default: 0). If specified, must accurately represent the total payload size.

Returns

TCP/TLS file descriptor (> 0) if connection was successfully established and headers sent, or a negative `tcpSocketStatus` error code on failure (connection error, timeout, or HTTP protocol error)

Note

The returned file descriptor must be used with [DoMultipartItem\(\)](#), [DoMultipartBoundary\(\)](#), and [DoMultipartFinished\(\)](#) to complete the multipart request sequence. Using pre-parsed URIs improves performance for repeated requests to the same host. If `contentLength` is specified, the total data sent must match exactly. The separator string must be consistent throughout all subsequent multipart operations.

See also

[DoMultipartStartHttp\(\)](#) for the underlying generic method implementation
[ParsedURI](#) for URL parsing and component access
[DoMultipartItem\(\)](#) to send data items
[DoMultipartBoundary\(\)](#) to send boundary markers
[DoMultipartFinished\(\)](#) to complete the request and receive response
[TCP Socket Status](#) for error code definitions

14.60.2.60 DoUrlEncodedFormHttp() [1/3]

```

bool DoUrlEncodedFormHttp (
    HttpMethod method,
    const char * pUrl,
    char * headers,
    char * form_data,
    buffer_object & result_buffer,
    uint16_t TIMEOUT_WAIT)
  
```

```
#include <http_funcs.h>
```

Executes an HTTP request with URL-encoded form data using a URL string and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is the core URL-encoded form HTTP function that performs an HTTP request with the specified method using URL-encoded form data and a URL string. The complete response body is stored in the provided buffer object. This function provides the foundation for form-based HTTP operations, supporting all standard HTTP methods while handling URL parsing and DNS resolution automatically. The form data is sent with the appropriate Content-Type header for proper server processing of web form submissions.

Parameters

<i>method</i>	HTTP method to use for the request (POST recommended for form data) See HttpMethod enumeration for all supported methods
<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>headers</i>	Pointer to null-terminated string containing HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr if no custom headers are needed
<i>form_data</i>	Pointer to null-terminated string containing URL-encoded form data Data should be in "key1=value1&key2=value2" format with proper URL encoding
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response Use TICKS_PER_SECOND for conversion from seconds to ticks

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The result_buffer contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. Content-Type header is automatically set to "application/x-www-form-urlencoded". Form data must be properly URL-encoded before passing to this function.

See also

[HttpMethod](#) for available HTTP methods

[buffer_object](#) for response buffer management

[DoUrlEncodedFormHttp\(HttpMethod, ParsedURI&, char*, char*, buffer_object&, uint16_t\)](#) for pre-parsed URI version

[DoUrlEncodedFormPost\(const char*, char*, char*, buffer_object&, uint16_t\)](#) for POST-specific convenience wrapper

14.60.2.61 DoUrlEncodedFormHttp() [2/3]

```
bool DoUrlEncodedFormHttp (
    HttpMethod method,
    ParsedURI & TheUri,
    char * headers,
    char * form_data,
    buffer_object & result_buffer,
    uint16_t TIMEOUT_WAIT)
```

```
#include <http_funcs.h>
```

Executes an HTTP request with URL-encoded form data using a pre-parsed URI and stores the response in a buffer object.

This is the core URL-encoded form HTTP function that performs an HTTP request with the specified method using URL-encoded form data and a pre-parsed URI object. The complete response body is stored in the provided buffer object. This function provides the foundation for form-based HTTP operations with optimal performance through pre-parsed URI components, supporting all standard HTTP methods while eliminating the overhead of repeated URL parsing when making multiple form submissions to the same host.

Parameters

<i>method</i>	HTTP method to use for the request (POST, PUT, PATCH recommended for form data) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>headers</i>	Pointer to null-terminated string containing additional HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr if no additional headers are needed
<i>form_data</i>	Pointer to null-terminated string containing URL-encoded form data Data should be in "key1=value1&key2=value2" format with proper URL encoding
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response Use TICKS_PER_SECOND for conversion from seconds to ticks

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The result_buffer contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. Content-Type header is automatically set to "application/x-www-form-urlencoded". Form data must be properly URL-encoded before passing to this function. Using pre-parsed URIs improves performance for repeated requests to the same host.

See also

[HttpMethod](#) for available HTTP methods

[ParsedURI](#) for URL parsing and component access

[buffer_object](#) for response buffer management

[DoUrlEncodedFormHttp\(HttpMethod, const char*, char*, char*, buffer_object&, uint16_t\)](#) for string URL version

14.60.2.62 DoUrlEncodedFormHttp() [3/3]

```
bool DoUrlEncodedFormHttp (
    HttpMethod method,
    ParsedURI & TheUri,
    char * headers,
    char * form_data,
    NBString & result_string,
    uint16_t TIMEOUT_WAIT) [inline]
```

```
#include <http_funcs.h>
```

Executes an HTTP request with URL-encoded form data using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP request with the specified method using URL-encoded form data and a pre-parsed URI object. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying DoUrlEncodedFormHttp implementation. This provides flexibility for form-based HTTP operations with pre-parsed URIs and string-based response handling.

Parameters

<i>method</i>	HTTP method to use for the request (POST, PUT, PATCH recommended for form data) See HttpMethod enumeration for all supported methods
<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>headers</i>	Pointer to null-terminated string containing additional HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr if no additional headers are needed
<i>form_data</i>	Pointer to null-terminated string containing URL-encoded form data Data should be in "key1=value1&key2=value2" format with proper URL encoding
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response Use TICKS_PER_SECOND for conversion from seconds to ticks

Returns

true if the HTTP request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The result_string contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Content-Type header is automatically set to "application/x-www-form-urlencoded". Form data must be properly URL-encoded before passing to this function. Using pre-parsed URIs improves performance for repeated requests to the same host.

See also

[DoUrlEncodedFormHttp\(HttpMethod, ParsedURI&, char*, char*, buffer_object&, uint16_t\)](#) for the underlying implementation

[HttpMethod](#) for available HTTP methods

[ParsedURI](#) for URL parsing and component access

[StringBufferObject](#) for response handling details

14.60.2.63 DoUrlEncodedFormPost() [1/4]

```
bool DoUrlEncodedFormPost (
    const char * pUrl,
    char * headers,
    char * form_data,
    buffer_object & result_buffer,
    uint16_t TIMEOUT_WAIT) [inline]
```

```
#include <http_funcs.h>
```

Executes an HTTP POST request with URL-encoded form data using a URL string and stores the response in a buffer object. This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request with URL-encoded form data using a URL string. The complete response body is stored in the provided buffer object. This function is a simple wrapper around the generic [DoUrlEncodedFormHttp](#) function, specifically optimized for POST operations with form data. This provides the most straightforward interface for web form submissions when you have URL-encoded form data and want to use automatic URL parsing with buffer-based response handling.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>headers</i>	Pointer to null-terminated string containing HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr if no custom headers are needed
<i>form_data</i>	Pointer to null-terminated string containing URL-encoded form data Data should be in "key1=value1&key2=value2" format with proper URL encoding
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response Use TICKS_PER_SECOND for conversion from seconds to ticks

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The *result_buffer* contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. Content-Type header is automatically set to "application/x-www-form-urlencoded". Form data must be properly URL-encoded before passing to this function.

See also

[DoUrlEncodedFormHttp\(\)](#) for the underlying generic method implementation

[buffer_object](#) for response buffer management

[DoUrlEncodedFormPost\(ParsedURI&, char*, char*, buffer_object&, uint16_t\)](#) for pre-parsed URI version

[DoUrlEncodedFormPost\(const char*, char*, char*, NBString&, uint16_t\)](#) for [NBString](#) version

14.60.2.64 DoUrlEncodedFormPost() [2/4]

```
bool DoUrlEncodedFormPost (
    const char * pUrl,
    char * headers,
    char * form_data,
    NSString & result_string,
    uint16_t TIMEOUT_WAIT) [inline]
```

```
#include <http_funcs.h>
```

Executes an HTTP POST request with URL-encoded form data using a URL string and stores the response in an [NSString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request with URL-encoded form data using a URL string. The complete response body is stored in the provided [NSString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying [DoUrlEncodedFormHttp](#) implementation. This provides a simple interface for form-based POST operations with string-based response handling.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>headers</i>	Pointer to null-terminated string containing HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr if no custom headers are needed
<i>form_data</i>	Pointer to null-terminated string containing URL-encoded form data Data should be in "key1=value1&key2=value2" format
<i>result_string</i>	Reference to NSString in which to store the HTTP response body The string will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response Use TICKS_PER_SECOND for conversion from seconds to ticks

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, DNS resolution failure, or HTTP error response)

Note

The *result_string* contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Content-Type header is automatically set to "application/x-www-form-urlencoded". Form data must be properly URL-encoded before passing to this function.

See also

[DoUrlEncodedFormHttp\(\)](#) for the underlying generic method implementation

[StringBufferObject](#) for response handling details

[DoUrlEncodedFormPost\(const char*, char*, char*, buffer_object&, uint16_t\)](#) for *buffer_object* version

14.60.2.65 DoUrlEncodedFormPost() [3/4]

```
bool DoUrlEncodedFormPost (
    ParsedURI & TheUri,
    char * headers,
    char * form_data,
    buffer_object & result_buffer,
    uint16_t TIMEOUT_WAIT) [inline]
#include <http_funcs.h>
```

Executes an HTTP POST request with URL-encoded form data using a pre-parsed URI and stores the response in a buffer object.

This function performs an HTTP POST request with URL-encoded form data using a pre-parsed URI object. The complete response body is stored in the provided buffer object. This is a convenience wrapper around the generic `DoUrlEncodedFormHttp` function, specifically optimized for POST operations with form data. This provides enhanced performance for web form submissions by using pre-parsed URI components, eliminating URL parsing overhead for repeated submissions to the same endpoint.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>headers</i>	Pointer to null-terminated string containing additional HTTP headers Multiple headers should be concatenated and separated by "\r\n" Do not include trailing "\r\n" after the last header Pass nullptr if no additional headers are needed
<i>form_data</i>	Pointer to null-terminated string containing URL-encoded form data Data should be in "key1=value1&key2=value2" format with proper URL encoding
<i>result_buffer</i>	Reference to buffer_object where the HTTP response body will be stored The buffer will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response Use TICKS_PER_SECOND for conversion from seconds to ticks

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The `result_buffer` contents will be replaced with the response data. Ensure the buffer object has sufficient capacity for the expected response size. Content-Type header is automatically set to "application/x-www-form-urlencoded". Form data must be properly URL-encoded before passing to this function. Using pre-parsed URIs improves performance for repeated requests to the same host.

See also

[DoUrlEncodedFormHttp\(\)](#) for the underlying generic method implementation

[ParsedURI](#) for URL parsing and component access

[buffer_object](#) for response buffer management

[DoUrlEncodedFormPost\(const char*, char*, char*, buffer_object&, uint16_t\)](#) for string URL version

14.60.2.66 DoUrlEncodedFormPost() [4/4]

```
bool DoUrlEncodedFormPost (
    ParsedURI & TheUri,
    char * headers,
    char * form_data,
    NBString & result_string,
    uint16_t TIMEOUT_WAIT) [inline]
```

```
#include <http_funcs.h>
```

Executes an HTTP POST request with URL-encoded form data using a pre-parsed URI and stores the response in an [NBString](#). This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is a convenience overload that performs an HTTP POST request with URL-encoded form data using a pre-parsed URI object. The complete response body is stored in the provided [NBString](#) reference. This function uses a [StringBufferObject](#) internally to handle the response data collection and forwards the call to the underlying `DoUrlEncodedFormHttp` implementation. Using a pre-parsed URI can improve performance when making multiple form submissions to the same host by avoiding repeated URL parsing operations.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the pre-parsed URL components (host, port, path, scheme, query parameters, etc.)
<i>headers</i>	Pointer to null-terminated string containing HTTP headers Headers should be formatted as "Header: Value\r\n" pairs Pass nullptr if no custom headers are needed
<i>form_data</i>	Pointer to null-terminated string containing URL-encoded form data Data should be in "key1=value1&key2=value2" format
<i>result_string</i>	Reference to NBString in which to store the HTTP response body The string will be populated with the complete response data
<i>TIMEOUT_WAIT</i>	Timeout in system ticks to wait for the complete response Use TICKS_PER_SECOND for conversion from seconds to ticks

Returns

true if the POST request completed successfully and response was received, false on failure (network error, timeout, connection error, or HTTP error response)

Note

The *result_string* contents will be replaced with the response data. For large responses, ensure sufficient memory is available. Content-Type header is automatically set to "application/x-www-form-urlencoded". Form data must be properly URL-encoded before passing to this function. Using pre-parsed URIs improves performance for repeated requests to the same host.

See also

[DoUrlEncodedFormHttp\(\)](#) for the underlying generic method implementation

[ParsedURI](#) for URL parsing and component access

[StringBufferObject](#) for response handling details

[DoUrlEncodedFormPost\(const char*, char*, char*, NBString&, uint16_t\)](#) for string URL version

14.60.2.67 PopulateAuthHeader()

```
int PopulateAuthHeader (
    const char * user,
    const char * password,
    char * buffer,
    int maxlen)
```

```
#include <http_funcs.h>
```

Generates a Base64-encoded HTTP Basic Authentication header string for inclusion in HTTP requests.

This function creates a properly formatted HTTP Basic Authentication header by combining the provided username and password, encoding them using Base64, and formatting the result as an "Authorization: Basic" header string. The generated header can be used directly in HTTP requests that require Basic Authentication. The function handles the complete authentication header construction according to RFC 7617 specifications.

Parameters

<i>user</i>	Pointer to null-terminated string containing the username Must not contain colon characters as they are used as delimiters
<i>password</i>	Pointer to null-terminated string containing the password Can be empty string for username-only authentication scenarios
<i>buffer</i>	Pointer to destination buffer where the formatted authentication header will be written Buffer will contain "Authorization: Basic <base64-credentials>" format
<i>maxlen</i>	Maximum number of characters that can be written to the buffer including null terminator Must be sufficient to hold the complete formatted header string

Returns

Number of characters written to the buffer (excluding null terminator) on success, or a negative value if the buffer was insufficient to hold the complete header string

Note

The generated header string is null-terminated and ready for direct use in HTTP requests. The buffer must be large enough to accommodate the Base64-encoded credentials plus header formatting. Base64 encoding increases the size of the credentials by approximately 33%. The function does not validate credential format - ensure username does not contain colons. This implements HTTP Basic Authentication which transmits credentials in encoded but not encrypted form.

See also

[WriteHttpRequestHeaders\(\)](#) for incorporating the generated header into HTTP requests

RFC 7617 for HTTP Basic Authentication specification details

14.60.2.68 SetHttpDiag() [1/2]

```
void SetHttpDiag (
    bool b)
#include <http_funcs.h>
Enable/disable Web Client HTTP diagnostics to the console port.
```

Parameters

<i>b</i>	Set true to enable, false to disable.

14.60.2.69 SetHttpDiag() [2/2]

```
void SetHttpDiag (
    int level)
#include <http_funcs.h>
Enable/disable Web Client HTTP diagnostics to the console port.
```

Parameters

<i>level</i>	Set 1 to enable, 0 to disable, or 2 for extra verbosity.

14.60.2.70 StartConnection()

```
int StartConnection (
    ParsedURI & TheUri,
    uint32_t timeout = TICKS_PER_SECOND*10) [inline]
#include <http_funcs.h>
```

Establishes a TCP/TLS connection to a host using a pre-parsed URI with timeout specified in ticks.

This is a convenience overload that establishes a connection to the host specified in the pre-parsed URI object using a timeout value expressed in system ticks. The function creates a TickTimeout object internally and forwards the call to the underlying StartConnection implementation. This provides a simple interface for connection establishment when you need to specify timeout in system ticks rather than using a TickTimeout object directly.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the connection details (host, port, scheme for HTTP/HTTPS selection)
<i>timeout</i>	Connection timeout in system ticks (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

TCP/TLS file descriptor (> 0) if connection was successfully established, or a negative error code on failure (connection refused, timeout, DNS resolution failure, or network error)

Note

For HTTPS URLs, SSL/TLS connection will be established automatically. The returned file descriptor must be properly closed when no longer needed. Connection timeout includes both DNS resolution and TCP connection establishment time.

See also

[StartConnection\(ParsedURI&, TickTimeout&\)](#) for the underlying implementation

[ParsedURI](#) for URL parsing and component access

[TickTimeout](#) for timeout management

14.60.2.71 StartTransaction() [1/2]

```
int StartTransaction (
    const char * pUrl,
    const char * headers = 0,
    const char * action = "GET",
    uint32_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
#include <http_funcs.h>
```

Initiates an HTTP transaction using a URL string by establishing a connection and sending request headers. This function performs the initial phase of an HTTP request by parsing the provided URL, establishing a connection to the specified host, and sending the HTTP request headers including the request line, Host header, and any additional custom headers. The function returns a file descriptor for the established connection that can be used to send request body data (if needed) and receive the server response. This provides a simple interface for starting HTTP transactions with automatic URL parsing and DNS resolution.

Parameters

<i>pUrl</i>	Pointer to null-terminated URL string (http:// or https://) DNS lookup will be performed automatically for hostnames
<i>headers</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted without leading or trailing "\r\n" Each header should be on a separate line with "\r\n" between headers Pass nullptr if no additional headers are needed (default: nullptr)
<i>action</i>	Pointer to null-terminated string containing the HTTP method (default: "GET"). Common values: "GET", "POST", "PUT", "DELETE"
<i>TIMEOUT_WAIT</i>	Timeout in system ticks for URL parsing, connection establishment, and header transmission (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

TCP/TLS file descriptor (> 0) if connection was established and headers sent successfully, or a negative error code on failure (URL parsing error, DNS resolution failure, connection error, timeout, or header transmission failure)

Note

The returned file descriptor can be used to send request body data and receive responses. The Host header is automatically included based on the parsed URL information. For HTTPS URLs, SSL/TLS connection will be established automatically. The connection remains active until explicitly closed. Custom headers must be properly formatted without leading/trailing CRLF.

See also

[StartTransaction\(ParsedURI&, const char*, const char*, uint32_t\)](#) for pre-parsed URI version

14.60.2.72 StartTransaction() [2/2]

```
int StartTransaction (
    ParsedURI & TheUri,
    const char * headers = 0,
    const char * action = "GET",
    uint32_t TIMEOUT_WAIT = 10 *TICKS_PER_SECOND)
#include <http_funcs.h>
```

Initiates an HTTP transaction using a pre-parsed URI by establishing a connection and sending request headers. This function performs the initial phase of an HTTP request by establishing a connection to the specified host and sending the HTTP request headers including the request line, Host header, and any additional custom headers. The function returns a file descriptor for the established connection that can be used to send request body data (if needed) and receive the server response. This provides low-level control over the HTTP transaction lifecycle with optimal performance through pre-parsed URI components.

Parameters

<i>TheUri</i>	Reference to ParsedURI object containing the connection and request details (host, port, path, scheme, query parameters)
<i>headers</i>	Pointer to null-terminated string containing additional HTTP headers Headers should be formatted without leading or trailing "\r\n" Each header should be on a separate line with "\r\n" between headers Pass nullptr if no additional headers are needed (default: nullptr)
<i>action</i>	Pointer to null-terminated string containing the HTTP method (default: "GET"). Common values: "GET", "POST", "PUT", "DELETE"
<i>TIMEOUT_WAIT</i>	Timeout in system ticks for connection establishment and header transmission (default: 10 seconds). Use TICKS_PER_SECOND for conversion

Returns

TCP/TLS file descriptor (> 0) if connection was established and headers sent successfully, or a negative error code on failure (connection error, timeout, or header transmission failure)

Note

The returned file descriptor can be used to send request body data and receive responses. The Host header is automatically included based on the URI information. For HTTPS URLs, SSL/TLS connection will be established automatically. The connection remains active until explicitly closed. Custom headers must be properly formatted without leading/trailing CRLF.

See also

[ParsedURI](#) for URL parsing and component access

[StartTransaction\(const char*, const char*, const char*, uint32_t\)](#) for string URL version

14.60.2.73 StartWebClient() [1/3]

```
bool StartWebClient (
    int prio,
    const char * url1,
    const char * url2 = NULL,
    bool bDoNtp = false)
#include <web_client.h>
```

Start the web client using a URL string.

Parameters

<i>prio</i>	Priority

<i>url1</i>	Pointer to a string representing the URL.
<i>url2</i>	Pointer to a second URL. Optional parameter, default is NULL.
<i>bDoNtp</i>	Enable the Network Time Protocol (NTP). Optional parameter, default is false.

Returns

True on success, false on failure.

14.60.2.74 StartWebClient() [2/3]

```
bool StartWebClient (  
    int prio,  
    const NSString & url1,  
    bool bDoNtp = false)  
#include <web_client.h>  
Start the web client using a NSString.
```

Parameters

<i>prio</i>	Priority
<i>url1</i>	Reference to a NSString representing the URL.
<i>bDoNtp</i>	Enable the Network Time Protocol (NTP). Optional parameter, default is false.

Returns

True on success, false on failure.

14.60.2.75 StartWebClient() [3/3]

```
bool StartWebClient (  
    int prio,  
    const NSString & url1,  
    const NSString & url2,  
    bool bDoNtp = false)  
#include <web_client.h>  
Start the web client using a NSString.
```

Parameters

<i>prio</i>	Priority
<i>url1</i>	Reference to a string representing the URL.
<i>url2</i>	Reference to a second URL. Optional parameter, default is NULL.
<i>bDoNtp</i>	Enable the Network Time Protocol (NTP). Optional parameter, default is false.

Returns

True on success, false on failure.

14.60.2.76 WriteHttpRequestHeaders()

```
int WriteHttpRequestHeaders (
    int fd,
    ParsedURI & uri,
    const char * methodStr,
    int content_len = 0,
    const char * content_type = 0,
    const char * extra_header = 0)
#include <http_funcs.h>
```

Writes HTTP request headers to an established connection file descriptor.

This function writes properly formatted HTTP request headers to an established TCP/TLS connection using information from a parsed URI object. The function constructs and sends the HTTP request line, Host header, and any additional headers as specified. This is typically used after establishing a connection but before sending any request body data. The headers are formatted according to HTTP/1.1 specification requirements.

Parameters

<i>fd</i>	TCP/TLS file descriptor of an established connection Must be a valid descriptor returned from connection establishment functions
<i>uri</i>	Reference to ParsedURI object containing the request details (host, port, path, query parameters for request construction)
<i>methodStr</i>	Pointer to null-terminated string containing the HTTP method (e.g., "GET", "POST", "PUT", "DELETE")
<i>content_len</i>	Content length for the request body in bytes If 0, no Content-Length header is added (default: 0) If > 0, Content-Length header is automatically included
<i>content_type</i>	Pointer to null-terminated string containing the Content-Type header value If nullptr, no Content-Type header is added (default: nullptr) Common values include "application/json", "application/x-www-form-urlencoded"
<i>extra_header</i>	Pointer to null-terminated string containing additional custom headers Multiple headers should be separated by "\r\n" If nullptr, no extra headers are added (default: nullptr)

Returns

Number of bytes written to the connection on success (> 0), or a negative error code on failure (write error, connection closed, or formatting error)

Note

The function automatically adds required HTTP/1.1 headers including Host header. Headers are properly terminated with "\r\n\r\n" to indicate end of header section. The connection must be established before calling this function. This function only writes headers - request body must be sent separately if needed. Custom headers in *extra_header* parameter must be properly formatted.

See also

StartConnection() to establish the initial connection
[ParsedURI](#) for URL parsing and component access

14.60.3 Web Client Error Codes

Macros

- `#define WEB_CLIENT_ERROR_NO_ETHERNET (1)`
No Ethernet connection.
- `#define WEB_CLIENT_ERROR_NO_ADDRESS (2)`
Interface has no IP address.

- `#define WEB_CLIENT_ERROR_NO_GATEWAY (3)`
Interface has no gateway IP address.
- `#define WEB_CLIENT_ERROR_GATEWAY_WRONG (4)`
Interface has incorrect gateway IP address.
- `#define WEB_CLIENT_ERROR_NO_DNS_ADDR (5)`
Interface has a DNS error.
- `#define WEB_CLIENT_ERROR_NO_DNS_RESOLVE (6)`
DNS was not able to resolve the URL name.
- `#define WEB_CLIENT_ERROR_NO_NTP (7)`
NTP failed.
- `#define WEB_CLIENT_ERROR_NO_SERVER_RESPONSE (8)`
No response from server.
- `#define WEB_CLIENT_ERROR_NO_SERVER_CONNECT (9)`
Unable to connect to server.
- `#define WEB_CLIENT_ERROR_NO_ERROR (10)`
No error.
- `#define WEB_CLIENT_ERROR_INVALID_REQ (11)`
Invalid request.

14.60.3.1 Detailed Description

```
#include< web_client.h>
```

Web Client error codes

14.61 Web Sockets

Web Socket functions and definitions.

Topics

- [Web Socket Definitions](#)
WebSocket Protocol Implementation for Real-Time Bidirectional Communication `#include< websockets.h>`

Classes

- class [NB::WebSocket](#)
WebSocket connection management class.

Functions

- int [WSPing](#) (int fd, uint32_t len, uint32_t *sentTick)
Send a WebSocket ping frame to test connection liveness and measure round-trip time.
- int [WSGetPingReplyTick](#) (int fd, uint32_t *replyTick)
Retrieve the timestamp of the most recent pong reply received on a WebSocket connection.
- int [WSWaitForPingReply](#) (int fd, uint32_t timeout)
Wait for a pong reply to a previously sent ping with a specified timeout.
- int [WSUpgrade](#) ([HTTP_Request](#) *req, int sock)
Upgrade an HTTP connection to a WebSocket connection.

14.61.1 Detailed Description

Web Socket functions and definitions.

14.61.2 Function Documentation

14.61.2.1 WSGetPingReplyTick()

```
int WSGetPingReplyTick (
    int fd,
    uint32_t * replyTick)
```

```
#include <websockets.h>
```

Retrieve the timestamp of the most recent pong reply received on a WebSocket connection.

Gets the system tick count when the last pong frame was received in response to a ping. This function is used in conjunction with [WSPing\(\)](#) to measure round-trip time and verify connection responsiveness. The reply tick is automatically recorded by the WebSocket implementation when a pong frame arrives.

This is useful for:

- Calculating network latency after sending a ping
- Verifying that a connection is still responsive
- Implementing timeout detection for unresponsive connections
- Gathering connection quality metrics

Parameters

<i>fd</i>	The file descriptor of the WebSocket connection
<i>replyTick</i>	Pointer to store the tick count when the last pong was received. If no pong has been received yet, the value is undefined.

Returns

Returns a status code indicating success or failure:

- 0 or positive: Success, replyTick contains the timestamp of the last pong
- Negative value: Error occurred (invalid fd, no pong received, connection closed, etc.)

Note

This function returns the timestamp of the most recently received pong frame

Call this after sending a ping with [WSPing\(\)](#) to measure round-trip time

The returned tick value is in system ticks (use TICKS_PER_SECOND to convert)

If no pong has been received since the connection opened, behavior is undefined

Compare with the sentTick from [WSPing\(\)](#) to calculate latency

The pong reply is automatically processed by the WebSocket stack

Multiple pings can be in flight, but this function only returns the most recent pong timestamp.

Expand for Example Usage

Examples

Example 1: Basic Round-Trip Time Measurement

```
int wsFd = ...; // WebSocket connection
uint32_t pingSent, pongReceived;

// Send ping
if (WSPing(wsFd, 0, &pingSent) >= 0) {
    printf("Ping sent at tick %lu\r\n", pingSent);

    // Wait a moment for the pong
    OSTimeDly(TICKS_PER_SECOND / 10); // 100ms

    // Get pong reply time
    if (WSGetPingReplyTick(wsFd, &pongReceived) >= 0) {
        uint32_t rtt = pongReceived - pingSent;
        uint32_t rttMs = rtt / (TICKS_PER_SECOND / 1000);
        printf("Round-trip time: %lu ms\r\n", rttMs);
    } else {
        printf("No pong received yet\r\n");
    }
}
```

Example 2: Polling for Pong Response

```
#define PONG_TIMEOUT_MS 3000

bool WaitForPong(int wsFd, uint32_t pingSentTick, uint32_t *latencyMs) {
    uint32_t startWait = TimeTick;
    uint32_t pongTick;

    while (1) {
        // Check if pong received
        if (WSGetPingReplyTick(wsFd, &pongTick) >= 0) {
            // Verify this pong is for our ping
            if (pongTick > pingSentTick) {
                uint32_t rtt = pongTick - pingSentTick;
                *latencyMs = rtt / (TICKS_PER_SECOND / 1000);
                return true;
            }
        }

        // Check timeout
        uint32_t elapsed = TimeTick - startWait;
        if (elapsed / (TICKS_PER_SECOND / 1000) > PONG_TIMEOUT_MS) {
            printf("Pong timeout\r\n");
            return false;
        }

        OSTimeDly(TICKS_PER_SECOND / 100); // 10ms poll interval
    }
}

// Usage
uint32_t pingSent, latency;
WSPing(wsFd, 0, &pingSent);
if (WaitForPong(wsFd, pingSent, &latency)) {
    printf("Connection latency: %lu ms\r\n", latency);
}
```

Example 3: Connection Quality Monitoring

```
struct ConnectionStats {
    uint32_t minLatency;
    uint32_t maxLatency;
    uint32_t avgLatency;
    uint32_t sampleCount;
};

void UpdateConnectionStats(int wsFd, uint32_t pingSent, ConnectionStats *stats) {
    uint32_t pongReceived;

    if (WSGetPingReplyTick(wsFd, &pongReceived) >= 0) {
        if (pongReceived > pingSent) {
            uint32_t latency = (pongReceived - pingSent) / (TICKS_PER_SECOND / 1000);

            // Update statistics
            if (stats->sampleCount == 0 || latency < stats->minLatency) {
                stats->minLatency = latency;
            }
            if (latency > stats->maxLatency) {
                stats->maxLatency = latency;
            }

            // Calculate running average
            stats->avgLatency = ((stats->avgLatency * stats->sampleCount) + latency)
                               / (stats->sampleCount + 1);
            stats->sampleCount++;

            printf("Latency: %lu ms (min: %lu, max: %lu, avg: %lu)\r\n",
                   latency, stats->minLatency, stats->maxLatency, stats->avgLatency);
        }
    }
}
```

Example 4: Timeout Detection

```
#define PING_INTERVAL_SEC 30
#define PONG_TIMEOUT_SEC 5

void ConnectionHealthCheck(int wsFd) {
    static uint32_t lastPingSent = 0;
    static bool waitingForPong = false;
    uint32_t pongReceived;
```

```

if (!waitingForPong) {
    // Time to send a new ping
    if (WSPing(wsFd, 0, &lastPingSent) >= 0) {
        waitingForPong = true;
        printf("Ping sent\r\n");
    }
} else {
    // Check for pong response
    if (WSGetPingReplyTick(wsFd, &pongReceived) >= 0) {
        if (pongReceived > lastPingSent) {
            uint32_t rtt = pongReceived - lastPingSent;
            printf("Pong received, RTT: %lu ms\r\n",
                rtt / (TICKS_PER_SECOND / 1000));
            waitingForPong = false;
        }
    }

    // Check for timeout
    uint32_t elapsed = TimeTick - lastPingSent;
    if (elapsed / TICKS_PER_SECOND > PONG_TIMEOUT_SEC) {
        printf("ERROR: Pong timeout - connection may be dead\r\n");
        close(wsFd);
        return;
    }
}
}
}

```

Example 5: Detailed Latency Analysis

```

void AnalyzeConnectionLatency(int wsFd) {
    const int NUM_PINGS = 10;
    uint32_t latencies[NUM_PINGS];
    int successCount = 0;

    printf("Starting latency analysis (%d pings)...\r\n", NUM_PINGS);

    for (int i = 0; i < NUM_PINGS; i++) {
        uint32_t pingSent, pongReceived;

        // Send ping
        if (WSPing(wsFd, 0, &pingSent) < 0) {
            printf("Ping %d failed\r\n", i + 1);
            continue;
        }

        // Wait for pong (with timeout)
        uint32_t waitStart = TimeTick;
        bool gotPong = false;

        while ((TimeTick - waitStart) < TICKS_PER_SECOND) {
            if (WSGetPingReplyTick(wsFd, &pongReceived) >= 0) {
                if (pongReceived > pingSent) {
                    latencies[successCount++] =
                        (pongReceived - pingSent) / (TICKS_PER_SECOND / 1000);
                    gotPong = true;
                    break;
                }
            }
            OSTimeDly(TICKS_PER_SECOND / 100); // 10ms
        }

        if (!gotPong) {
            printf("Ping %d timeout\r\n", i + 1);
        }

        // Small delay between pings
        OSTimeDly(TICKS_PER_SECOND / 2);
    }

    // Calculate statistics
    if (successCount > 0) {
        uint32_t sum = 0, min = latencies[0], max = latencies[0];
        for (int i = 0; i < successCount; i++) {
            sum += latencies[i];
            if (latencies[i] < min) min = latencies[i];
            if (latencies[i] > max) max = latencies[i];
        }

        printf("\nLatency Analysis Results:\r\n");
        printf("  Successful pings: %d/%d\r\n", successCount, NUM_PINGS);
        printf("  Min: %lu ms\r\n", min);
        printf("  Max: %lu ms\r\n", max);
        printf("  Avg: %lu ms\r\n", sum / successCount);
    }
}

```

See also

[WSPing\(\)](#) - Send a ping frame and record sent timestamp

[WSPong\(\)](#) - Manually send a pong response

[WSRead\(\)](#) - Read WebSocket frames

[TimeTick](#) - Current system tick count

[TICKS_PER_SECOND](#) - Conversion constant for tick-to-time calculations

14.61.2.2 WSPing()

```
int WSPing (  
    int fd,  
    uint32_t len,  
    uint32_t * sentTick)  
#include <websockets.h>
```

Send a WebSocket ping frame to test connection liveness and measure round-trip time.

14.61.3 WSPing

14.61.3.1 Description

Sends a WebSocket ping control frame with optional payload data. The server or client should respond with a pong frame containing the same payload. This function records the tick count when the ping is sent, allowing calculation of round-trip time when the corresponding pong is received.

Ping frames are used to:

- Verify that the connection is still alive
- Measure network latency
- Keep the connection active (prevent timeouts)
- Detect unresponsive peers

14.61.3.2 Parameters

Parameters

<i>fd</i>	The file descriptor of the WebSocket connection
<i>len</i>	Length of the ping payload data in bytes (typically 0-125 bytes)
<i>sentTick</i>	Pointer to store the system tick count when the ping was sent. Can be NULL if timestamp recording is not needed. Use this value with the pong response timestamp to calculate round-trip time.

14.61.3.3 Return Value

Returns the number of bytes sent on success, or a negative value on error:

- Positive value: Number of bytes successfully sent (ping frame)
- Negative value: Error occurred (connection closed, invalid parameters, etc.)

14.61.3.4 Usage Notes

- Ping payloads should be small (WebSocket spec recommends 125 bytes or less)
- The peer must respond with a pong frame containing the same payload
- Store the sentTick value to calculate latency when the pong arrives

- Convert ticks to milliseconds using: $(\text{receivedTick} - \text{sentTick}) / (\text{TICKS\_PER\_SECOND} / 1000)$
- Frequent pings can be used as a keep-alive mechanism
- If no pong is received within a reasonable timeout, the connection may be dead

Expand for Example Usage

Examples

Example 1: Basic Ping without Payload

```
int wsFd = ...; // WebSocket connection
uint32_t pingTime;

// Send ping with no payload
int result = WSPing(wsFd, 0, &pingTime);
if (result >= 0) {
    printf("Ping sent at tick %lu\r\n", pingTime);
} else {
    printf("Failed to send ping\r\n");
}
```

Example 2: Measuring Round-Trip Time

```
uint32_t pingSentTick;
uint32_t pongReceivedTick;

// Send ping
if (WSPing(wsFd, 0, &pingSentTick) >= 0) {
    // Wait for pong response (in your pong handler)
    // ... receive pong ...
    pongReceivedTick = TimeTick;

    // Calculate latency
    uint32_t latencyTicks = pongReceivedTick - pingSentTick;
    uint32_t latencyMs = latencyTicks / (TICKS_PER_SECOND / 1000);
    printf("Round-trip time: %lu ms\r\n", latencyMs);
}
```

Example 3: Periodic Keep-Alive Pings

```
void KeepAliveTask(void *pd) {
    int wsFd = (int)(intptr_t)pd;
    uint32_t lastPingTime;

    while (1) {
        // Send ping every 30 seconds
        OSTimeDly(TICKS_PER_SECOND * 30);

        int result = WSPing(wsFd, 0, &lastPingTime);
        if (result < 0) {
            printf("Keep-alive ping failed, connection may be closed\r\n");
            break;
        }
    }
}
```

Example 4: Ping with Payload for Identification

```
// Send ping with small payload to identify the ping
uint32_t pingId = 12345;
uint32_t sentTime;

// In practice, you'd write the payload first using WSWrite or similar
// This example shows the WSPing call
int result = WSPing(wsFd, sizeof(pingId), &sentTime);
if (result >= 0) {
    printf("Sent ping #%lu at tick %lu\r\n", pingId, sentTime);
}
```

Example 5: Connection Health Monitoring

```

#define PING_TIMEOUT_MS 5000
#define MAX_MISSED_PONGS 3

struct WSConnection {
    int fd;
    uint32_t lastPingSent;
    uint32_t lastPongReceived;
    int missedPongs;
};

void MonitorConnection(WSConnection *conn) {
    // Send ping
    if (WSPing(conn->fd, 0, &conn->lastPingSent) < 0) {
        printf("Connection lost\r\n");
        return;
    }

    // Check if previous pong was received
    uint32_t elapsed = TimeTick - conn->lastPingSent;
    uint32_t elapsedMs = elapsed / (TICKS_PER_SECOND / 1000);

    if (elapsedMs > PING_TIMEOUT_MS) {
        conn->missedPongs++;
        if (conn->missedPongs >= MAX_MISSED_PONGS) {
            printf("Connection appears dead, closing\r\n");
            close(conn->fd);
        }
    }
}

```

14.61.3.5 See Also

- WSPong() - Send a pong response frame
- WSRead() - Read WebSocket frames (including pong responses)
- WSWrite() - Send WebSocket data frames
- WSClose() - Close a WebSocket connection

Note

The WebSocket protocol requires that a pong frame echo back the exact payload received in the ping frame.

14.61.3.6 WSUpgrade()

```

int WSUpgrade (
    HTTP_Request * req,
    int sock)

```

```
#include <websockets.h>
```

Upgrade an HTTP connection to a WebSocket connection.

Performs the WebSocket handshake by upgrading an existing HTTP request to the WebSocket protocol. This function validates the upgrade request, sends the appropriate handshake response, and transitions the socket into WebSocket mode. After a successful upgrade, the socket can be used with WebSocket functions like WSRead() and WSWrite().

This function handles:

- Validation of WebSocket upgrade headers (Upgrade, Connection, Sec-WebSocket-Key)
- Generation of the Sec-WebSocket-Accept response header
- Protocol version negotiation
- Switching the connection from HTTP to WebSocket mode

Parameters

<i>req</i>	Pointer to the HTTP_Request structure containing the upgrade request. The request must contain valid WebSocket upgrade headers including: • Upgrade: websocket • Connection: Upgrade • Sec-WebSocket-Key: <client key> • Sec-WebSocket-Version: 13
<i>sock</i>	The socket file descriptor of the HTTP connection to upgrade. This socket will be transitioned to WebSocket mode on success.

Returns

Returns a status code indicating the result:

- 0 or positive: Success, connection upgraded to WebSocket
- Negative value: Error occurred (invalid request, missing headers, handshake failed)

Note

This function must be called from within an HTTP request handler

After successful upgrade, do not send HTTP responses on this socket

The socket remains open and must be managed using WebSocket functions

Subprotocol negotiation (Sec-WebSocket-Protocol) may be handled automatically

Extensions (Sec-WebSocket-Extensions) may be negotiated based on implementation

After upgrade, use WSRead() and WSWrite() for all communication on this socket

The HTTP request handler should return without sending additional HTTP data

Always check the return value to ensure the upgrade was successful

Expand for Example Usage

Examples

Example 1: Basic WebSocket Upgrade Handler

```
// HTTP handler for WebSocket endpoint
int HandleWebSocketUpgrade(HTTP_Request *req, int sock) {
    // Attempt to upgrade to WebSocket
    if (WSUpgrade(req, sock) >= 0) {
        printf("WebSocket connection established on socket %d\r\n", sock);

        // Connection is now in WebSocket mode
        // Handle WebSocket communication here or in a separate task
        return 0; // Don't send HTTP response
    } else {
        printf("WebSocket upgrade failed\r\n");

        // Send error response
        writestring(sock, "HTTP/1.1 400 Bad Request\r\n");
        writestring(sock, "Content-Type: text/plain\r\n\r\n");
        writestring(sock, "WebSocket upgrade failed");
        return -1;
    }
}

// Register the handler
void UserMain(void *pd) {
    init();
    StartHttp();

    // Register WebSocket endpoint
    SetNewWSHandler("/ws", HandleWebSocketUpgrade);

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

Example 2: WebSocket Upgrade with Echo Server

```
// Simple echo server after upgrade
void WebSocketEchoTask(void *pd) {
    int sock = (int)(intptr_t)pd;
    char buffer[256];

    printf("WebSocket echo task started on socket %d\r\n", sock);

    while (1) {
        // Read WebSocket frame
        int len = WSRead(sock, buffer, sizeof(buffer) - 1, TICKS_PER_SECOND * 30);

        if (len > 0) {
            buffer[len] = '\0';
            printf("Received: %s\r\n", buffer);

            // Echo back
            if (WSWrite(sock, buffer, len) < 0) {
                printf("Failed to echo message\r\n");
                break;
            }
        } else if (len < 0) {
            printf("WebSocket connection closed or error\r\n");
            break;
        }
    }

    close(sock);
    printf("WebSocket echo task ended\r\n");
}

int HandleWebSocketUpgrade(HTTP_Request *req, int sock) {
    if (WSUpgrade(req, sock) >= 0) {
        // Spawn task to handle WebSocket communication
        OSSimpleTaskCreateWithName(WebSocketEchoTask,
                                   MAIN_PRIO - 1,
                                   (void *) (intptr_t) sock,
                                   "WSEcho");
        return 0;
    }

    return -1;
}
```

Example 3: Multiple WebSocket Endpoints

```
// Chat endpoint handler
int HandleChatWebSocket(HTTP_Request *req, int sock) {
    if (WSUpgrade(req, sock) >= 0) {
        printf("Chat WebSocket connected\r\n");
        // Start chat handler task
        OSSimpleTaskCreateWithName(ChatTask, MAIN_PRIO - 1,
                                   (void *) (intptr_t) sock, "WSChat");
        return 0;
    }
    return -1;
}

// Data stream endpoint handler
int HandleDataWebSocket(HTTP_Request *req, int sock) {
    if (WSUpgrade(req, sock) >= 0) {
        printf("Data WebSocket connected\r\n");
        // Start data streaming task
        OSSimpleTaskCreateWithName(DataStreamTask, MAIN_PRIO - 1,
                                   (void *) (intptr_t) sock, "WSData");
        return 0;
    }
    return -1;
}

void UserMain(void *pd) {
    init();
    StartHttp();

    // Register multiple WebSocket endpoints
    SetNewWSHandler("/chat", HandleChatWebSocket);
    SetNewWSHandler("/data", HandleDataWebSocket);

    printf("WebSocket server running with multiple endpoints\r\n");

    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}
```

```
}
```

Example 4: WebSocket Upgrade with Authentication

```
// Check authentication token from query string or headers
bool ValidateWebSocketAuth(HTTP_Request *req) {
    // Example: Check for auth token in query string
    const char *token = GetRequestParam(req, "token");
    if (token == NULL) {
        return false;
    }

    // Validate token (simplified example)
    return (strcmp(token, "secret123") == 0);
}

int HandleSecureWebSocket(HTTP_Request *req, int sock) {
    // Authenticate before upgrading
    if (!ValidateWebSocketAuth(req)) {
        printf("WebSocket authentication failed\r\n");

        writestring(sock, "HTTP/1.1 401 Unauthorized\r\n");
        writestring(sock, "Content-Type: text/plain\r\n\r\n");
        writestring(sock, "Authentication required");
        return -1;
    }

    // Upgrade to WebSocket
    if (WSUpgrade(req, sock) >= 0) {
        printf("Authenticated WebSocket connection established\r\n");
        OSSimpleTaskCreateWithName(SecureWSTask, MAIN_PRIO - 1,
                                   (void *) (intptr_t) sock, "WSSecure");
        return 0;
    }

    return -1;
}
```

Example 5: WebSocket Upgrade with Connection Tracking

```
#define MAX_WS_CONNECTIONS 10

struct WSConnectionInfo {
    int sock;
    bool active;
    uint32_t connectedTime;
    char clientIP[16];
};

WSConnectionInfo wsConnections[MAX_WS_CONNECTIONS];
OS_CRIT wsCrit;

// Add connection to tracking array
bool AddWSConnection(int sock, const char *clientIP) {
    OSCritEnter(&wsCrit, TICKS_PER_SECOND);

    for (int i = 0; i < MAX_WS_CONNECTIONS; i++) {
        if (!wsConnections[i].active) {
            wsConnections[i].sock = sock;
            wsConnections[i].active = true;
            wsConnections[i].connectedTime = Secs;
            strncpy(wsConnections[i].clientIP, clientIP,
                    sizeof(wsConnections[i].clientIP) - 1);
            OSCritLeave(&wsCrit);
            return true;
        }
    }

    OSCritLeave(&wsCrit);
    return false; // No slots available
}

int HandleTrackedWebSocket(HTTP_Request *req, int sock) {
    // Get client IP
    IPADDR clientAddr = req->client_addr;
    char ipStr[16];
    sprintf(ipStr, "%d.%d.%d.%d",
            (int)clientAddr.GetByte(0), (int)clientAddr.GetByte(1),
            (int)clientAddr.GetByte(2), (int)clientAddr.GetByte(3));

    // Check connection limit
    if (!AddWSConnection(sock, ipStr)) {
        printf("Maximum WebSocket connections reached\r\n");
    }
}
```

```

        writestring(sock, "HTTP/1.1 503 Service Unavailable\r\n");
        writestring(sock, "Content-Type: text/plain\r\n\r\n");
        writestring(sock, "Too many connections");
        return -1;
    }

    // Upgrade to WebSocket
    if (WSUpgrade(req, sock) >= 0) {
        printf("WebSocket connected from %s (socket %d)\r\n", ipStr, sock);
        OSSimpleTaskCreateName(TrackedWSTask, MAIN_PRIO - 1,
                               (void *) (intptr_t) sock, "WSTracked");

        return 0;
    } else {
        // Remove from tracking on upgrade failure
        OSCritEnter(&wsCrit, TICKS_PER_SECOND);
        for (int i = 0; i < MAX_WS_CONNECTIONS; i++) {
            if (wsConnections[i].sock == sock) {
                wsConnections[i].active = false;
                break;
            }
        }
        OSCritLeave(&wsCrit);
    }

    return -1;
}

// Display active connections
void ShowWSConnections() {
    OSCritEnter(&wsCrit, TICKS_PER_SECOND);

    printf("\nActive WebSocket Connections:\r\n");
    for (int i = 0; i < MAX_WS_CONNECTIONS; i++) {
        if (wsConnections[i].active) {
            uint32_t uptime = Secs - wsConnections[i].connectedTime;
            printf("  [%d] %s (socket %d) - uptime: %lu sec\r\n",
                   i, wsConnections[i].clientIP,
                   wsConnections[i].sock, uptime);
        }
    }

    OSCritLeave(&wsCrit);
}

```

Example 6: WebSocket Upgrade with Subprotocol Negotiation

```

// Check if client requested a specific subprotocol
const char* NegotiateSubprotocol(HTTP_Request *req) {
    // Example: Client sends "Sec-WebSocket-Protocol: chat, superchat"
    const char *requested = GetRequestHeader(req, "Sec-WebSocket-Protocol");

    if (requested == NULL) {
        return NULL; // No subprotocol requested
    }

    // Check if we support the requested protocol
    if (strstr(requested, "chat") != NULL) {
        return "chat";
    }

    return NULL; // Unsupported subprotocol
}

int HandleSubprotocolWebSocket(HTTP_Request *req, int sock) {
    const char *subprotocol = NegotiateSubprotocol(req);

    if (subprotocol) {
        printf("Negotiated subprotocol: %s\r\n", subprotocol);
        // Note: WSUpgrade may handle this automatically
        // or you may need to add the header before calling WSUpgrade
    }

    if (WSUpgrade(req, sock) >= 0) {
        printf("WebSocket upgraded with subprotocol: %s\r\n",
               subprotocol ? subprotocol : "none");
        return 0;
    }

    return -1;
}

```

See also

[SetNewWSHandler\(\)](#) - Register a handler for WebSocket upgrade requests

[WSRead\(\)](#) - Read data from a WebSocket connection

[WSWrite\(\)](#) - Write data to a WebSocket connection

[WSClose\(\)](#) - Close a WebSocket connection gracefully

[WSPing\(\)](#) - Send a ping frame to test connection

[HTTP_Request](#) - Structure containing HTTP request information

14.61.3.7 WSWaitForPingReply()

```
int WSWaitForPingReply (
    int fd,
    uint32_t timeout)
#include <websockets.h>
```

Wait for a pong reply to a previously sent ping with a specified timeout.

Blocks execution until a pong frame is received in response to a ping, or until the specified timeout expires. This is a convenience function that combines waiting and timeout detection, simplifying the common pattern of sending a ping and waiting for the corresponding pong response.

This function is useful for:

- Synchronous ping/pong operations with automatic timeout handling
- Verifying connection responsiveness in a blocking manner
- Simplifying connection health checks without manual polling
- Testing connection latency with built-in timeout protection

Parameters

<i>fd</i>	The file descriptor of the WebSocket connection
<i>timeout</i>	Maximum time to wait for a pong reply, in system ticks. Use TICKS_PER_SECOND to specify timeout in seconds (e.g., TICKS_PER_SECOND * 5 for a 5-second timeout). A timeout of 0 returns immediately.

Returns

Returns a status code indicating the result:

- 0 or positive: Success, pong received within timeout period
- Negative value: Error occurred (timeout expired, connection closed, invalid parameters)

Note

This function blocks the calling task until a pong is received or timeout expires

A ping should be sent using [WSPing\(\)](#) before calling this function

The timeout parameter is in system ticks, not milliseconds

Use TICKS_PER_SECOND to convert seconds to ticks: `timeout = TICKS_PER_SECOND * seconds`

This function does not return the latency; use [WSGetPingReplyTick\(\)](#) if you need timing info

If multiple pongs arrive, this returns on the first pong received during the wait period

For non-blocking operations, use [WSGetPingReplyTick\(\)](#) with manual polling instead

Expand for Example Usage**Examples**

Example 1: Basic Ping/Pong with Timeout

```

int wsFd = ...; // WebSocket connection
uint32_t pingSent;

// Send ping
if (WSPing(wsFd, 0, &pingSent) >= 0) {
    printf("Ping sent, waiting for reply...\r\n");

    // Wait up to 3 seconds for pong
    int result = WSWaitForPingReply(wsFd, TICKS_PER_SECOND * 3);
    if (result >= 0) {
        printf("Pong received!\r\n");
    } else {
        printf("Timeout or error waiting for pong\r\n");
    }
}

```

Example 2: Connection Health Check with Blocking Wait

```

bool CheckConnectionHealth(int wsFd) {
    uint32_t pingSent;

    // Send ping and wait for reply
    if (WSPing(wsFd, 0, &pingSent) < 0) {
        printf("Failed to send ping\r\n");
        return false;
    }

    // Wait up to 5 seconds for pong
    if (WSWaitForPingReply(wsFd, TICKS_PER_SECOND * 5) >= 0) {
        printf("Connection is healthy\r\n");
        return true;
    } else {
        printf("Connection appears unhealthy or slow\r\n");
        return false;
    }
}

```

Example 3: Periodic Keep-Alive with Timeout Detection

```

void KeepAliveTask(void *pd) {
    int wsFd = (int)(intptr_t)pd;
    uint32_t pingSent;
    int consecutiveFailures = 0;
    const int MAX_FAILURES = 3;

    while (1) {
        // Wait 30 seconds between pings
        OSTimeDly(TICKS_PER_SECOND * 30);

        // Send ping
        if (WSPing(wsFd, 0, &pingSent) < 0) {
            printf("Failed to send ping\r\n");
            consecutiveFailures++;
        } else {
            // Wait for pong with 10 second timeout
            if (WSWaitForPingReply(wsFd, TICKS_PER_SECOND * 10) >= 0) {
                printf("Keep-alive successful\r\n");
                consecutiveFailures = 0;
            } else {
                printf("Keep-alive timeout\r\n");
                consecutiveFailures++;
            }
        }

        // Check if connection is dead
        if (consecutiveFailures >= MAX_FAILURES) {
            printf("Connection lost after %d failures\r\n", MAX_FAILURES);
            close(wsFd);
            break;
        }
    }
}

```

Example 4: Connection Latency Measurement

```

bool MeasureLatency(int wsFd, uint32_t *latencyMs) {
    uint32_t pingSent, pongReceived;

```



```

    // Send ping
    if (WSPing(wsFd, 0, &pingSent) < 0) {
        return false;
    }

    // Wait for pong with 2 second timeout
    if (WSWaitForPingReply(wsFd, TICKS_PER_SECOND * 2) < 0) {
        printf("Ping timeout\r\n");
        return false;
    }

    // Get the pong timestamp to calculate latency
    if (WSGetPingReplyTick(wsFd, &pongReceived) >= 0) {
        uint32_t rtt = pongReceived - pingSent;
        *latencyMs = rtt / (TICKS_PER_SECOND / 1000);
        return true;
    }

    return false;
}

// Usage
uint32_t latency;
if (MeasureLatency(wsFd, &latency)) {
    printf("Connection latency: %lu ms\r\n", latency);
}

```

Example 5: Progressive Timeout Strategy

```

typedef enum {
    CONN_EXCELLENT, // < 100ms
    CONN_GOOD,      // < 500ms
    CONN_FAIR,      // < 2000ms
    CONN_POOR       // >= 2000ms or timeout
} ConnectionQuality;

ConnectionQuality AssessConnectionQuality(int wsFd) {
    uint32_t pingSent, pongReceived;

    // Send ping
    if (WSPing(wsFd, 0, &pingSent) < 0) {
        return CONN_POOR;
    }

    // Try quick response first (100ms)
    if (WSWaitForPingReply(wsFd, TICKS_PER_SECOND / 10) >= 0) {
        return CONN_EXCELLENT;
    }

    // Try good response (additional 400ms)
    if (WSWaitForPingReply(wsFd, (TICKS_PER_SECOND * 4) / 10) >= 0) {
        return CONN_GOOD;
    }

    // Try fair response (additional 1500ms)
    if (WSWaitForPingReply(wsFd, (TICKS_PER_SECOND * 15) / 10) >= 0) {
        return CONN_FAIR;
    }

    return CONN_POOR;
}

// Usage
ConnectionQuality quality = AssessConnectionQuality(wsFd);
const char* qualityStr[] = {"Excellent", "Good", "Fair", "Poor"};
printf("Connection quality: %s\r\n", qualityStr[quality]);

```

Example 6: Batch Connection Testing

```

struct WSConnection {
    int fd;
    char name[32];
    bool isAlive;
};

void TestMultipleConnections(WSConnection *connections, int count) {
    printf("Testing %d WebSocket connections...\r\n", count);

    for (int i = 0; i < count; i++) {
        uint32_t pingSent;

        printf("Testing %s... ", connections[i].name);
    }
}

```

```

// Send ping
if (WSPing(connections[i].fd, 0, &pingSent) < 0) {
    printf("FAILED (ping)\r\n");
    connections[i].isAlive = false;
    continue;
}

// Wait for pong with 3 second timeout
if (WSWaitForPingReply(connections[i].fd, TICKS_PER_SECOND * 3) >= 0) {
    printf("OK\r\n");
    connections[i].isAlive = true;
} else {
    printf("FAILED (timeout)\r\n");
    connections[i].isAlive = false;
}
}

// Summary
int aliveCount = 0;
for (int i = 0; i < count; i++) {
    if (connections[i].isAlive) aliveCount++;
}
printf("\nResults: %d/%d connections alive\r\n", aliveCount, count);
}

```

See also

- [WSPing\(\)](#) - Send a ping frame and record sent timestamp
- [WSGetPingReplyTick\(\)](#) - Get timestamp of last pong (for latency calculation)
- [WSPong\(\)](#) - Manually send a pong response
- [WSRead\(\)](#) - Read WebSocket frames
- [TICKS_PER_SECOND](#) - Conversion constant for timeout specification

14.61.4 Web Socket Definitions

WebSocket Protocol Implementation for Real-Time Bidirectional Communication `#include< websockets.h>`

Macros

- **#define WS_BUFFER_SEGMENTS 4**
Number of buffer segments per WebSocket connection (RX, TX, Control buffers)
- **#define WS_MAX_SOCKETS 4**
Maximum number of simultaneous WebSocket connections supported.
- **#define WS_BUFFER_MIN 10**
Minimum buffer space threshold (bytes) before triggering flush/block.
- **#define WS_ACCUM_DLY ((TICKS_PER_SECOND / 4) + 1)**
Accumulation delay (ticks) for batching small writes (~250ms)
- **#define WS_FLUSH_TIMEOUT 0**
Flush operation timeout in ticks (0 = non-blocking)
- **#define WS_FIN_BIT 0x80**
Frame FIN bit (0x80) indicating final fragment of message.
- **#define WS_OP_CONT 0x0**
WebSocket continuation frame opcode for message fragments.
- **#define WS_OP_TEXT 0x1**
WebSocket text frame opcode for UTF-8 encoded text data.
- **#define WS_OP_BIN 0x2**
WebSocket binary frame opcode for raw binary data.
- **#define WS_OP_CLOSE 0x8**
WebSocket close frame opcode for graceful connection termination.
- **#define WS_OP_PING 0x9**
WebSocket ping frame opcode for connection liveness check.

- `#define WS_OP_PONG 0xA`
WebSocket pong frame opcode for ping response.
- `#define WS_SO_TEXT 0x8`
Socket option flag to enable text frame mode (vs binary mode)

14.61.4.1 Detailed Description

WebSocket Protocol Implementation for Real-Time Bidirectional Communication `#include <websockets.h>`

14.61.4.2 Overview

The NetBurner WebSocket implementation provides full-duplex communication channels over a single TCP connection, enabling real-time data exchange between clients and servers. WebSockets upgrade from HTTP connections and maintain persistent connections for efficient, low-latency bidirectional messaging.

Key features include:

- RFC 6455 compliant WebSocket protocol implementation
- Automatic connection upgrade from HTTP to WebSocket
- Support for text and binary data frames
- Built-in ping/pong for connection health monitoring
- Efficient buffering and flow control
- Support for both client and server roles
- Graceful connection closure with status codes
- Frame fragmentation for large messages

Note

See detailed information in the [Programmers Guide Web Sockets](#)

Defines for configuration, status and error codes.

Note

See detailed information in the [Programmers Guide Web Sockets Definitions](#)

14.62 Wifi

NetBurner WiFi API.

Topics

- [BSS Options](#)
Option list BSS type values.
- [Cipher Options](#)
- [Configuration Errors](#)
- [Connect Request Errors](#)
- [Save Config Record Errors](#)
- [Scan Request Errors](#)
- [Security Options](#)

Classes

- struct [wifi_init](#)

Functions

- `nbWifiScanResult * WifilnitScan_SPI` (int irqNum=-1, int moduleNum=-1, int csNum=-1, int connectorNum=-1, int gpioPinNum=-1, int resetPinNum=-1)
Initializes the WiFi hardware, initializes the driver over the SPI bus, and performs an AP scan.
- `int WifilnitScanAndShow_SPI` (int irqNum=-1, int moduleNum=-1, int csNum=-1, int connectorNum=-1, int gpioPinNum=-1, int resetPinNum=-1)
Initializes the WiFi hardware, initializes the driver using the SPI bus, performs an AP scan, and prints the scan results via serial output.
- `int InitWifi_SPI` (const char *SSID="", const char *password="", int irqNum=-1, int moduleNum=-1, int csNum=-1, int connectorNum=-1, int gpioPinNum=-1, int resetPinNum=-1)
Initializes the WiFi hardware, initializes the driver using the SPI bus, and attempts to establish the specified connection.
- `int InitAP_SPI` (const char *SSID="", const char *password="", uint8_t channel=NBWIFI_DEFAULT_WIFICHANNEL, int irqNum=-1, int moduleNum=-1, int csNum=-1, int connectorNum=-1, int gpioPinNum=-1, int resetPinNum=-1)
Initializes the WiFi hardware, initializes the driver using the SPI bus, and attempts to establish the specified access point.
- `nbWifiScanResult * WifilnitScan_Serial` (int portNum=-1, int resetPinNum=-1, int connectorNum=-1)
Initializes the WiFi hardware, initializes the driver using the UART interface, performs an AP scan, and prints the scan results via serial output.
- `int WifilnitScanAndShow_Serial` (int portNum=-1, int resetPinNum=-1, int connectorNum=-1)
Initializes the WiFi hardware, initializes the driver using the UART interface, performs an AP scan, and prints the scan results via serial output.
- `int InitWifi_Serial` (const char *SSID="", const char *password="", int portNum=-1, int resetPinNum=-1, int connectorNum=-1)
Initializes the WiFi hardware, initializes the driver using the UART interface, and attempts to establish the specified access point.
- `void SetWifiSPISpeed` (int busSpeed)
Set SPI bus speed.
- `void ScanAndShowNetworks` ()
Scan for surrounding access points and print the results via iprintf.
- `nbWifiScanResult * ScanForNetworks` ()
Scan for surrounding access points.

14.62.1 Detailed Description

NetBurner WiFi API.

```
#include< nbWifiConstants.h>
```

Initialize and use a NetBurner WiFi module via SPI or Serial in either Client or AP mode.

```
#include< wifi.h>
```

See also

[WiFi Examples](#)

14.62.2 Function Documentation

14.62.2.1 InitAP_SPI()

```
int InitAP_SPI (
    const char * SSID = "",
    const char * password = "",
    uint8_t channel = NBWIFI_DEFAULT_WIFICHANNEL,
    int irqNum = -1,
    int moduleNum = -1,
```

```

int csNum = -1,
int connectorNum = -1,
int gpioPinNum = -1,
int resetPinNum = -1)

```

```
#include <wifi.h>
```

Initializes the WiFi hardware, initializes the driver using the SPI bus, and attempts to establish the specified access point.

Parameters

in	<i>SSID</i>	Service Set Identity (SSID), nullptr uses SSID stored in the configuration record.
in	<i>password</i>	key shared with access point/peer; nullptr uses the key stored in the configuration record.
in	<i>channel</i>	802.11 channel to establish an access point on.
in	<i>irqNum</i>	IRQ signal used to communicate with the WiFi module.
in	<i>moduleNum</i>	SPI module number used to communicate with the WiFi module.
in	<i>csNum</i>	Chip Select signal used for SPI communication.
in	<i>connectorNum</i>	physical header number of the NetBurner module used.
in	<i>gpioPinNum</i>	GPIO pin number to use for the SPI chip select if not using the SPI module's native chip select signal.
in	<i>resetPinNum</i>	pin number to use as an reset signal output to the wifi module.

Returns

returns the Wifi interface number, if successful. Otherwise, returns [NB::Error::GeneralErrors](#) or [NB::Error::Init::InitializationErrors](#)

14.62.2.2 InitWifi_Serial()

```

int InitWifi_Serial (
    const char * SSID = "",
    const char * password = "",
    int portNum = -1,
    int resetPinNum = -1,
    int connectorNum = -1)

```

```
#include <wifi.h>
```

Initializes the WiFi hardware, initializes the driver using the UART interface, and attempts to establish the specified access point.

NOTE: Serial functions pertaining to WiFi are only available for the NBWIFIIN.

Parameters

in	<i>SSID</i>	Service Set Identity (SSID), nullptr uses SSID stored in the configuration record.
in	<i>password</i>	key shared with access point/peer; nullptr uses the key stored in the configuration record.
in	<i>portNum</i>	UART number
in	<i>resetPinNum</i>	pin number to use as an reset signal output to the wifi module.
in	<i>connectorNum</i>	physical header number of the NetBurner module used.

Returns

returns zero if successful, otherwise returns [NB::Error::GeneralErrors](#) or [NB::Error::Init::InitializationErrors](#)

14.62.2.3 InitWifi_SPI()

```
int InitWifi_SPI (
    const char * SSID = "",
    const char * password = "",
    int irqNum = -1,
    int moduleNum = -1,
    int csNum = -1,
    int connectorNum = -1,
    int gpioPinNum = -1,
    int resetPinNum = -1)
```

```
#include <wifi.h>
```

Initializes the WiFi hardware, initializes the driver using the SPI bus, and attempts to establish the specified connection.

Parameters

in	<i>SSID</i>	Service Set Identity (SSID), nullptr uses SSID stored in the configuration record.
in	<i>password</i>	key shared with access point/peer; nullptr uses the key stored in the configuration record.
in	<i>irqNum</i>	IRQ signal used to communicate with the WiFi module.
in	<i>moduleNum</i>	SPI module number used to communicate with the WiFi module.
in	<i>csNum</i>	Chip Select signal used for SPI communication.
in	<i>connectorNum</i>	physical header number of the NetBurner module used.
in	<i>gpioPinNum</i>	GPIO pin number to use for the SPI chip select, if not using the SPI's native chip select signal.
in	<i>resetPinNum</i>	pin number to use as an reset signal output to the wifi module.

Returns

returns the Wifi interface number, if successful. Otherwise, returns [NB::Error::GeneralErrors](#) or [NB::Error::Init::InitializationErrors](#)

14.62.2.4 ScanAndShowNetworks()

```
void ScanAndShowNetworks ()
#include <wifi.h>
```

Scan for surrounding access points and print the results via iprintf.
This function is bus interface agnostic.

14.62.2.5 ScanForNetworks()

```
nbWifiScanResult * ScanForNetworks ()
#include <wifi.h>
```

Scan for surrounding access points.

Returns

nbWifiScanResult Head of linked list containing the AP scan results. nullptr if scan failed or no access points were found

This function is bus interface agnostic.

14.62.2.6 SetWifiSPISpeed()

```
void SetWifiSPISpeed (
    int busSpeed)
#include <wifi.h>
```

Set SPI bus speed.

Parameters

in	<i>busSpeed</i>	Bus speed in Hz.

14.62.2.7 WifilnitScan_Serial()

```
nbWifiScanResult * WifiInitScan_Serial (
    int portNum = -1,
    int resetPinNum = -1,
    int connectorNum = -1)
```

```
#include <wifi.h>
```

Initializes the WiFi hardware, initializes the driver using the UART interface, performs an AP scan, and prints the scan results via serial output.

NOTE: Serial functions pertaining to WiFi are only available for the NBWIFIIN.

Parameters

in	<i>portNum</i>	UART number
in	<i>resetPinNum</i>	pin number to use as an reset signal output to the wifi module.
in	<i>connectorNum</i>	physical header number of the NetBurner module used.

Returns

nbWifiScanResult Head of linked list containing the AP scan results. nullptr if scan failed or no access points were found

14.62.2.8 WifilnitScan_SPI()

```
nbWifiScanResult * WifiInitScan_SPI (
    int irqNum = -1,
    int moduleNum = -1,
    int csNum = -1,
    int connectorNum = -1,
    int gpioPinNum = -1,
    int resetPinNum = -1)
```

```
#include <wifi.h>
```

Initializes the WiFi hardware, initializes the driver over the SPI bus, and performs an AP scan.

Parameters

in	<i>irqNum</i>	IRQ signal used to communicate with the WiFi module.
in	<i>moduleNum</i>	SPI module number used to communicate with the WiFi module.
in	<i>csNum</i>	Chip Select signal used for SPI communication.
in	<i>connectorNum</i>	physical header number of the NetBurner module used.
in	<i>gpioPinNum</i>	GPIO pin number to use for the SPI chip select if not using the SPI's native chip select signal.
in	<i>resetPinNum</i>	pin number to use as an reset signal output to the wifi module.

Returns

nbWifiScanResult Head of linked list containing the AP scan results. nullptr if scan failed or no access points were found

Initializes the WiFi Driver to communicate with the WiFi module over the SPI bus and performs a scan for surrounding access points (AP). The result of the scan is returned as the head of a linked list that contains the scan results. If the WiFi driver has already been initialized by a previous call to one of the Wifilnit function variations, then only a scan will be performed.

14.62.2.9 WifilnitScanAndShow_Serial()

```
int WifiInitScanAndShow_Serial (
    int portNum = -1,
    int resetPinNum = -1,
    int connectorNum = -1)
```

```
#include <wifi.h>
```

Initializes the WiFi hardware, initializes the driver using the UART interface, performs an AP scan, and prints the scan results via serial output.

NOTE: Serial functions pertaining to WiFi are only available for the NBWIFIIN.

Parameters

in	<i>portNum</i>	UART number
in	<i>resetPinNum</i>	pin number to use as an reset signal output to the wifi module.
in	<i>connectorNum</i>	physical header number of the NetBurner module used.

Returns

the Wifi interface number if successful. Otherwise, returns [NB::Error::GeneralErrors](#) or [NB::Error::Init::InitializationErrors](#)

14.62.2.10 WifilnitScanAndShow_SPI()

```
int WifiInitScanAndShow_SPI (
    int irqNum = -1,
    int moduleNum = -1,
    int csNum = -1,
    int connectorNum = -1,
    int gpioPinNum = -1,
    int resetPinNum = -1)
```

```
#include <wifi.h>
```

Initializes the WiFi hardware, initializes the driver using the SPI bus, performs an AP scan, and prints the scan results via serial output.

Parameters

in	<i>irqNum</i>	IRQ signal used to communicate with the WiFi module.
in	<i>moduleNum</i>	SPI module number used to communicate with the WiFi module.
in	<i>csNum</i>	Chip Select signal used for SPI communication.
in	<i>connectorNum</i>	physical header number of the NetBurner module used.
in	<i>gpioPinNum</i>	GPIO pin number to use for the SPI chip select if not using the SPI's native chip select signal.
in	<i>resetPinNum</i>	pin number to use as an reset signal output to the wifi module.

Returns

returns the Wifi interface number, if successful. Otherwise, returns [NB::Error::GeneralErrors](#) or [NB::Error::Init::InitializationErrors](#)

14.62.3 BSS Options

Option list BSS type values.

Macros

- #define **BSSTYPE_VALUE_INFR** (0x00)

Infrastructure.

- #define **BSSTYPE_VALUE_ADHOC** (0x01)

ADHOC.

- #define **BSSTYPE_VALUE_ANY** (0x02)

Any.

- #define **BSSTYPE_VALUE_UNKNOWN** (0xFF)

Unknown.

14.62.3.1 Detailed Description

Option list BSS type values.

14.62.4 Cipher Options

Macros

- #define **CIPH_VALUE_NONE** (0x00)

Option list cipher value: No cipher.

- #define **CIPH_VALUE_TKIP** (0x01)

Option list cipher value: TKIP.

- #define **CIPH_VALUE_AES** (0x02)

Option list cipher value: AES.

- #define **CIPH_VALUE_MIXED** (0x03)

Option list cipher value: Mixed TKIP/AES.

- #define **CIPH_VALUE_UNKNOWN** (0xFF)

Option list cipher value: Unknown cipher.

14.62.4.1 Detailed Description

14.62.5 Configuration Errors

Macros

- #define **CONFIG_ERR_SUCCESS** 0

Success.

- #define **CONFIG_ERR_MSG_LENGTH** -1

Message length.

- #define **CONFIG_ERR_SSID_LEN_SHORT** -2

SSID length too short.

- #define **CONFIG_ERR_SSID_LEN_LONG** -3

SSID length too long.

- #define **CONFIG_ERR_PASSWD_LEN_LONG** -4

Password length too long.

- #define **CONFIG_ERR_INVALID_TABLE** -5

Invalid table.

- #define **CONFIG_ERR_INVALID_OPTION** -6

Invalid option.

- #define **CONFIG_ERR_BSSID_OVERRUN** -7

BSSID overrun.

- #define **CONFIG_ERR_BSSID_UNDERRUN** -8

BSSID underrun.

- #define **CONFIG_ERR_MULTI_CHANNEL** -9

Multi-channel.

- #define **CONFIG_ERR_CONNECTED** -10

Already connected.

- #define **CONFIG_ERR_UNKNOWN** 1

Unknown.

14.62.5.1 Detailed Description

14.62.6 Connect Request Errors

Macros

- #define **CONNECT_ERR_SUCCESS** 0
Successfull connect request.
- #define **CONNECT_ERR_NOT_CONFIG** -1
Paramaters not configured.
- #define **CONNECT_ERR_INVALID_CONFIG_NUM** -2
Invalid config number.
- #define **CONNECT_ERR_CONNECTED** -3
Already connected.
- #define **CONNECT_ERR_SSID_NOT_FOUND** -4
SSID not found.
- #define **CONNECT_ERR_BSSID_NOT_FOUND** -5
BSSID not found.
- #define **CONNECT_ERR_SEC_MISMATCH** -6
Security mismatch.
- #define **CONNECT_ERR_CIPH_MISMATCH** -7
Cipher mismatch.
- #define **CONNECT_ERR_INVALID_KEY** -8
Invalid password.
- #define **CONNECT_ERR_UNKNOWN** 1
Unknown error.

14.62.6.1 Detailed Description

14.62.7 Save Config Record Errors

Macros

- #define **SAVE_CONF_ERR_SUCCESS** 0
Success.
- #define **SAVE_CONF_ERR_INVALID_CONFIG_NUM** -1
Invalid configuration number.
- #define **SAVE_CONF_ERR_NOT_CONFIGURED** -2
Not configured.
- #define **SAVE_CONF_ERR_UNKNOWN** 1
Unknown error.

14.62.7.1 Detailed Description

14.62.8 Scan Request Errors

Macros

- #define **SCAN_ERR_SUCCESS** 0
Success.
- #define **SCAN_ERR_MSG_LENGTH** -1
Invalid message length.

- #define **SCAN_ERR_IN_PROGRESS** -2
Scan in progress.
- #define **SCAN_ERR_SSID_LEN_LONG** -3
SSID length too long.
- #define **SCAN_ERR_INVALID_TABLE** -4
Invalid table.
- #define **SCAN_ERR_INVALID_OPTION** -5
Invalid option.
- #define **SCAN_ERR_TOO_MANY_CHANNELS** -6
Too many channels.
- #define **SCAN_ERR_UNKNOWN** 1
Unknown error.

14.62.8.1 Detailed Description

14.62.9 Security Options

Macros

- #define **SEC_VALUE_OPEN** (0x00)
Option list security value: Open network.
- #define **SEC_VALUE_WEP** (0x01)
Option list security value: WEP.
- #define **SEC_VALUE_WPA** (0x02)
Option list security value: WPA.
- #define **SEC_VALUE_WPA2** (0x03)
Option list security value: WPA2.
- #define **SEC_VALUE_WPS** (0x04)
Option list security value: WPS.
- #define **SEC_VALUE_ANY** (0xFE)
Option list security value: Any, used to connect as a client regardless of how security is configured for AP.
- #define **SEC_VALUE_UNKNOWN** (0xFF)
Option list security value: Unknown security.

14.62.9.1 Detailed Description

Chapter 15

Namespace Documentation

15.1 canMCF5441x Namespace Reference

[canMCF5441x](#) namespace

Classes

- class [CanRxMessage](#)
Class to hold received CAN messages.

15.1.1 Detailed Description

[canMCF5441x](#) namespace

15.2 DHCP Namespace Reference

[DHCP](#) Namespace.

Classes

- class [BlockAllocator](#)
Basic allocator that handles multiple leases in a contiguous IP block.
- struct [DhcpInfo](#)
DHCP Info.
- struct [DhcpLeaseData](#)
Lease Data.
- struct [DhcpLeaseRequest](#)
Lease Request.
- class [LeaseAllocator](#)
Base class/interface for lease allocators for the [DHCP](#) server.
- class [MacPrefixAllocator](#)
Derived from [BlockAllocator](#) Has the ability to whitelist/blacklist certain mac address ranges based on a mask. Useful for allocating addresses based on device manufacturer.
- class [Server](#)
DHCP Server class Requires a lease allocator to be added in order to function.
- class [SingleAllocator](#)
Stripped down allocator Fails to conform to many standard behaviors, but shows the basics of what needs to be implemented.

Typedefs

- typedef enum [DHCP::LeaseState](#) **LeaseState_t**
Lease State.

Enumerations

- enum [LeaseState](#) {
[LEASE_OPEN](#) = 0x0 , [LEASE_OFFERED](#) = 0x1 , [LEASE_TAKEN](#) = 0x2 , [LEASE_STATIC](#) = 0x3 ,
[ARP_CONFLICT](#) = 0x4 }
Lease State.

15.2.1 Detailed Description

[DHCP](#) Namespace.

15.2.2 Enumeration Type Documentation

15.2.2.1 LeaseState

enum [DHCP::LeaseState](#)
Lease State.

Enumerator

LEASE_OPEN	
LEASE_OFFERED	
LEASE_TAKEN	
LEASE_STATIC	
ARP_CONFLICT	

15.3 mcanMODM7AE70 Namespace Reference

[mcanMODM7AE70](#) namespace

Classes

- class [CanRxMessage](#)
Class to hold received CAN messages.
- class [mcan_config](#)
MCAN configuration structure.
- class [mcan_module](#)
MCAN Module Class.

Variables

MCAN Constants

- const uint32_t **CONF_MCAN_RX_FIFO_0_NUM** = 32
Range: 1..64.
- const uint32_t **CONF_MCAN_RX_FIFO_1_NUM** = 1
Range: 1..64.
- const uint32_t **CONF_MCAN_RX_BUFFER_NUM** = 1
Range: 1..64.
- const uint32_t **CONF_MCAN_TX_BUFFER_NUM** = 8
Range: 1..16.

- const uint32_t **CONF_MCAN_TX_FIFO_QUEUE_NUM** = 1
Range: 1..16.
- const uint32_t **CONF_MCAN_TX_EVENT_FIFO** = 8
Range: 1..32.
- const uint32_t **CONF_MCAN_RX_STANDARD_ID_FILTER_NUM** = 32
Range: 1..128.
- const uint32_t **CONF_MCAN_RX_EXTENDED_ID_FILTER_NUM** = 32
Range: 1..64.

15.3.1 Detailed Description

[mcanMODM7AE70](#) namespace

15.4 MQTT Namespace Reference

[MQTT](#) Namespace.

Namespaces

- namespace [PacketInfo](#)
[MQTT PacketInfo](#) Namespace.

Classes

- class [Client](#)
Advanced low-level [MQTT Client](#) for interacting with Mqtt Brokers. The client handles all network communications with the Broker and handles local message queueing and dispatching to topic handlers.
- class [ConfigExposer](#)
Easy MqttObj class for automatically publishing and subscribing NetBurner Config system objects to/from an [MQTT](#) topic. Use like a normal ConfigObj.
- class [ConfiguredClient](#)
Easy [MQTT Client](#) wrapper for interacting with [MQTT](#) brokers. The client handles all network communications with the Broker and handles local message queueing and dispatching to topic handlers. Adds ConfigObj for automatically configuring all [Client](#) settings via the Config System.
- struct [ConnectRequest](#)
Structure used to specify details for a Publish call.
- class [mqtt_bool](#)
Easy MqttObj class for automatically publishing and subscribing boolean values to/from an [MQTT](#) topic. Use like a normal bool.
- class [mqtt_float](#)
Easy MqttObj class for automatically publishing and subscribing floating point values to/from an [MQTT](#) topic. Use like a normal double.
- class [mqtt_int](#)
Easy MqttObj class for automatically publishing and subscribing unsigned integer values to/from an [MQTT](#) topic. Use like a normal int32_t.
- class [mqtt_IO](#)
Easy MqttObj class for automatically publishing and subscribing hardware GPIO pins to/from an [MQTT](#) topic. Use like a normal [PinIO](#).
- class [mqtt_IO_Input](#)
MqttObj connecting hardware GPIO pins to an Mqtt Broker. This class defaults the pin to an Input state.
- class [mqtt_IO_Output](#)
MqttObj connecting hardware GPIO pins to an Mqtt Broker. This class defaults the pin to an Output state.
- class [mqtt_leaf](#)
Base class for all Mqtt Objects (MqttObj). Mqtt Objects provide a seamless bridge/synchronization to communicate information through an [MQTT](#) Broker via a [Client](#) object.

- class [mqtt_obj](#)
Base class for easy [MQTT](#) Objects (MqttObj). [mqtt_obj](#) are comprised of higher order parent objects that contain registered child objects.
- class [mqtt_string](#)
Easy MqttObj class for automatically publishing and subscribing string values to/from an [MQTT](#) topic. Use like a normal [NBString](#).
- class [mqtt_uint](#)
Easy MqttObj class for automatically publishing and subscribing unsigned integer values to/from an [MQTT](#) topic. Use like a normal [uint32_t](#).
- struct [PublishRequest](#)
Structure used to specify details for a Publish call.
- class [SubHandler](#)
Easy class for registering a callback function that receives messages on an [MQTT](#) topic and handles them appropriately. Behaves similarly to [HttpResponseHandler](#).
- struct [SubscribeFilter](#)
Structure used to specify a topic filter for subscriptions.
- struct [SubscribeFilter_NbStr](#)
Structure used to specify a topic filter for subscriptions.
- struct [SubscribeRequest](#)
Structure used to specify details for a Subscribe call.
- struct [TopicHandler](#)
[TopicHandler](#) context definition.

Typedefs

- typedef int(* [MsgCallback_t](#)) ([MQTT::Client](#) *mConn, int mqttFd, [MQTT::PacketInfo::Msg](#) *msg, void *ctx)
Function signature for a user-implemented callback function to receive messages from a subscription.

Enumerations

- enum [eResult_t](#) {
[eResult_Success](#) = 0 , [eResult_Mqtt_Unspecified](#) = -Pkt::eReason_Error_Unspecified , [eResult_Mqtt_MalformedPkt](#) = -Pkt::eReason_Error_MalformedPkt , [eResult_Mqtt_Protocol](#) = -Pkt::eReason_Error_Protocol ,
[eResult_Mqtt_Implementation](#) = -Pkt::eReason_Error_Implementation , [eResult_Mqtt_UnsupportedProtoVer](#) = -Pkt::eReason_Error_UnsupportedProtoVer , [eResult_Mqtt_ClientIdInvalid](#) = -Pkt::eReason_Error_ClientIdInvalid , [eResult_Mqtt_UserPassword](#) = -Pkt::eReason_Error_UserPassword ,
[eResult_Mqtt_NotAuthorized](#) = -Pkt::eReason_Error_NotAuthorized , [eResult_Mqtt_ServerUnavail](#) = -Pkt::eReason_Error_ServerUnavail , [eResult_Mqtt_ServerBusy](#) = -Pkt::eReason_Error_ServerBusy ,
[eResult_Mqtt_Banned](#) = -Pkt::eReason_Error_Banned ,
[eResult_Mqtt_ServerShutdown](#) = -Pkt::eReason_Error_ServerShutdown , [eResult_Mqtt_BadAuthMethod](#) = -Pkt::eReason_Error_BadAuthMethod , [eResult_Mqtt_KATimeout](#) = -Pkt::eReason_Error_KATimeout ,
[eResult_Mqtt_SessTakeover](#) = -Pkt::eReason_Error_SessTakeover ,
[eResult_Mqtt_FilterInvalid](#) = -Pkt::eReason_Error_FilterInvalid , [eResult_Mqtt_NameInvalid](#) = -Pkt::eReason_Error_NameInvalid , [eResult_Mqtt_PkId_InUse](#) = -Pkt::eReason_Error_PkId_InUse ,
[eResult_Mqtt_PkId_NotFound](#) = -Pkt::eReason_Error_PkId_NotFound ,
[eResult_Mqtt_RcvMax_Exceeded](#) = -Pkt::eReason_Error_RcvMax_Exceeded , [eResult_Mqtt_TopicAlias_Invalid](#) = -Pkt::eReason_Error_TopicAlias_Invalid , [eResult_Mqtt_TooLarge](#) = -Pkt::eReason_Error_TooLarge ,
[eResult_Mqtt_MsgRate](#) = -Pkt::eReason_Error_MsgRate ,
[eResult_Mqtt_Quota](#) = -Pkt::eReason_Error_Quota , [eResult_Mqtt_Admin](#) = -Pkt::eReason_Error_Admin ,
[eResult_Mqtt_PayFmt_Invalid](#) = -Pkt::eReason_Error_PayFmt_Invalid , [eResult_Mqtt_NotSupported_Retain](#) = -Pkt::eReason_Error_NotSupported_Retain ,
[eResult_Mqtt_NotSupported_QoS](#) = -Pkt::eReason_Error_NotSupported_QoS , [eResult_Mqtt_UseOther](#) = -Pkt::eReason_Error_UseOther , [eResult_Mqtt_ServerMoved](#) = -Pkt::eReason_Error_ServerMoved ,
[eResult_Mqtt_NotSupported_SharedSubs](#) = -Pkt::eReason_Error_NotSupported_SharedSubs ,


```
eResult_Mqtt_ConnRate = -Pkt::eReason_Error_ConnRate , eResult_Mqtt_MaxConnTime = -Pkt::eReason_Error_MaxConnTime , eResult_Mqtt_NotSupported_Sublds = -Pkt::eReason_Error_NotSupported_Sublds , eResult_Mqtt_NotSupported_WildSubs = -Pkt::eReason_Error_NotSupported_WildSubs ,
eResult_Mqtt_Transport_Error = -Pkt::eReason_Error_Transport_Error , eResult_Error_NetErr = -Pkt::eReason_Error_Transport_Error , eResult_Socket_TryAgain = -256 , eResult_Socket_NeedDNS = -257 ,
eResult_Cont_Auth = -258 , eResult_Error_InvalidURI = -259 , eResult_Error_NoneAvail = -260 ,
eResult_Error_Malformed = -261 ,
eResult_Error_Timeout = -262 , eResult_Error_BadProp = -263 , eResult_Error_NoSuchConn = -264 ,
eResult_Error_BadArg = -265 }
```

MQTT library return codes.

- enum {
 eObj_Flag_PubOnWrite = 0x0001 , eObj_Flag_PubQoS_AtLeastOnce = 0x0002 , eObj_Flag_PubQoS_ExactlyOnce = 0x0004 , eObj_Flag_PubRetain = 0x0008 ,
 eObj_Flag_Subscribe = 0x0010 , eObj_Flag_Sub_HandlerOnly = 0x0020 , eObj_Flag_Sub_IncludeLocal = 0x0040 , eObj_Flag_Sub_QoS_AtLeastOnce = 0x0080 ,
 eObj_Flag_Sub_QoS_ExactlyOnce = 0x0100 , eObj_Flag_Sub_NoLocal = 0x0200 , eObj_Flag_Sub_Retained_IfNew = 0x0400 , eObj_Flag_Sub_Retained_Never = 0x0800 ,
 eObj_Flag_WaitForPub = 0x1000 , eObj_Flag_DynTopicName = 0x2000 }

MqttObj Control Flags.

- enum eObj_Serialize_t {
 eObj_Serialize_Disabled = 0x00 , eObj_Serialize_String = 0x01 , eObj_Serialize_JSON = 0x02 ,
 eObj_Serialize_JSON_ValOnly = 0x04 ,
 eObj_Serialize_MqttString = 0x08 , eObj_Serialize_BigEndian = 0x10 , eObj_Serialize_LittleEndian = 0x20 ,
 eObj_Serialize_NativeEndian = 0x40 ,
 eObj_Serialize_CURRENT_FORM = 0x80 }

MqttObj Serialization Formats.

Functions

- const char * [GetPacketTypeString](#) (Pkt::eType_t pkt)

Get the name of a specific eType_t protocol packet type.
- const char * [GetReturnCodeString](#) (MQTT::eResult_t result)

Get the name of a specific API eResult_t return code.
- const char * [GetConnectionStatusString](#) (int status)

Get the name of a specific GetConnectionStatus return code.

15.4.1 Detailed Description

MQTT Namespace.

```
#include <mqtt/mqtt.h>
```

[Client](#) or [ConfiguredClient](#) are the base classes to use MQTT.

[SubHandler](#) and [TopicHandler](#) are event callbacks for the [Client](#).

[ConfigExposer](#) and the mqtt_* classes are "magic" classes that automatically handle pub/sub of the associated object to/from the specified MQTT topic.

Topic Names are also handled as an [NBInterpolatedString](#) and so can be dynamically assigned to NetBurner Config System values using specially formatting strings.

See the [MQTT](#) examples folder for more details.

15.4.2 Typedef Documentation

15.4.2.1 MsgCallback_t

```
typedef int(* MQTT::MsgCallback_t) (MQTT::Client *mConn, int mqttFd, MQTT::PacketInfo::Msg *msg, void *ctx)
```

Function signature for a user-implemented callback function to receive messages from a subscription.

Parameters

<i>*mConn</i>	Pointer to the Mqtt Client that received the message
<i>mqttFd</i>	The file descriptor for the socket the message was received on
<i>*msg</i>	Pointer to the message struct
<i>*ctx</i>	A void pointer passed into the register function. Can be used to reference objects or other data

See also

[TopicHandler](#), [RegisterTopicHandler](#)

15.4.3 Enumeration Type Documentation

15.4.3.1 anonymous enum

`anonymous enum`

MqttObj Control Flags.

Describes how the object should behave in relation to the [Client](#) and normal code.

Enumerator

<code>eObj_Flag_PubOnWrite</code>	Publish object whenever it is written to.
<code>eObj_Flag_PubQoS_AtLeastOnce</code>	Publish using QoS == At Least Once.
<code>eObj_Flag_PubQoS_ExactlyOnce</code>	Publish using QoS == At Most Once.
<code>eObj_Flag_PubRetain</code>	Publish with Retain set.
<code>eObj_Flag_Subscribe</code>	Automatically subscribe to the objects Topic.
<code>eObj_Flag_Sub_HandlerOnly</code>	<++>
<code>eObj_Flag_Sub_IncludeLocal</code>	Receive Publish messages sent by the object's client (aka, receive an Echo)
<code>eObj_Flag_Sub_QoS_AtLeastOnce</code>	Receive Publish messages at QoS == At Least Once.
<code>eObj_Flag_Sub_QoS_ExactlyOnce</code>	Receive Publish messages at QoS == At Most Once.
<code>eObj_Flag_Sub_NoLocal</code>	Do not Receive Local Publish Messages.
<code>eObj_Flag_Sub_Retained_IfNew</code>	When subscribing/connecting, only receive the Retained Publish, if it is new since last subscribed/connected.
<code>eObj_Flag_Sub_Retained_Never</code>	Never receive Retained messages, unless connected and subscribed when they are sent.
<code>eObj_Flag_WaitForPub</code>	Assigning to the object blocks until new value has been published.
<code>eObj_Flag_DynTopicName</code>	<++>

15.4.3.2 eObj_Serialize_t

`enum MQTT::eObj_Serialize_t`

MqttObj Serialization Formats.

Enumerator

<code>eObj_Serialize_Disabled</code>	Do not serialize, used to disable serialization of branches in complex structures.
<code>eObj_Serialize_String</code>	Serialize as a null terminated string.
<code>eObj_Serialize_JSON</code>	Serialize as a JSON name:value pair.
<code>eObj_Serialize_JSON_ValOnly</code>	Serialize as a JSON value.
<code>eObj_Serialize_MqttString</code>	Serialize as a MQTT Protocol level string.
<code>eObj_Serialize_BigEndian</code>	Serialize as binary, Big Endian ordering.

Enumerator

eObj_Serialize_LittleEndian	Serialize as binary, Little Endian ordering.
eObj_Serialize_NativeEndian	Serialize as binary, host native ordering.
eObj_Serialize_CURRENT_FORM	Serialize as the object's current defined form (only valid if the object has a currently assigned format)

15.4.3.3 eResult_t

enum MQTT::eResult_t
MQTT library return codes.

Enumerator

eResult_Success	No error, completed successfully.
eResult_Mqtt_Unspecified	Unknown/Unspecified error.
eResult_Mqtt_MalformedPkt	Malformed Mqtt msg was sent/received.
eResult_Mqtt_Protocol	Unsupported or unknown protocol.
eResult_Mqtt_Implementation	Feature/option not implemented.
eResult_Mqtt_UnsupportedProtoVer	Unsupported MQTT protocol version.
eResult_Mqtt_ClientIdInvalid	Broker refusing current Client ID.
eResult_Mqtt_UserPassword	Invalid Username/password.
eResult_Mqtt_NotAuthorized	Client not authorized.
eResult_Mqtt_ServerUnavail	Broker unavailable.
eResult_Mqtt_ServerBusy	Broker is unable to process message at this time.
eResult_Mqtt_Banned	Client is banned by the broker.
eResult_Mqtt_ServerShutdown	Broker is shutting down.
eResult_Mqtt_BadAuthMethod	Broker does not support the requested authentication method.
eResult_Mqtt_KATimeout	Keep Alive Timeout exceeded.
eResult_Mqtt_SessTakeover	Session has been taken over by another connection.
eResult_Mqtt_FilterInvalid	A submitted request filter is invalid.
eResult_Mqtt_NameInvalid	Submitted Topic Name is invalid.
eResult_Mqtt_PktId_InUse	Broker has an outstanding Reliable QOS Packet with the same ID (sending Subscribe when a Publish with the same ID is in progress, etc.)
eResult_Mqtt_PktId_NotFound	Broker has no outstanding packet with the requested ID.
eResult_Mqtt_RcvMax_Exceeded	< ++ >
eResult_Mqtt_TopicAlias_Invalid	Requested Topic Alias is invalid/unknown.
eResult_Mqtt_TooLarge	Message Size too large.
eResult_Mqtt_MsgRate	Exceeded client rate limit.
eResult_Mqtt_Quota	Exceeded client message quota.
eResult_Mqtt_Admin	General Administrative error.
eResult_Mqtt_PayFmt_Invalid	Payload Format error (malformed message)
eResult_Mqtt_NotSupported_Retain	Retain not supported for prior Publish Message.
eResult_Mqtt_NotSupported_QoS	Requested QoS level not supported for prior Publish Message.
eResult_Mqtt_UseOther	Client should connect to specified Other Broker (equivalent to HTTP 3xx redirect)
eResult_Mqtt_ServerMoved	Broker Permanently Moved to new address.
eResult_Mqtt_NotSupported_SharedSubs	Broker does not support Shared Subscriptions.
eResult_Mqtt_ConnRate	Exceeded Connection Rate Limit.

Enumerator

eResult_Mqtt_MaxConnTime	Exceeded Maximum Connection Time limit.
eResult_Mqtt_NotSupported_SubIds	Broker does not support Subscription IDs.
eResult_Mqtt_NotSupported_WildSubs	Broker does not support Wildcard Subscriptions.
eResult_Mqtt_Transport_Error	Transport layer error.
eResult_Error_NetErr	General network layer error.
eResult_Socket_TryAgain	Client Network Socket error, try again.
eResult_Socket_NeedDNS	Broker address failed to resolve DNS.
eResult_Cont_Auth	<++>
eResult_Error_InvalidURI	Provided Broker address URI is invalid.
eResult_Error_NoneAvail	Required resource exhausted, none available.
eResult_Error_Malformed	Malformed argument.
eResult_Error_Timeout	Operation timed out.
eResult_Error_BadProp	Argument contains malformed Message Property.
eResult_Error_NoSuchConn	No such connection.
eResult_Error_BadArg	General Bad Argument.

15.4.4 Function Documentation

15.4.4.1 GetConnectionStatusString()

```
const char * MQTT::GetConnectionStatusString (
    int status)
```

Get the name of a specific GetConnectionStatus return code.

Parameters

<i>status</i>	The integer output from GetConnectionStatus()

Returns

A pointer to the string containing the return code name.

See also

[Client::Connect](#), [Client::Disconnect](#), [Client::GetConnectionStatus](#)

15.4.4.2 GetPacketTypeString()

```
const char * MQTT::GetPacketTypeString (
    Pkt::eType_t pkt)
```

Get the name of a specific eType_t protocol packet type.

Parameters

<i>pkt</i>	The protocol packet type to get the name of.

Returns

A pointer to the string containing the packet type name.

See also

[GetReturnCodeString](#)

15.4.4.3 GetReturnCodeString()

```
const char * MQTT::GetReturnCodeString (
    MQTT::eResult_t result)
```

Get the name of a specific API eResult_t return code.

Parameters

<i>result</i>	The return code to get the name of.

Returns

A pointer to the string containing the return code name.

See also

[GetReturnCodeString](#)

15.5 MQTT::PacketInfo Namespace Reference

[MQTT PacketInfo](#) Namespace.

Classes

- struct [Msg](#)
PacketInfo::Msg message data structure.

15.5.1 Detailed Description

[MQTT PacketInfo](#) Namespace.

15.6 NB::Error Namespace Reference

Namespaces

- namespace [Connect](#)
- namespace [Init](#)
- namespace [Scan](#)

Enumerations

- enum [GeneralErrors](#) {
[NoError](#) = 0 , [Timeout](#) = -256 , [BusTimeout](#) = -257 , [InvalidArgument](#) = -258 ,
[TooManyPendingCommands](#) = -259 , [InvalidRequest](#) = -512 }

15.6.1 Detailed Description

[NB::Error](#) namespace that describes WiFi request errors. See also [InitWifi_SPI\(\)](#), [InitAP_SPI\(\)](#), [InitWifi_Serial\(\)](#), [NB::Master::ConnectToAP\(\)](#), [NB::Master::StartAP\(\)](#), [WifiInitScanAndShow_SPI\(\)](#), [WifiInitScanAndShow_Serial\(\)](#)

15.6.2 Enumeration Type Documentation

15.6.2.1 GeneralErrors

```
enum NB::Error::GeneralErrors
```

15.6.3 GeneralErrors

8 bit values reserved for command specific errors

Enumerator

NoError	Successful execution.
Timeout	Request timed out.
BusTimeout	Request timed out on the bus.
InvalidArgument	Parameter/argument not defined or is invalid.
TooManyPendingCommands	Too many pending commands.
InvalidRequest	Invalid request.

15.7 NB::Error::Connect Namespace Reference

Enumerations

- enum [ConnectErrors](#) {
[Success](#) = 0 , [NotInitialized](#) = -1 , [AlreadyConnected](#) = -2 , [Option](#) = -3 ,
[CouldNotConfig](#) = -4 , [SSID_NotFound](#) = -5 , [BSSID_NotFound](#) = -6 , [Sec_NotFound](#) = -7 ,
[Cipher_NotFound](#) = -8 , [ConnectFailed](#) = -9 }

15.7.1 Detailed Description

[NB::Error::Connect](#) namespace that describes WiFi connect request errors. See also [InitWifi_SPI\(\)](#), [InitAP_SPI\(\)](#), [InitWifi_Serial\(\)](#), [NB::Master::ConnectToAP\(\)](#), [NB::Master::StartAP\(\)](#)

15.7.2 Enumeration Type Documentation

15.7.2.1 ConnectErrors

```
enum NB::Error::Connect::ConnectErrors
```

15.7.3 ConnectErrors

Enumerator

Success	Success.
NotInitialized	Not Initialized.
AlreadyConnected	Already Connected.
Option	Option.
CouldNotConfig	Could Not Configure.
SSID_NotFound	SSID Not Found.
BSSID_NotFound	BSSID Not Found.
Sec_NotFound	Sec Not Found.
Cipher_NotFound	Cipher Not Found.
ConnectFailed	Connect Failed.

15.8 NB::Error::Init Namespace Reference

Enumerations

- enum [InitializationErrors](#) {
[Success](#) = 0 , [AlreadyInit](#) = -1 , [NoDevice](#) = -2 , [InvalidInfo](#) = -3 ,
[DevFirmVer](#) = -4 , [DevHwVer](#) = -5 , [OptionTables](#) = -6 }

15.8.1 Detailed Description

[NB::Error::Init](#) namespace that describes WiFi initialization request errors. See also [WifiInitScanAndShow_SPI\(\)](#), [WifiInitScanAndShow_Serial\(\)](#), [InitWifi_SPI\(\)](#), [InitAP_SPI\(\)](#), [InitWifi_Serial\(\)](#)

15.8.2 Enumeration Type Documentation

15.8.2.1 InitializationErrors

enum [NB::Error::Init::InitializationErrors](#)

15.8.3 InitializationErrors

Enumerator

Success	Success.
AlreadyInit	Already Initialized.
NoDevice	No Device detected or attempting to use WiFi functions on an un-initialized device.
InvalidInfo	Invalid Info.
DevFirmVer	Device Firmware Version.
DevHwVer	Device Hardware Version.
OptionTables	Option Tables.

15.9 NB::Error::Scan Namespace Reference

Enumerations

- enum [ScanErrors](#) { [Success](#) = 0 , [NotInitialized](#) = -1 , [InProgress](#) = -2 , [Option](#) = -3 }

15.9.1 Detailed Description

[NB::Error::Scan](#) namespace that describes WiFi access point scan request errors. See also [WifiInitScanAndShow_SPI\(\)](#), [WifiInitScanAndShow_Serial\(\)](#)

15.9.2 Enumeration Type Documentation

15.9.2.1 ScanErrors

enum [NB::Error::Scan::ScanErrors](#)

15.9.3 ScanErrors

Enumerator

Success	Success.
NotInitialized	Not Initialized.
InProgress	In Progress.
Option	Option.

15.10 ReceiveUDP Namespace Reference

Functions

- [signal_handler](#) (sig, frame)
- [send_test_packet](#) (host, port, message)

15.10.1 Detailed Description

UDP Packet Receiver and Display Tool

This application listens for UDP packets on a specified port and displays their contents in real-time with timestamps and sender information.

15.10.2 Function Documentation

15.10.2.1 `send_test_packet()`

```
ReceiveUDP.send_test_packet (  
    host,  
    port,  
    message)
```

Send a test UDP packet (for testing purposes)

15.10.2.2 `signal_handler()`

```
ReceiveUDP.signal_handler (  
    sig,  
    frame)
```

Handle Ctrl+C gracefully

Chapter 16

Class Documentation

16.1 `_edma_channel_Preemption_config` Struct Reference

eDMA channel priority configuration

```
#include <fsl_edma.h>
```

Public Attributes

- bool [enableChannelPreemption](#)
- bool [enablePreemptAbility](#)
- uint8_t [channelPriority](#)

16.1.1 Detailed Description

eDMA channel priority configuration

16.1.2 Member Data Documentation

16.1.2.1 `channelPriority`

```
uint8_t _edma_channel_Preemption_config::channelPriority
```

Channel priority

16.1.2.2 `enableChannelPreemption`

```
bool _edma_channel_Preemption_config::enableChannelPreemption
```

If true: a channel can be suspended by other channel with higher priority

16.1.2.3 `enablePreemptAbility`

```
bool _edma_channel_Preemption_config::enablePreemptAbility
```

If true: a channel can suspend other channel with low priority

The documentation for this struct was generated from the following file:

- `fsl_edma.h`

16.2 `_edma_config` Struct Reference

eDMA global configuration structure.

```
#include <fsl_edma.h>
```

Public Attributes

- bool [enableContinuousLinkMode](#)
- bool [enableHaltOnError](#)

- bool [enableRoundRobinArbitration](#)
- bool [enableDebugMode](#)

16.2.1 Detailed Description

eDMA global configuration structure.

16.2.2 Member Data Documentation

16.2.2.1 enableContinuousLinkMode

```
bool _edma_config::enableContinuousLinkMode
```

Enable (true) continuous link mode. Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

16.2.2.2 enableDebugMode

```
bool _edma_config::enableDebugMode
```

Enable(true) eDMA debug mode. When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete.

16.2.2.3 enableHaltOnError

```
bool _edma_config::enableHaltOnError
```

Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

16.2.2.4 enableRoundRobinArbitration

```
bool _edma_config::enableRoundRobinArbitration
```

Enable (true) round robin channel arbitration method or fixed priority arbitration is used for channel selection

The documentation for this struct was generated from the following file:

- [fsl_edma.h](#)

16.3 _edma_handle Struct Reference

eDMA transfer handle structure

```
#include <fsl_edma.h>
```

Public Attributes

- edma_callback [callback](#)
- void * [userData](#)
- DMA_Type * [base](#)
- [edma_tcd_t](#) * [tcdPool](#)
- uint8_t [channel](#)
- volatile int8_t [header](#)
- volatile int8_t [tail](#)
- volatile int8_t [tcdUsed](#)
- volatile int8_t [tcdSize](#)
- uint8_t [flags](#)

16.3.1 Detailed Description

eDMA transfer handle structure

16.3.2 Member Data Documentation

16.3.2.1 base

DMA_Type* _edma_handle::base
eDMA peripheral base address.

16.3.2.2 callback

edma_callback _edma_handle::callback
Callback function for major count exhausted.

16.3.2.3 channel

uint8_t _edma_handle::channel
eDMA channel number.

16.3.2.4 flags

uint8_t _edma_handle::flags
The status of the current channel.

16.3.2.5 header

volatile int8_t _edma_handle::header
The first TCD index. Should point to the next TCD to be loaded into the eDMA engine.

16.3.2.6 tail

volatile int8_t _edma_handle::tail
The last TCD index. Should point to the next TCD to be stored into the memory pool.

16.3.2.7 tcdPool

edma_tcd_t* _edma_handle::tcdPool
Pointer to memory stored TCDs.

16.3.2.8 tcdSize

volatile int8_t _edma_handle::tcdSize
The total number of TCD slots in the queue.

16.3.2.9 tcdUsed

volatile int8_t _edma_handle::tcdUsed
The number of used TCD slots. Should reflect the number of TCDs can be used/loaded in the memory.

16.3.2.10 userData

void* _edma_handle::userData
Callback function parameter.
The documentation for this struct was generated from the following file:

- fsl_edma.h

16.4 _edma_minor_offset_config Struct Reference

eDMA minor offset configuration
#include <fsl_edma.h>

Public Attributes

- bool [enableSrcMinorOffset](#)
- bool [enableDestMinorOffset](#)
- uint32_t [minorOffset](#)

16.4.1 Detailed Description

eDMA minor offset configuration

16.4.2 Member Data Documentation**16.4.2.1 enableDestMinorOffset**

bool `_edma_minor_offset_config::enableDestMinorOffset`

Enable(true) or Disable(false) destination minor loop offset.

16.4.2.2 enableSrcMinorOffset

bool `_edma_minor_offset_config::enableSrcMinorOffset`

Enable(true) or Disable(false) source minor loop offset.

16.4.2.3 minorOffset

uint32_t `_edma_minor_offset_config::minorOffset`

Offset for a minor loop mapping.

The documentation for this struct was generated from the following file:

- `fsl_edma.h`

16.5 _edma_tcd Struct Reference

eDMA TCD.

```
#include <fsl_edma.h>
```

Public Attributes

- __IO uint32_t [SADDR](#)
- __IO uint16_t [SOFF](#)
- __IO uint16_t [ATTR](#)
- __IO uint32_t [NBYTES](#)
- __IO uint32_t [SLAST](#)
- __IO uint32_t [DADDR](#)
- __IO uint16_t [DOFF](#)
- __IO uint16_t [CITER](#)
- __IO uint32_t [DLAST_SGA](#)
- __IO uint16_t [CSR](#)
- __IO uint16_t [BITER](#)

16.5.1 Detailed Description

eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

16.5.2 Member Data Documentation

16.5.2.1 ATTR

`__IO uint16_t _edma_tcd::ATTR`

ATTR register, source/destination transfer size and modulo

16.5.2.2 BITER

`__IO uint16_t _edma_tcd::BITER`

BITER register, begin minor loop count.

16.5.2.3 CITER

`__IO uint16_t _edma_tcd::CITER`

CITER register, current minor loop numbers, for unfinished minor loop.

16.5.2.4 CSR

`__IO uint16_t _edma_tcd::CSR`

CSR register, for TCD control status

16.5.2.5 DADDR

`__IO uint32_t _edma_tcd::DADDR`

DADDR register, used for destination address

16.5.2.6 DLAST_SGA

`__IO uint32_t _edma_tcd::DLAST_SGA`

DLASTSGA register, next tcd address used in scatter-gather mode

16.5.2.7 DOFF

`__IO uint16_t _edma_tcd::DOFF`

DOFF register, used for destination offset

16.5.2.8 NBYTES

`__IO uint32_t _edma_tcd::NBYTES`

Nbytes register, minor loop length in bytes

16.5.2.9 SADDR

`__IO uint32_t _edma_tcd::SADDR`

SADDR register, used to save source address

16.5.2.10 SLAST

`__IO uint32_t _edma_tcd::SLAST`

SLAST register

16.5.2.11 SOFF

`__IO uint16_t _edma_tcd::SOFF`

SOFF register, save offset bytes every transfer

The documentation for this struct was generated from the following file:

- fsl_edma.h

16.6 `_edma_transfer_config` Struct Reference

eDMA transfer configuration

```
#include <fsl_edma.h>
```

Public Attributes

- `uint32_t` [srcAddr](#)
- `uint32_t` [destAddr](#)
- `edma_transfer_size_t` [srcTransferSize](#)
- `edma_transfer_size_t` [destTransferSize](#)
- `int16_t` [srcOffset](#)
- `int16_t` [destOffset](#)
- `uint32_t` [minorLoopBytes](#)
- `uint32_t` [majorLoopCounts](#)

16.6.1 Detailed Description

eDMA transfer configuration

This structure configures the source/destination transfer attribute.

16.6.2 Member Data Documentation

16.6.2.1 `destAddr`

```
uint32_t _edma_transfer_config::destAddr
```

Destination data address.

16.6.2.2 `destOffset`

```
int16_t _edma_transfer_config::destOffset
```

Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

16.6.2.3 `destTransferSize`

```
edma_transfer_size_t _edma_transfer_config::destTransferSize
```

Destination data transfer size.

16.6.2.4 `majorLoopCounts`

```
uint32_t _edma_transfer_config::majorLoopCounts
```

Major loop iteration count.

16.6.2.5 `minorLoopBytes`

```
uint32_t _edma_transfer_config::minorLoopBytes
```

Bytes to transfer in a minor loop

16.6.2.6 `srcAddr`

```
uint32_t _edma_transfer_config::srcAddr
```

Source data address.

16.6.2.7 `srcOffset`

```
int16_t _edma_transfer_config::srcOffset
```

Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.

16.6.2.8 srcTransferSize

edma_transfer_size_t _edma_transfer_config::srcTransferSize

Source data transfer size.

The documentation for this struct was generated from the following file:

- fsl_edma.h

16.7 _EthernetFrameHeader Struct Reference

Ethernet Type II Frame Header.

```
#include <ethernet.h>
```

Public Attributes

- MACADDRESS_48 **destinationMacAddress**
Destination MAC address.
- MACADDRESS_48 **sourceMacAddress**
Source MAC address.
- beuint16_t **etherType**
Protocol.

16.7.1 Detailed Description

Ethernet Type II Frame Header.

The documentation for this struct was generated from the following file:

- ethernet.h

16.8 _flexcan_config Struct Reference

FlexCAN module configuration structure.

```
#include <fsl_flexcan.h>
```

Public Attributes

- flexcan_clock_source_t clkSrc
- flexcan_wake_up_source_t wakeupSrc
- uint8_t maxMbNum
- bool enableLoopBack
- bool enableTimerSync
- bool enableSelfWakeup
- bool enableIndividMask
- bool disableSelfReception
- bool enableListenOnlyMode
- bool enableSupervisorMode

16.8.1 Detailed Description

FlexCAN module configuration structure.

Deprecated Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

16.8.2 Member Data Documentation

16.8.2.1 baudRate

`uint32_t _flexcan_config::baudRate`

FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

16.8.2.2 bitRate

`uint32_t _flexcan_config::bitRate`

FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

16.8.2.3 clkSrc

`flexcan_clock_source_t _flexcan_config::clkSrc`

Clock source for FlexCAN Protocol Engine.

16.8.2.4 disableSelfReception

`bool _flexcan_config::disableSelfReception`

Enable or Disable Self Reflection.

16.8.2.5 enableIndividMask

`bool _flexcan_config::enableIndividMask`

Enable or Disable Rx Individual Mask and Queue feature.

16.8.2.6 enableListenOnlyMode

`bool _flexcan_config::enableListenOnlyMode`

Enable or Disable Listen Only Mode.

16.8.2.7 enableLoopBack

`bool _flexcan_config::enableLoopBack`

Enable or Disable Loop Back Self Test Mode.

16.8.2.8 enableSelfWakeup

`bool _flexcan_config::enableSelfWakeup`

Enable or Disable Self Wakeup Mode.

16.8.2.9 enableSupervisorMode

`bool _flexcan_config::enableSupervisorMode`

Enable or Disable Supervisor Mode, enable this mode will make registers allow only Supervisor access.

16.8.2.10 enableTimerSync

`bool _flexcan_config::enableTimerSync`

Enable or Disable Timer Synchronization.

16.8.2.11 maxMbNum

`uint8_t _flexcan_config::maxMbNum`

The maximum number of Message Buffers used by user.

16.8.2.12 wakeupSrc

`flexcan_wake_up_source_t _flexcan_config::wakeupSrc`

Wake up source selection.

The documentation for this struct was generated from the following file:

- FlexcanLoopback/src/fsl_flexcan.h

16.9 _flexcan_fifo_transfer Struct Reference

FlexCAN Rx FIFO transfer.

```
#include <fsl_flexcan.h>
```

Public Attributes

- `flexcan_frame_t* frame`
- `size_t frameNum`

16.9.1 Detailed Description

FlexCAN Rx FIFO transfer.

16.9.2 Member Data Documentation

16.9.2.1 frame

`flexcan_frame_t* _flexcan_fifo_transfer::frame`

The buffer of CAN Message to be received from Legacy Rx FIFO.

16.9.2.2 frameNum

`size_t _flexcan_fifo_transfer::frameNum`

Number of CAN Message need to be received from Legacy or Enhanced Rx FIFO.

The documentation for this struct was generated from the following file:

- FlexcanLoopback/src/fsl_flexcan.h

16.10 _flexcan_frame Struct Reference

FlexCAN message frame structure.

```
#include <fsl_flexcan.h>
```

16.10.1 Detailed Description

FlexCAN message frame structure.

16.10.2 Member Data Documentation

16.10.2.1 __pad0__

`uint32_t _flexcan_frame::__pad0__`

Reserved.

16.10.2.2 dataByte0

`uint8_t _flexcan_frame::dataByte0`

CAN Frame payload byte0.

16.10.2.3 dataByte1

`uint8_t _flexcan_frame::dataByte1`
CAN Frame payload byte1.

16.10.2.4 dataByte2

`uint8_t _flexcan_frame::dataByte2`
CAN Frame payload byte2.

16.10.2.5 dataByte3

`uint8_t _flexcan_frame::dataByte3`
CAN Frame payload byte3.

16.10.2.6 dataByte4

`uint8_t _flexcan_frame::dataByte4`
CAN Frame payload byte4.

16.10.2.7 dataByte5

`uint8_t _flexcan_frame::dataByte5`
CAN Frame payload byte5.

16.10.2.8 dataByte6

`uint8_t _flexcan_frame::dataByte6`
CAN Frame payload byte6.

16.10.2.9 dataByte7

`uint8_t _flexcan_frame::dataByte7`
CAN Frame payload byte7.

16.10.2.10 dataWord0

`uint32_t _flexcan_frame::dataWord0`
CAN Frame payload word0.

16.10.2.11 dataWord1

`uint32_t _flexcan_frame::dataWord1`
CAN Frame payload word1.

16.10.2.12 format

`uint32_t _flexcan_frame::format`
CAN Frame Identifier(STD or EXT format).

16.10.2.13 id

`uint32_t _flexcan_frame::id`
CAN Frame Identifier, should be set using [FLEXCAN_ID_EXT\(\)](#) or [FLEXCAN_ID_STD\(\)](#) macro.

16.10.2.14 idhit

`uint32_t _flexcan_frame::idhit`
CAN Rx FIFO filter hit [id](#)(This value is only used in Rx FIFO receive mode).

16.10.2.15 length

```
uint32_t _flexcan_frame::length
```

CAN frame data length in bytes (Range: 0~8).

16.10.2.16 timestamp

```
uint32_t _flexcan_frame::timestamp
```

FlexCAN internal Free-Running Counter Time Stamp.

16.10.2.17 type

```
uint32_t _flexcan_frame::type
```

CAN Frame Type(DATA or REMOTE).

The documentation for this struct was generated from the following file:

- FlexcanLoopback/src/fsl_flexcan.h

16.11 _flexcan_handle Struct Reference

FlexCAN handle structure.

```
#include <fsl_flexcan.h>
```

Public Attributes

- flexcan_transfer_callback_t [callback](#)
- void * [userData](#)
- [flexcan_frame_t](#) *volatile [mbFrameBuf](#) [CAN_WORD1_COUNT]
- [flexcan_frame_t](#) *volatile [rxFifoFrameBuf](#)
- size_t [rxFifoFrameNum](#)
- size_t [rxFifoTransferTotalNum](#)
- volatile uint8_t [mbState](#) [CAN_WORD1_COUNT]
- volatile uint8_t [rxFifoState](#)
- volatile uint32_t [timestamp](#) [CAN_WORD1_COUNT]

16.11.1 Detailed Description

FlexCAN handle structure.

16.11.2 Member Data Documentation**16.11.2.1 callback**

```
flexcan_transfer_callback_t _flexcan_handle::callback
```

Callback function.

16.11.2.2 mbFrameBuf

```
flexcan\_frame\_t* volatile _flexcan_handle::mbFrameBuf [CAN_WORD1_COUNT]
```

The buffer for received CAN data from Message Buffers.

16.11.2.3 mbState

```
volatile uint8_t _flexcan_handle::mbState [CAN_WORD1_COUNT]
```

Message Buffer transfer state.

16.11.2.4 rxFifoFrameBuf

```
flexcan\_frame\_t* volatile _flexcan_handle::rxFifoFrameBuf
```

The buffer for received CAN data from Legacy Rx FIFO.

16.11.2.5 rxFifoFrameNum

size_t _flexcan_handle::rxFifoFrameNum

The number of CAN messages remaining to be received from Legacy or Enhanced Rx FIFO.

16.11.2.6 rxFifoState

volatile uint8_t _flexcan_handle::rxFifoState

Rx FIFO transfer state.

16.11.2.7 rxFifoTransferTotalNum

size_t _flexcan_handle::rxFifoTransferTotalNum

Total CAN Message number need to be received from Legacy or Enhanced Rx FIFO.

16.11.2.8 timestamp

volatile uint32_t _flexcan_handle::timestamp[CAN_WORD1_COUNT]

Mailbox transfer timestamp.

16.11.2.9 userData

void* _flexcan_handle::userData

FlexCAN callback function parameter.

The documentation for this struct was generated from the following file:

- FlexcanLoopback/src/fsl_flexcan.h

16.12 _flexcan_mb_transfer Struct Reference

FlexCAN Message Buffer transfer.

```
#include <fsl_flexcan.h>
```

Public Attributes

- [flexcan_frame_t](#) * frame
- uint8_t mbIdx

16.12.1 Detailed Description

FlexCAN Message Buffer transfer.

16.12.2 Member Data Documentation

16.12.2.1 frame

[flexcan_frame_t](#)* _flexcan_mb_transfer::frame

The buffer of CAN Message to be transfer.

16.12.2.2 mbIdx

uint8_t _flexcan_mb_transfer::mbIdx

The index of Message buffer used to transfer Message.

The documentation for this struct was generated from the following file:

- FlexcanLoopback/src/fsl_flexcan.h

16.13 _flexcan_rx_fifo_config Struct Reference

FlexCAN Legacy Rx FIFO configuration structure.

```
#include <fsl_flexcan.h>
```

Public Attributes

- [uint32_t * idFilterTable](#)
- [uint8_t idFilterNum](#)
- [flexcan_rx_fifo_filter_type_t idFilterType](#)
- [flexcan_rx_fifo_priority_t priority](#)

16.13.1 Detailed Description

FlexCAN Legacy Rx FIFO configuration structure.

16.13.2 Member Data Documentation

16.13.2.1 idFilterNum

```
uint8_t _flexcan_rx_fifo_config::idFilterNum
```

The FlexCAN Legacy Rx FIFO Filter elements quantity.

16.13.2.2 idFilterTable

```
uint32_t* _flexcan_rx_fifo_config::idFilterTable
```

Pointer to the FlexCAN Legacy Rx FIFO identifier filter table.

16.13.2.3 idFilterType

```
flexcan_rx_fifo_filter_type_t _flexcan_rx_fifo_config::idFilterType
```

The FlexCAN Legacy Rx FIFO Filter type.

16.13.2.4 priority

```
flexcan_rx_fifo_priority_t _flexcan_rx_fifo_config::priority
```

The FlexCAN Legacy Rx FIFO receive priority.

The documentation for this struct was generated from the following file:

- FlexcanLoopback/src/fsl_flexcan.h

16.14 _flexcan_rx_mb_config Struct Reference

FlexCAN Receive Message Buffer configuration structure.

```
#include <fsl_flexcan.h>
```

Public Attributes

- [uint32_t id](#)
- [flexcan_frame_format_t format](#)
- [flexcan_frame_type_t type](#)

16.14.1 Detailed Description

FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of [FLEXCAN_SetRxMbConfig\(\)](#) function. The [FLEXCAN_SetRxMbConfig\(\)](#) function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

16.14.2 Member Data Documentation

16.14.2.1 format

`flexcan_frame_format_t _flexcan_rx_mb_config::format`
CAN Frame Identifier [format\(Standard of Extend\)](#).

16.14.2.2 id

`uint32_t _flexcan_rx_mb_config::id`
CAN Message Buffer Frame Identifier, should be set using [FLEXCAN_ID_EXT\(\)](#) or [FLEXCAN_ID_STD\(\)](#) macro.

16.14.2.3 type

`flexcan_frame_type_t _flexcan_rx_mb_config::type`
CAN Frame Type(Data or Remote).
The documentation for this struct was generated from the following file:

- FlexcanLoopback/src/fsl_flexcan.h

16.15 _flexcan_timing_config Struct Reference

FlexCAN protocol timing characteristic configuration structure.
`#include <fsl_flexcan.h>`

Public Attributes

- `uint16_t preDivider`
- `uint8_t rJumpwidth`
- `uint8_t phaseSeg1`
- `uint8_t phaseSeg2`
- `uint8_t propSeg`

16.15.1 Detailed Description

FlexCAN protocol timing characteristic configuration structure.

16.15.2 Member Data Documentation

16.15.2.1 phaseSeg1

`uint8_t _flexcan_timing_config::phaseSeg1`
Classic CAN or CAN FD nominal phase Segment 1.

16.15.2.2 phaseSeg2

`uint8_t _flexcan_timing_config::phaseSeg2`
Classic CAN or CAN FD nominal phase Segment 2.

16.15.2.3 preDivider

`uint16_t _flexcan_timing_config::preDivider`
Classic CAN or CAN FD nominal phase bit rate prescaler.

16.15.2.4 propSeg

`uint8_t _flexcan_timing_config::propSeg`
Classic CAN or CAN FD nominal phase Propagation Segment.

16.15.2.5 rJumpwidth

uint8_t _flexcan_timing_config::rJumpwidth

Classic CAN or CAN FD nominal phase Re-sync Jump Width.

The documentation for this struct was generated from the following file:

- FlexcanLoopback/src/fsl_flexcan.h

16.16 _FlexSPIConfig Struct Reference

FlexSPI Memory Configuration Block.

```
#include <evkmimxrt1060_flexspi_nor_config.h>
```

Public Attributes

- uint32_t **tag**
[0x000-0x003] Tag, fixed value 0x42464346UL
- uint32_t **version**
[0x004-0x007] Version, [31:24] - 'V', [23:16] - Major, [15:8] - Minor, [7:0] - bugfix
- uint32_t **reserved0**
[0x008-0x00b] Reserved for future use
- uint8_t **readSampleClkSrc**
[0x00c-0x00c] Read Sample Clock Source, valid value: 0/1/3
- uint8_t **csHoldTime**
[0x00d-0x00d] CS hold time, default value: 3
- uint8_t **csSetupTime**
[0x00e-0x00e] CS setup time, default value: 3
- uint8_t **columnAddressWidth**
- uint8_t **deviceModeCfgEnable**
Serial NAND, need to refer to datasheet.
- uint8_t **deviceModeType**
- uint16_t **waitTimeCfgCommands**
Generic configuration, etc.
- flexspi_lut_seq_t **deviceModeSeq**
DPI/QPI/OPI switch or reset command.
- uint32_t **deviceModeArg**
sequence number, [31:16] Reserved
- uint8_t **configCmdEnable**
[0x01c-0x01c] Configure command Enable Flag, 1 - Enable, 0 - Disable
- uint8_t **configModeType** [3]
[0x01d-0x01f] Configure Mode Type, similar as deviceModeType
- flexspi_lut_seq_t **configCmdSeqs** [3]
[0x020-0x02b] Sequence info for Device Configuration command, similar as deviceModeSeq
- uint32_t **reserved1**
[0x02c-0x02f] Reserved for future use
- uint32_t **configCmdArgs** [3]
[0x030-0x03b] Arguments/Parameters for device Configuration commands
- uint32_t **reserved2**
[0x03c-0x03f] Reserved for future use
- uint32_t **controllerMiscOption**
- uint8_t **deviceType**
details
- uint8_t **sflashPadType**

- [0x045-0x045] Serial Flash Pad Type: 1 - Single, 2 - Dual, 4 - Quad, 8 - Octal*
- uint8_t [serialClkFreq](#)
- uint8_t [lutCustomSeqEnable](#)
- Chapter for more details.*
- uint32_t [reserved3](#) [2]
- be done using 1 LUT sequence, currently, only applicable to HyperFLASH*
- uint32_t **sflashA1Size**
- [0x050-0x053] Size of Flash connected to A1*
- uint32_t **sflashA2Size**
- [0x054-0x057] Size of Flash connected to A2*
- uint32_t **sflashB1Size**
- [0x058-0x05b] Size of Flash connected to B1*
- uint32_t **sflashB2Size**
- [0x05c-0x05f] Size of Flash connected to B2*
- uint32_t **csPadSettingOverride**
- [0x060-0x063] CS pad setting override value*
- uint32_t **sclkPadSettingOverride**
- [0x064-0x067] SCK pad setting override value*
- uint32_t **dataPadSettingOverride**
- [0x068-0x06b] data pad setting override value*
- uint32_t **dqsPadSettingOverride**
- [0x06c-0x06f] DQS pad setting override value*
- uint32_t **timeoutInMs**
- [0x070-0x073] Timeout threshold for read status command*
- uint32_t **commandInterval**
- [0x074-0x077] CS deselect interval between two commands*
- uint16_t **dataValidTime** [2]
- [0x078-0x07b] CLK edge to data valid time for PORT A and PORT B, in terms of 0.1ns*
- uint16_t **busyOffset**
- [0x07c-0x07d] Busy offset, valid value: 0-31*
- uint16_t [busyBitPolarity](#)
- uint32_t [lookupTable](#) [64]
- busy flag is 0 when flash device is busy*
- [flexspi_lut_seq_t](#) **lutCustomSeq** [12]
- [0x180-0x1af] Customizable LUT Sequences*
- uint32_t **reserved4** [4]
- [0x1b0-0x1bf] Reserved for future use*

16.16.1 Detailed Description

FlexSPI Memory Configuration Block.

16.16.2 Member Data Documentation

16.16.2.1 busyBitPolarity

uint16_t [_FlexSPIConfig::busyBitPolarity](#)

[0x07e-0x07f] Busy flag polarity, 0 - busy flag is 1 when flash device is busy, 1 -

16.16.2.2 columnAddressWidth

uint8_t [_FlexSPIConfig::columnAddressWidth](#)

[0x00f-0x00f] Column Address with, for HyperBus protocol, it is fixed to 3, For

16.16.2.3 controllerMiscOption

uint32_t _FlexSPIConfig::controllerMiscOption

[0x040-0x043] Controller Misc Options, see Misc feature bit definitions for more

16.16.2.4 deviceModeArg

uint32_t _FlexSPIConfig::deviceModeArg

sequence number, [31:16] Reserved

[0x018-0x01b] Argument/Parameter for device configuration

16.16.2.5 deviceModeCfgEnable

uint8_t _FlexSPIConfig::deviceModeCfgEnable

Serial NAND, need to refer to datasheet.

[0x010-0x010] Device Mode Configure enable flag, 1 - Enable, 0 - Disable

16.16.2.6 deviceModeSeq

flexspi_lut_seq_t _FlexSPIConfig::deviceModeSeq

DPI/QPI/OPI switch or reset command.

[0x014-0x017] Device mode sequence info, [7:0] - LUT sequence id, [15:8] - LUT

16.16.2.7 deviceModeType

uint8_t _FlexSPIConfig::deviceModeType

[0x011-0x011] Specify the configuration command type:Quad Enable, DPI/QPI/OPI switch,

16.16.2.8 deviceType

uint8_t _FlexSPIConfig::deviceType

details

[0x044-0x044] Device Type: See Flash Type Definition for more details

16.16.2.9 lookupTable

uint32_t _FlexSPIConfig::lookupTable[64]

busy flag is 0 when flash device is busy

[0x080-0x17f] Lookup table holds Flash command sequences

16.16.2.10 lutCustomSeqEnable

uint8_t _FlexSPIConfig::lutCustomSeqEnable

Chapter for more details.

[0x047-0x047] LUT customization Enable, it is required if the program/erase cannot

16.16.2.11 reserved3

uint32_t _FlexSPIConfig::reserved3[2]

be done using 1 LUT sequence, currently, only applicable to HyperFLASH

[0x048-0x04f] Reserved for future use

16.16.2.12 serialClkFreq

uint8_t _FlexSPIConfig::serialClkFreq

[0x046-0x046] Serial Flash Frequency, device specific definitions, See System Boot

16.16.2.13 waitTimeCfgCommands

```
uint16_t _FlexSPIConfig::waitTimeCfgCommands
```

Generic configuration, etc.

[0x012-0x013] Wait time for all configuration commands, unit: 100us, Used for

The documentation for this struct was generated from the following file:

- evkmimxrt1060_flexspi_nor_config.h

16.17 _IpHeaderIPv4 Struct Reference

IP Header Structure (IPv4)

```
#include <ip.h>
```

Public Attributes

- **uint8_t versionNLength**
Version (4 bits) and header length (4 bits). Version=4 for IPv4, length in 32-bit words [IPPKT::bVerHdrLen](#).
- **uint8_t diffServNEcn**
Differentiated Services Code Point (6 bits) and Explicit Congestion Notification (2 bits) - used for QoS [IPPKT::bTos](#).
- **beuint16_t totalLength**
Total datagram size in bytes including IP header and data payload [IPPKT::wLength](#).
- **beuint16_t identification**
Fragment identification field - same for all fragments of a datagram [IPPKT::wpktId](#).
- **beuint16_t flagsNFragmentOffset**
Flags (3 bits) and fragment offset (13 bits) - offset in 8-byte units [IPPKT::wFlags_Frag](#).
- **uint8_t timeToLive**
Time To Live - typically used as hop count, decremented by each router [IPPKT::bTTL](#).
- **uint8_t protocol**
IANA protocol number identifying next-level protocol (TCP=6, UDP=17, ICMP=1) [IPPKT::proto](#).
- **beuint16_t headerChecksum**
Header checksum - 16-bit one's complement checksum of IP header only (not data) [IPPKT::hCSum](#).
- **beuint32_t sourceAddress**
Source IP address as 4 octets in network byte order [IPPKT::ipSrc](#).
- **beuint32_t destinationAddress**
Destination IP address as 4 octets in network byte order [IPPKT::ipDst](#).

16.17.1 Detailed Description

IP Header Structure (IPv4)

Standard IPv4 packet header as defined in IETF RFC 791 (September 1981) and MIL-STD-1777. This structure represents the 20-byte minimum IP header. IP options, if present, follow this structure and are indicated by a header length value greater than 5.

All multi-byte fields are in network byte order (big-endian) and use the beuint types for automatic endianness handling.

Note

The structure is packed to ensure proper alignment with network packet data.

Standard header is 20 bytes when no options are present (header length = 5).

Maximum header size is 60 bytes (header length = 15) when options are included.

See also

[IPPKT](#) for the internal representation used within NetBurner stack

The documentation for this struct was generated from the following file:

- ip.h

16.18 _lpspi_master_config Struct Reference

LPSPi master configuration structure.

```
#include <fsl_lpspi.h>
```

Public Attributes

- uint32_t [baudRate](#)
- uint32_t [bitsPerFrame](#)
- lpspi_clock_polarity_t [cpol](#)
- lpspi_clock_phase_t [cpha](#)
- lpspi_shift_direction_t [direction](#)
- uint32_t [pcsToSckDelayInNanoSec](#)
- uint32_t [lastSckToPcsDelayInNanoSec](#)
- uint32_t [betweenTransferDelayInNanoSec](#)
- lpspi_which_pcs_t [whichPcs](#)
- lpspi_pcs_polarity_config_t [pcsActiveHighOrLow](#)
- lpspi_pin_config_t [pinCfg](#)
- lpspi_data_out_config_t [dataOutConfig](#)
- bool [enableInputDelay](#)

16.18.1 Detailed Description

LPSPi master configuration structure.

16.18.2 Member Data Documentation

16.18.2.1 baudRate

```
uint32_t _lpspi_master_config::baudRate
```

Baud Rate for LPSPi.

16.18.2.2 betweenTransferDelayInNanoSec

```
uint32_t _lpspi_master_config::betweenTransferDelayInNanoSec
```

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

16.18.2.3 bitsPerFrame

```
uint32_t _lpspi_master_config::bitsPerFrame
```

Bits per frame, minimum 8, maximum 4096.

16.18.2.4 cpha

```
lpspi_clock_phase_t _lpspi_master_config::cpha
```

Clock phase.

16.18.2.5 cpol

```
lpspi_clock_polarity_t _lpspi_master_config::cpol
```

Clock polarity.

16.18.2.6 dataOutConfig

```
lpspi_data_out_config_t _lpspi_master_config::dataOutConfig
```

Configures if the output data is tristated between accesses (LPSPi_PCS is negated).

16.18.2.7 direction

`lpspi_shift_direction_t _lpspi_master_config::direction`
 MSB or LSB data shift direction.

16.18.2.8 enableInputDelay

`bool _lpspi_master_config::enableInputDelay`
 Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

16.18.2.9 lastSckToPcsDelayInNanoSec

`uint32_t _lpspi_master_config::lastSckToPcsDelayInNanoSec`
 Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

16.18.2.10 pcsActiveHighOrLow

`lpspi_pcs_polarity_config_t _lpspi_master_config::pcsActiveHighOrLow`
 Desired PCS active high or low

16.18.2.11 pcsToSckDelayInNanoSec

`uint32_t _lpspi_master_config::pcsToSckDelayInNanoSec`
 PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

16.18.2.12 pinCfg

`lpspi_pin_config_t _lpspi_master_config::pinCfg`
 Configures which pins are used for input and output data during single bit transfers.

16.18.2.13 whichPcs

`lpspi_which_pcs_t _lpspi_master_config::whichPcs`
 Desired Peripheral Chip Select (PCS).
 The documentation for this struct was generated from the following file:

- `fsl_lpspi.h`

16.19 _lpspi_master_edma_handle Struct Reference

LPSPi master eDMA transfer handle structure used for transactional API.
`#include <fsl_lpspi_edma.h>`

Public Attributes

- volatile bool [isPcsContinuous](#)
- volatile bool [isByteSwap](#)
- volatile uint8_t [fifoSize](#)
- volatile uint8_t [rxWatermark](#)
- volatile uint8_t [bytesEachWrite](#)
- volatile uint8_t [bytesEachRead](#)
- volatile uint8_t [bytesLastRead](#)
- volatile bool [isThereExtraRxBytes](#)
- uint8_t *volatile [txData](#)
- uint8_t *volatile [rxData](#)
- volatile size_t [txRemainingByteCount](#)

- volatile size_t rxRemainingByteCount
- volatile uint32_t writeRegRemainingTimes
- volatile uint32_t readRegRemainingTimes
- uint32_t totalByteCount
- uint32_t txBuffIfNull
- uint32_t rxBuffIfNull
- uint32_t transmitCommand
- volatile uint8_t state
- uint8_t nbytes
- lpspi_master_edma_transfer_callback_t callback
- void * userData
- edma_handle_t * edmaRxRegToRxDataHandle
- edma_handle_t * edmaTxDataToTxRegHandle
- edma_tcd_t lpspiSoftwareTCD [3]

16.19.1 Detailed Description

LPSPi master eDMA transfer handle structure used for transactional API.

16.19.2 Member Data Documentation

16.19.2.1 bytesEachRead

volatile uint8_t _lpspi_master_edma_handle::bytesEachRead
Bytes for each read RDR.

16.19.2.2 bytesEachWrite

volatile uint8_t _lpspi_master_edma_handle::bytesEachWrite
Bytes for each write TDR.

16.19.2.3 bytesLastRead

volatile uint8_t _lpspi_master_edma_handle::bytesLastRead
Bytes for last read RDR.

16.19.2.4 callback

lpspi_master_edma_transfer_callback_t _lpspi_master_edma_handle::callback
Completion callback.

16.19.2.5 edmaRxRegToRxDataHandle

edma_handle_t* _lpspi_master_edma_handle::edmaRxRegToRxDataHandle
edma_handle_t handle point used for RxReg to RxData buff

16.19.2.6 edmaTxDataToTxRegHandle

edma_handle_t* _lpspi_master_edma_handle::edmaTxDataToTxRegHandle
edma_handle_t handle point used for TxData to TxReg buff

16.19.2.7 fifoSize

volatile uint8_t _lpspi_master_edma_handle::fifoSize
FIFO dataSize.

16.19.2.8 isByteSwap

volatile bool _lpspi_master_edma_handle::isByteSwap
A flag that whether should byte swap.

16.19.2.9 isPcsContinuous

`volatile bool _lpspi_master_edma_handle::isPcsContinuous`
Is PCS continuous in transfer.

16.19.2.10 isThereExtraRxBytes

`volatile bool _lpspi_master_edma_handle::isThereExtraRxBytes`
Is there extra RX byte.

16.19.2.11 lpspiSoftwareTCD

`edma_tcd_t _lpspi_master_edma_handle::lpspiSoftwareTCD[3]`
SoftwareTCD, internal used

16.19.2.12 nbytes

`uint8_t _lpspi_master_edma_handle::nbytes`
eDMA minor byte transfer count initially configured.

16.19.2.13 readRegRemainingTimes

`volatile uint32_t _lpspi_master_edma_handle::readRegRemainingTimes`
Read RDR register remaining times.

16.19.2.14 rxBuffIfNull

`uint32_t _lpspi_master_edma_handle::rxBuffIfNull`
Used if there is not rxData for DMA purpose.

16.19.2.15 rxData

`uint8_t* volatile _lpspi_master_edma_handle::rxData`
Receive buffer.

16.19.2.16 rxRemainingByteCount

`volatile size_t _lpspi_master_edma_handle::rxRemainingByteCount`
Number of bytes remaining to receive.

16.19.2.17 rxWatermark

`volatile uint8_t _lpspi_master_edma_handle::rxWatermark`
Rx watermark.

16.19.2.18 state

`volatile uint8_t _lpspi_master_edma_handle::state`
LPSPi transfer state , `_lpspi_transfer_state`.

16.19.2.19 totalByteCount

`uint32_t _lpspi_master_edma_handle::totalByteCount`
Number of transfer bytes

16.19.2.20 transmitCommand

`uint32_t _lpspi_master_edma_handle::transmitCommand`
Used to write TCR for DMA purpose.

16.19.2.21 txBuffIfNull

```
uint32_t _lpspi_master_edma_handle::txBuffIfNull
```

Used if there is not txData for DMA purpose.

16.19.2.22 txData

```
uint8_t* volatile _lpspi_master_edma_handle::txData
```

Send buffer.

16.19.2.23 txRemainingByteCount

```
volatile size_t _lpspi_master_edma_handle::txRemainingByteCount
```

Number of bytes remaining to send.

16.19.2.24 userData

```
void* _lpspi_master_edma_handle::userData
```

Callback user data.

16.19.2.25 writeRegRemainingTimes

```
volatile uint32_t _lpspi_master_edma_handle::writeRegRemainingTimes
```

Write TDR register remaining times.

The documentation for this struct was generated from the following file:

- fsl_lpspi_edma.h

16.20 _lpspi_master_edma_private_handle Struct Reference

Structure definition for `dsapi_master_edma_private_handle_t`. The structure is private.

```
#include <dsapi.h>
```

Public Attributes

- `LPSPI_Type * base`
- `lpspi_master_edma_handle_t * handle`

16.20.1 Detailed Description

Structure definition for `dsapi_master_edma_private_handle_t`. The structure is private.

16.20.2 Member Data Documentation**16.20.2.1 base**

```
LPSPI_Type* _lpspi_master_edma_private_handle::base
```

LPSPI peripheral base address.

16.20.2.2 handle

```
lpspi_master_edma_handle_t* _lpspi_master_edma_private_handle::handle
```

`lpspi_master_edma_handle_t` handle

The documentation for this struct was generated from the following file:

- cortex-m7/cpu/MIMXRT10xx/include/dsapi.h

16.21 `_lpspi_master_handle` Struct Reference

LPSPi master transfer handle structure used for transactional API.

```
#include <fsl_lpspi.h>
```

Public Attributes

- volatile bool [isPcsContinuous](#)
- volatile bool [writeTcrInIsr](#)
- volatile bool [isByteSwap](#)
- volatile bool [isTxMask](#)
- volatile uint16_t [bytesPerFrame](#)
- volatile uint8_t [fifoSize](#)
- volatile uint8_t [rxWatermark](#)
- volatile uint8_t [bytesEachWrite](#)
- volatile uint8_t [bytesEachRead](#)
- uint8_t *volatile [txData](#)
- uint8_t *volatile [rxData](#)
- volatile size_t [txRemainingByteCount](#)
- volatile size_t [rxRemainingByteCount](#)
- volatile uint32_t [writeRegRemainingTimes](#)
- volatile uint32_t [readRegRemainingTimes](#)
- uint32_t [totalByteCount](#)
- uint32_t [txBuffIfNull](#)
- volatile uint8_t [state](#)
- [lpspi_master_transfer_callback_t](#) [callback](#)
- void * [userData](#)

16.21.1 Detailed Description

LPSPi master transfer handle structure used for transactional API.

16.21.2 Member Data Documentation

16.21.2.1 `bytesEachRead`

```
volatile uint8_t _lpspi_master_handle::bytesEachRead
```

Bytes for each read RDR.

16.21.2.2 `bytesEachWrite`

```
volatile uint8_t _lpspi_master_handle::bytesEachWrite
```

Bytes for each write TDR.

16.21.2.3 `bytesPerFrame`

```
volatile uint16_t _lpspi_master_handle::bytesPerFrame
```

Number of bytes in each frame

16.21.2.4 `callback`

```
lpspi_master_transfer_callback_t _lpspi_master_handle::callback
```

Completion callback.

16.21.2.5 `fifoSize`

```
volatile uint8_t _lpspi_master_handle::fifoSize
```

FIFO dataSize.

16.21.2.6 isByteSwap

```
volatile bool _lpspi_master_handle::isByteSwap
```

A flag that whether should byte swap.

16.21.2.7 isPcsContinuous

```
volatile bool _lpspi_master_handle::isPcsContinuous
```

Is PCS continuous in transfer.

16.21.2.8 isTxMask

```
volatile bool _lpspi_master_handle::isTxMask
```

A flag that whether TCR[TXMSK] is set.

16.21.2.9 readRegRemainingTimes

```
volatile uint32_t _lpspi_master_handle::readRegRemainingTimes
```

Read RDR register remaining times.

16.21.2.10 rxData

```
uint8_t* volatile _lpspi_master_handle::rxData
```

Receive buffer.

16.21.2.11 rxRemainingByteCount

```
volatile size_t _lpspi_master_handle::rxRemainingByteCount
```

Number of bytes remaining to receive.

16.21.2.12 rxWatermark

```
volatile uint8_t _lpspi_master_handle::rxWatermark
```

Rx watermark.

16.21.2.13 state

```
volatile uint8_t _lpspi_master_handle::state
```

LPSPi transfer state , _lpspi_transfer_state.

16.21.2.14 totalByteCount

```
uint32_t _lpspi_master_handle::totalByteCount
```

Number of transfer bytes

16.21.2.15 txBuffIfNull

```
uint32_t _lpspi_master_handle::txBuffIfNull
```

Used if the txData is NULL.

16.21.2.16 txData

```
uint8_t* volatile _lpspi_master_handle::txData
```

Send buffer.

16.21.2.17 txRemainingByteCount

```
volatile size_t _lpspi_master_handle::txRemainingByteCount
```

Number of bytes remaining to send.

16.21.2.18 userData

`void* _lpspi_master_handle::userData`
Callback user data.

16.21.2.19 writeRegRemainingTimes

`volatile uint32_t _lpspi_master_handle::writeRegRemainingTimes`
Write TDR register remaining times.

16.21.2.20 writeTcrInIsr

`volatile bool _lpspi_master_handle::writeTcrInIsr`
A flag that whether should write TCR in ISR.
The documentation for this struct was generated from the following file:

- `fsl_lpspi.h`

16.22 _lpspi_slave_config Struct Reference

LPSPi slave configuration structure.
`#include <fsl_lpspi.h>`

Public Attributes

- `uint32_t bitsPerFrame`
- `lpspi_clock_polarity_t cpol`
- `lpspi_clock_phase_t cpha`
- `lpspi_shift_direction_t direction`
- `lpspi_which_pcs_t whichPcs`
- `lpspi_pcs_polarity_config_t pcsActiveHighOrLow`
- `lpspi_pin_config_t pinCfg`
- `lpspi_data_out_config_t dataOutConfig`

16.22.1 Detailed Description

LPSPi slave configuration structure.

16.22.2 Member Data Documentation

16.22.2.1 bitsPerFrame

`uint32_t _lpspi_slave_config::bitsPerFrame`
Bits per frame, minimum 8, maximum 4096.

16.22.2.2 cpha

`lpspi_clock_phase_t _lpspi_slave_config::cpha`
Clock phase.

16.22.2.3 cpol

`lpspi_clock_polarity_t _lpspi_slave_config::cpol`
Clock polarity.

16.22.2.4 dataOutConfig

`lpspi_data_out_config_t _lpspi_slave_config::dataOutConfig`
Configures if the output data is tristated between accesses (LPSPi_PCS is negated).

16.22.2.5 direction

`lpspi_shift_direction_t _lpspi_slave_config::direction`
MSB or LSB data shift direction.

16.22.2.6 pcsActiveHighOrLow

`lpspi_pcs_polarity_config_t _lpspi_slave_config::pcsActiveHighOrLow`
Desired PCS active high or low

16.22.2.7 pinCfg

`lpspi_pin_config_t _lpspi_slave_config::pinCfg`
Configures which pins are used for input and output data during single bit transfers.

16.22.2.8 whichPcs

`lpspi_which_pcs_t _lpspi_slave_config::whichPcs`
Desired Peripheral Chip Select (pcs)
The documentation for this struct was generated from the following file:

- `fsl_lpspi.h`

16.23 _lpspi_slave_edma_handle Struct Reference

LPSPi slave eDMA transfer handle structure used for transactional API.
`#include <fsl_lpspi_edma.h>`

Public Attributes

- volatile bool [isByteSwap](#)
- volatile uint8_t [fifoSize](#)
- volatile uint8_t [rxWatermark](#)
- volatile uint8_t [bytesEachWrite](#)
- volatile uint8_t [bytesEachRead](#)
- volatile uint8_t [bytesLastRead](#)
- volatile bool [isThereExtraRxBytes](#)
- uint8_t [nbytes](#)
- uint8_t *volatile [txData](#)
- uint8_t *volatile [rxData](#)
- volatile size_t [txRemainingByteCount](#)
- volatile size_t [rxRemainingByteCount](#)
- volatile uint32_t [writeRegRemainingTimes](#)
- volatile uint32_t [readRegRemainingTimes](#)
- uint32_t [totalByteCount](#)
- uint32_t [txBuffIfNull](#)
- uint32_t [rxBuffIfNull](#)
- volatile uint8_t [state](#)
- uint32_t [errorCount](#)
- `lpspi_slave_edma_transfer_callback_t` [callback](#)
- void * [userData](#)
- `edma_handle_t` * [edmaRxRegToRxDataHandle](#)
- `edma_handle_t` * [edmaTxDataToTxRegHandle](#)
- `edma_tcd_t` [lpspiSoftwareTCD](#) [2]

16.23.1 Detailed Description

LPSPi slave eDMA transfer handle structure used for transactional API.

16.23.2 Member Data Documentation

16.23.2.1 bytesEachRead

`volatile uint8_t _lpspi_slave_edma_handle::bytesEachRead`
Bytes for each read RDR.

16.23.2.2 bytesEachWrite

`volatile uint8_t _lpspi_slave_edma_handle::bytesEachWrite`
Bytes for each write TDR.

16.23.2.3 bytesLastRead

`volatile uint8_t _lpspi_slave_edma_handle::bytesLastRead`
Bytes for last read RDR.

16.23.2.4 callback

`lpspi_slave_edma_transfer_callback_t _lpspi_slave_edma_handle::callback`
Completion callback.

16.23.2.5 edmaRxRegToRxDataHandle

`edma_handle_t* _lpspi_slave_edma_handle::edmaRxRegToRxDataHandle`
`edma_handle_t` handle point used for RxReg to RxData buff

16.23.2.6 edmaTxDataToTxRegHandle

`edma_handle_t* _lpspi_slave_edma_handle::edmaTxDataToTxRegHandle`
`edma_handle_t` handle point used for TxData to TxReg

16.23.2.7 errorCount

`uint32_t _lpspi_slave_edma_handle::errorCount`
Error count for slave transfer.

16.23.2.8 fifoSize

`volatile uint8_t _lpspi_slave_edma_handle::fifoSize`
FIFO dataSize.

16.23.2.9 isByteSwap

`volatile bool _lpspi_slave_edma_handle::isByteSwap`
A flag that whether should byte swap.

16.23.2.10 isThereExtraRxBytes

`volatile bool _lpspi_slave_edma_handle::isThereExtraRxBytes`
Is there extra RX byte.

16.23.2.11 lpspiSoftwareTCD

`edma_tcd_t _lpspi_slave_edma_handle::lpspiSoftwareTCD[2]`
SoftwareTCD, internal used

16.23.2.12 nbytes

`uint8_t _lpspi_slave_edma_handle::nbytes`
eDMA minor byte transfer count initially configured.

16.23.2.13 readRegRemainingTimes

`volatile uint32_t _lpspi_slave_edma_handle::readRegRemainingTimes`
Read RDR register remaining times.

16.23.2.14 rxBuffIfNull

`uint32_t _lpspi_slave_edma_handle::rxBuffIfNull`
Used if there is not rxData for DMA purpose.

16.23.2.15 rxData

`uint8_t* volatile _lpspi_slave_edma_handle::rxData`
Receive buffer.

16.23.2.16 rxRemainingByteCount

`volatile size_t _lpspi_slave_edma_handle::rxRemainingByteCount`
Number of bytes remaining to receive.

16.23.2.17 rxWatermark

`volatile uint8_t _lpspi_slave_edma_handle::rxWatermark`
Rx watermark.

16.23.2.18 state

`volatile uint8_t _lpspi_slave_edma_handle::state`
LPSPi transfer state.

16.23.2.19 totalByteCount

`uint32_t _lpspi_slave_edma_handle::totalByteCount`
Number of transfer bytes

16.23.2.20 txBuffIfNull

`uint32_t _lpspi_slave_edma_handle::txBuffIfNull`
Used if there is not txData for DMA purpose.

16.23.2.21 txData

`uint8_t* volatile _lpspi_slave_edma_handle::txData`
Send buffer.

16.23.2.22 txRemainingByteCount

`volatile size_t _lpspi_slave_edma_handle::txRemainingByteCount`
Number of bytes remaining to send.

16.23.2.23 userData

`void* _lpspi_slave_edma_handle::userData`
Callback user data.

16.23.2.24 writeRegRemainingTimes

`volatile uint32_t _lpspi_slave_edma_handle::writeRegRemainingTimes`
Write TDR register remaining times.

The documentation for this struct was generated from the following file:

- `fsl_lpspi_edma.h`

16.24 _lpspi_slave_handle Struct Reference

LPSPi slave transfer handle structure used for transactional API.

```
#include <fsl_lpspi.h>
```

Public Attributes

- volatile bool [isByteSwap](#)
- volatile uint8_t [fifoSize](#)
- volatile uint8_t [rxWatermark](#)
- volatile uint8_t [bytesEachWrite](#)
- volatile uint8_t [bytesEachRead](#)
- uint8_t *volatile [txData](#)
- uint8_t *volatile [rxData](#)
- volatile size_t [txRemainingByteCount](#)
- volatile size_t [rxRemainingByteCount](#)
- volatile uint32_t [writeRegRemainingTimes](#)
- volatile uint32_t [readRegRemainingTimes](#)
- uint32_t [totalByteCount](#)
- volatile uint8_t [state](#)
- volatile uint32_t [errorCount](#)
- lpspi_slave_transfer_callback_t [callback](#)
- void * [userData](#)

16.24.1 Detailed Description

LPSPi slave transfer handle structure used for transactional API.

16.24.2 Member Data Documentation

16.24.2.1 bytesEachRead

```
volatile uint8_t _lpspi_slave_handle::bytesEachRead
```

Bytes for each read RDR.

16.24.2.2 bytesEachWrite

```
volatile uint8_t _lpspi_slave_handle::bytesEachWrite
```

Bytes for each write TDR.

16.24.2.3 callback

```
lpspi_slave_transfer_callback_t _lpspi_slave_handle::callback
```

Completion callback.

16.24.2.4 errorCount

```
volatile uint32_t _lpspi_slave_handle::errorCount
```

Error count for slave transfer.

16.24.2.5 fifoSize

```
volatile uint8_t _lpspi_slave_handle::fifoSize
```

FIFO dataSize.

16.24.2.6 isByteSwap

```
volatile bool _lpspi_slave_handle::isByteSwap
```

A flag that whether should byte swap.

16.24.2.7 readRegRemainingTimes

`volatile uint32_t _lpspi_slave_handle::readRegRemainingTimes`
Read RDR register remaining times.

16.24.2.8 rxData

`uint8_t* volatile _lpspi_slave_handle::rxData`
Receive buffer.

16.24.2.9 rxRemainingByteCount

`volatile size_t _lpspi_slave_handle::rxRemainingByteCount`
Number of bytes remaining to receive.

16.24.2.10 rxWatermark

`volatile uint8_t _lpspi_slave_handle::rxWatermark`
Rx watermark.

16.24.2.11 state

`volatile uint8_t _lpspi_slave_handle::state`
LPSPI transfer state , `_lpspi_transfer_state`.

16.24.2.12 totalByteCount

`uint32_t _lpspi_slave_handle::totalByteCount`
Number of transfer bytes

16.24.2.13 txData

`uint8_t* volatile _lpspi_slave_handle::txData`
Send buffer.

16.24.2.14 txRemainingByteCount

`volatile size_t _lpspi_slave_handle::txRemainingByteCount`
Number of bytes remaining to send.

16.24.2.15 userData

`void* _lpspi_slave_handle::userData`
Callback user data.

16.24.2.16 writeRegRemainingTimes

`volatile uint32_t _lpspi_slave_handle::writeRegRemainingTimes`
Write TDR register remaining times.
The documentation for this struct was generated from the following file:

- `fsl_lpspi.h`

16.25 _lpspi_transfer Struct Reference

LPSPI master/slave transfer structure.
`#include <fsl_lpspi.h>`

Public Attributes

- uint8_t * [txData](#)
- uint8_t * [rxData](#)
- volatile size_t [dataSize](#)
- uint32_t [configFlags](#)

16.25.1 Detailed Description

LPSPi master/slave transfer structure.

16.25.2 Member Data Documentation

16.25.2.1 configFlags

uint32_t _lpspi_transfer::configFlags

Transfer transfer configuration flags. Set from _lpspi_transfer_config_flag_for_master if the transfer is used for master or _lpspi_transfer_config_flag_for_slave enumeration if the transfer is used for slave.

16.25.2.2 dataSize

volatile size_t _lpspi_transfer::dataSize

Transfer bytes.

16.25.2.3 rxData

uint8_t* _lpspi_transfer::rxData

Receive buffer.

16.25.2.4 txData

uint8_t* _lpspi_transfer::txData

Send buffer.

The documentation for this struct was generated from the following file:

- fsl_lpspi.h

16.26 _lut_sequence Struct Reference

FlexSPI LUT Sequence structure.

```
#include <evkmimxrt1060_flexspi_nor_config.h>
```

Public Attributes

- uint8_t **seqNum**
Sequence Number, valid number: 1-16.
- uint8_t **seqId**
Sequence Index, valid number: 0-15.

16.26.1 Detailed Description

FlexSPI LUT Sequence structure.

The documentation for this struct was generated from the following file:

- evkmimxrt1060_flexspi_nor_config.h

16.27 _PinVector Class Reference

GPIO Pin Vector Base Class.

#include <cpu_pins.h>

Inherited by [PinVector< 4 >](#), [PinVector< 4 >](#), [PinVector< n >](#), and [PinVector< n >](#).

Public Member Functions

- `uint32_t operator= (uint32_t val)`
Assign a value to the [_PinVector](#) Bus.
- `PinIO operator[] (int idx)`
Access the [PinIO](#) for a specific bit position in the [_PinVector](#).
- `void config (uint32_t idx, PinIO cfg)`
Set the [PinIO](#) that will be used for a given bit position in the [_PinVector](#).
- `void config (PinIO *pinCfgs, uint32_t count)`
Configure the [_PinVector](#) based on an array of PinIOs. The index of the [PinIO](#) in the configuration array will determine the bit position within the [_PinVector](#) that that [PinIO](#) represents.
- `operator uint32_t () const`
Read the line state of the [_PinVector](#) bus.

16.27.1 Detailed Description

GPIO Pin Vector Base Class.

This class is for building semi-parallel buses using GPIO pins. It facilitates driving a numeric value across a parallel bus of GPIO pins. It should not be confused with a true parallel bus, as each individual pin in the bus is modified sequentially/independently during a bus assignment.

Parameters

<code>_Pin::len</code>	Number of pins managed by the _PinVector .

16.27.2 Member Function Documentation

16.27.2.1 config() [1/2]

```
void _PinVector::config (
    PinIO * pinCfgs,
    uint32_t count)
```

Configure the [_PinVector](#) based on an array of PinIOs. The index of the [PinIO](#) in the configuration array will determine the bit position within the [_PinVector](#) that that [PinIO](#) represents.

Parameters

<code>pinCfgs</code>	The configuration to be used.
<code>count</code>	The number of PinIOs in the configuration array to be used.

16.27.2.2 config() [2/2]

```
void _PinVector::config (
    uint32_t idx,
    PinIO cfg)
```

Set the [PinIO](#) that will be used for a given bit position in the [_PinVector](#).

Parameters

<i>idx</i>	The bit position to be configured.
<i>cfg</i>	The PinIO that bit position will represent.

16.27.2.3 operator uint32_t()

`_PinVector::operator uint32_t () const`
 Read the line state of the [_PinVector](#) bus.

Returns

The current bus value as read from the line state.

16.27.2.4 operator=()

`uint32_t _PinVector::operator= (`
 `uint32_t val)`
 Assign a value to the [_PinVector](#) Bus.

Parameters

<i>val</i>	The value to assign to the bus.

Returns

The value driven on the bus.

16.27.2.5 operator[]()

`PinIO _PinVector::operator[] (`
 `int idx)`
 Access the [PinIO](#) for a specific bit position in the [_PinVector](#).

Parameters

<i>idx</i>	The bit index to access in the _PinVector .

Returns

The [PinIO](#) for the given bit position.

The documentation for this class was generated from the following files:

- cortex-m7/cpu/SAME70/include/cpu_pins.h
- cortex-m7/cpu/MIMXRT10xx/include/cpu_pins.h

16.28 _pit_config Struct Reference

PIT configuration structure.
`#include <fsl_pit.h>`

Public Attributes

- bool [enableRunInDebug](#)

16.28.1 Detailed Description

PIT configuration structure.

This structure holds the configuration settings for the PIT peripheral. To initialize this structure to reasonable defaults, call the `PIT_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The configuration structure can be made constant so it resides in flash.

16.28.2 Member Data Documentation

16.28.2.1 enableRunInDebug

```
bool _pit_config::enableRunInDebug
```

true: Timers run in debug mode; false: Timers stop in debug mode

The documentation for this struct was generated from the following file:

- `fsl_pit.h`

16.29 _qtmr_config Struct Reference

Quad Timer config structure.

```
#include <fsl_qtmr.h>
```

Public Attributes

- [qtmr_primary_count_source_t](#) primarySource
- [qtmr_input_source_t](#) secondarySource
- bool [enableMasterMode](#)
- bool [enableExternalForce](#)
- [uint8_t](#) [faultFilterCount](#)
- [uint8_t](#) [faultFilterPeriod](#)
- [qtmr_debug_action_t](#) debugMode

16.29.1 Detailed Description

Quad Timer config structure.

This structure holds the configuration settings for the Quad Timer peripheral. To initialize this structure to reasonable defaults, call the [QTMR_GetDefaultConfig\(\)](#) function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

16.29.2 Member Data Documentation

16.29.2.1 debugMode

```
qtmr_debug_action_t _qtmr_config::debugMode
```

Operation in Debug mode

16.29.2.2 enableExternalForce

```
bool _qtmr_config::enableExternalForce
```

true: Compare from another counter force state of OFLAG signal false: OFLAG controlled by local counter

16.29.2.3 enableMasterMode

```
bool _qtmr_config::enableMasterMode
```

true: Broadcast compare function output to other counters; false no broadcast

16.29.2.4 faultFilterCount

```
uint8_t _qtmr_config::faultFilterCount
```

Fault filter count

16.29.2.5 faultFilterPeriod

`uint8_t _qtmr_config::faultFilterPeriod`
Fault filter period;value of 0 will bypass the filter

16.29.2.6 primarySource

`qtmr_primary_count_source_t _qtmr_config::primarySource`
Specify the primary count source

16.29.2.7 secondarySource

`qtmr_input_source_t _qtmr_config::secondarySource`
Specify the secondary count source
The documentation for this struct was generated from the following file:

- [fsl_qtmr.h](#)

16.30 _tempmon_config Struct Reference

TEMPMON temperature structure.
`#include <fsl_tempmon.h>`

Public Attributes

- `uint16_t` [frequency](#)
- `int8_t` [highAlarmTemp](#)
- `int8_t` [panicAlarmTemp](#)
- `int8_t` [lowAlarmTemp](#)

16.30.1 Detailed Description

TEMPMON temperature structure.

16.30.2 Member Data Documentation

16.30.2.1 frequency

`uint16_t _tempmon_config::frequency`
The temperature measure frequency.

16.30.2.2 highAlarmTemp

`int8_t _tempmon_config::highAlarmTemp`
The high alarm temperature.

16.30.2.3 lowAlarmTemp

`int8_t _tempmon_config::lowAlarmTemp`
The low alarm temperature.

16.30.2.4 panicAlarmTemp

`int8_t _tempmon_config::panicAlarmTemp`
The panic alarm temperature.
The documentation for this struct was generated from the following file:

- [fsl_tempmon.h](#)

16.31 AcmeServletObject Class Reference

Main ACME Client Class.

#include <acmeRFC8555Servlet.h>

Inherits [DiagItemClass](#), and [WebClientServlet](#).

Inherited by [BuyPassAcmeServletObject](#), and [LetsEncryptAcmeServletObject](#).

Public Member Functions

- void [Delete_Everything_Restart](#) ()
Re-request the ACME certificate.
- void [SetDiag](#) (bool v)
Turn on diganostic printf messages to the console.
- [NBString](#) [GetGlobalStateString](#) ()
Returns the detailed state of the ACME process.
- [NBString](#) [GetStateString](#) ()
Returns the specific state condition of the acmeservlet.
- const char * [GetStateCC](#) ()
Returns the specific state condition of the acmeservlet.
- [AcmeServletObject](#) (const char *pUrlDir)
ACME Client Constructor.

Friends

- [CertGenData](#) * [GetDataForCertGen](#) ()
A function to be overridden to pass certificate details to the certificate generator.

16.31.1 Detailed Description

Main ACME Client Class.

16.31.2 Constructor & Destructor Documentation

16.31.2.1 AcmeServletObject()

```
AcmeServletObject::AcmeServletObject (
    const char * pUrlDir) [inline]
```

ACME Client Constructor.

Parameters

<i>pUrlDir</i>	URL for the ACME API directory endpoint (required, i.e. https://acme.example.com/dir)

16.31.3 Member Function Documentation

16.31.3.1 Delete_Everything_Restart()

```
void AcmeServletObject::Delete_Everything_Restart ()
```

Re-request the ACME certificate.

Once the system has loaded valid keys and certificate It sleeps and does not do much. If you want to change the names, or want to re-run the ACME protocol for debugging/diagnostic purposes, this function erases all keys, the certificate, and restarts the certificate issuing process.

16.31.3.2 GetStateCC()

```
const char * AcmeServletObject::GetStateCC ()
```

Returns the specific state condition of the acmeservlet.

16.31.4 Friends And Related Symbol Documentation

16.31.4.1 GetDataForCertGen

```
CertGenData * GetDataForCertGen () [friend]
```

A function to be overridden to pass certificate details to the certificate generator.

The library provides this function as a weak reference. The values returned by this weak reference are likely wrong for any real application and the user should provide their own function. Applicable for both self signed and ACME certificates generated onboard.

Returns

pointer to a [CertGenData](#) struct

The documentation for this class was generated from the following file:

- acmeRFC8555Servlet.h

16.32 aes_context Struct Reference

AES context structure.

```
#include <aes.h>
```

Public Attributes

- unsigned long [erk](#) [64]
- unsigned long [drk](#) [64]
- int [nr](#)

16.32.1 Detailed Description

AES context structure.

16.32.2 Member Data Documentation

16.32.2.1 drk

```
unsigned long aes_context::drk[64]
```

decryption round keys

16.32.2.2 erk

```
unsigned long aes_context::erk[64]
```

encryption round keys

16.32.2.3 nr

```
int aes_context::nr
```

number of rounds

The documentation for this struct was generated from the following file:

- aes.h

16.33 ARM_MPU_Region_t Struct Reference

```
#include <mpu_armv7.h>
```

Public Attributes

- uint32_t **RBAR**
The region base address register value (RBAR)
- uint32_t **RASR**
The region attribute and size register value (RASR)

16.33.1 Detailed Description

Struct for a single MPU Region

The documentation for this struct was generated from the following file:

- mpu_armv7.h

16.34 ARP Struct Reference

Address Resolution Protocol packet structure.

```
#include <arp.h>
```

Public Attributes

- beuint16_t [hard_Type](#)
Hardware type (1 for Ethernet)
- beuint16_t [prot_Type](#)
Protocol type (0x0800 for IPv4)
- uint8_t [hard_size](#)
Hardware address length (6 for MAC)
- uint8_t [prot_size](#)
Protocol address length (4 for IPv4)
- beuint16_t [op_code](#)
Operation: 1=Request, 2=Reply.
- [MACADR sender_phy](#)
Sender MAC address.
- [IPADDR4 sender_ip](#)
Sender IP address.
- [MACADR target_phy](#)
Target MAC address.
- [IPADDR4 target_ip](#)
Target IP address.

16.34.1 Detailed Description

Address Resolution Protocol packet structure.

Defines the structure of an [ARP](#) packet as transmitted on Ethernet networks. This structure matches the [ARP](#) packet format defined in RFC 826.

[ARP](#) packets are used to resolve IP addresses to MAC addresses on local networks. The packet contains both the sender's and target's IP and MAC addresses, along with operation type.

The documentation for this struct was generated from the following file:

- arp.h

16.35 BBI2C Class Reference

[I2C](#) Peripheral Class.

```
#include <bb_i2c.h>
```

Public Types

- enum [Result_t](#) {
[I2C_RES_ACK](#) , [I2C_RES_NACK](#) , [I2C_RES_ARB_LST](#) , [I2C_RES_BUSY](#) ,
[I2C_RES_ARG](#) , [I2C_RES_IN_PROGRESS](#) , [I2C_RES_ERR](#) }

Public Member Functions

- [BBI2C](#) ([PinIO](#) scl, [PinIO](#) sda)
Constructor for the [I2C](#) peripheral module.
- void [setup](#) (uint32_t maxBusSpeed)
Setup the [I2C](#) peripheral interface.
- void [resetBus](#) ()
Reset the [I2C](#) bus.
- void [setNumAddressBytes](#) (uint8_t numAddressBytes=1)
Specify the register address size for a read or write transaction. A number of bytes, from 0 to 3, can be sent to specify an address up to 24-bit (3 bytes): 0 = none, 1 = 8-bits, 2 = 16-bits, 3 = 24-bits.
- virtual [Result_t](#) [writeReg8](#) (uint8_t devAddr, uint32_t reg, uint8_t data)
Write an 8-bit value to a [I2C](#) slave device register.
- virtual [Result_t](#) [readReg8](#) (uint8_t devAddr, uint32_t reg, uint8_t &data)
Read an 8-bit value form an [I2C](#) slave device register.
- virtual [Result_t](#) [writeRegN](#) (uint8_t devAddr, uint32_t reg, uint8_t *buf, uint32_t blen)
Write a number of 8-bit values to an [I2C](#) slave to the specified register address.
- virtual [Result_t](#) [readRegN](#) (uint8_t devAddr, uint32_t reg, uint8_t *buf, uint32_t blen)
Read a number of 8-bit values from an [I2C](#) slave at the specified register address.
- void [SetMaxBusDelay](#) (uint16_t milliseconds)
Set the maximum duration that the bus is allowed to be in use without being clocked before being forcibly reset.

Friends

- class [WireIntf](#)

16.35.1 Detailed Description

[I2C](#) Peripheral Class.

For [I2C](#) communication, you can choose the [WireIntf](#) class, or the [I2C](#) class. The [WireIntf](#) class is simpler to use, while the [I2C](#) class provides more low level control.

Note that the system automatically instantiates [I2C](#) and [WireIntf](#) objects for each of the [I2C](#) peripheral modules. The [WireIntf](#) objects can be accessed with: Wire, Wire1 and Wire2. The [I2C](#) objects can be accessed with [I2C](#)[0], [I2C](#)[1] and [I2C](#)[2].

16.35.2 Member Enumeration Documentation

16.35.2.1 Result_t

enum [BBI2C::Result_t](#)

Enumerator

I2C_RES_ACK	Acknowledged.
I2C_RES_NACK	Not acknowledged.
I2C_RES_ARB_LST	Arbitration listening.
I2C_RES_BUSY	Bus is busy.
I2C_RES_ARG	Bad argument.
I2C_RES_IN_PROGRESS	Another transaction is currently in progress.
I2C_RES_ERR	Unknown Error.

16.35.3 Constructor & Destructor Documentation

16.35.3.1 BBI2C()

```
BBI2C::BBI2C (
    PinIO scl,
    PinIO sda)
```

Constructor for the [I2C](#) peripheral module.

Parameters

<i>scl</i>	
<i>sda</i>	

16.35.4 Member Function Documentation

16.35.4.1 readReg8()

```
virtual Result_t BBI2C::readReg8 (
    uint8_t devAddr,
    uint32_t reg,
    uint8_t & data) [virtual]
```

Read an 8-bit value form an [I2C](#) slave device register.

Parameters

<i>devAddr</i>	Address of I2C device
<i>reg</i>	Register address to read
<i>data</i>	Reference to variable in which to store the register data

Returns

[Result_t](#)

16.35.4.2 readRegN()

```
virtual Result_t BBI2C::readRegN (
    uint8_t devAddr,
    uint32_t reg,
    uint8_t * buf,
    uint32_t blen) [virtual]
```

Read a number of 8-bit values from an [I2C](#) slave at the specified register address.

Parameters

<i>devAddr</i>	Address of I2C device
<i>reg</i>	Register address to read
<i>*buf</i>	Pointer to buffer to store received data
<i>blen</i>	Number of bytes to read

Returns

[Result_t](#)

16.35.4.3 SetMaxBusDelay()

```
void BBI2C::SetMaxBusDelay (
    uint16_t milliseconds)
```

Set the maximum duration that the bus is allowed to be in use without being clocked before being forcibly reset.

Parameters

<i>milliseconds</i>	Maximum number of milliseconds before triggering a bus reset

16.35.4.4 setNumAddressBytes()

```
void BBI2C::setNumAddressBytes (
    uint8_t numAddressBytes = 1) [inline]
```

Specify the register address size for a read or write transaction. A number of bytes, from 0 to 3, can be sent to specify an address up to 24-bit (3 bytes): 0 = none, 1 = 8-bits, 2 = 16-bits, 3 = 24-bits.

If you are using a single [I2C](#) device, or all devices use the same address size, this function only needs to be called once. If you have multiple [I2C](#) devices with different address/register bit sizes, then this function should be called before each read/write operation for each different device.

Parameters

<i>numAddressBytes</i>	The number of address bytes to send: 0 - 3. The default value is 1 byte.

16.35.4.5 setup()

```
void BBI2C::setup (
    uint32_t maxBusSpeed)
```

Setup the [I2C](#) peripheral interface.

Parameters

<i>maxBusSpeed</i>	Target bus speed. If the specified speed cannot be achieved, a lower bus speed may be enabled.

16.35.4.6 writeReg8()

```
virtual Result_t BBI2C::writeReg8 (
    uint8_t devAddr,
    uint32_t reg,
    uint8_t data) [virtual]
```

Write an 8-bit value to a [I2C](#) slave device register.

Parameters

<i>devAddr</i>	Address of I2C device
<i>reg</i>	Register address to write
<i>data</i>	Data to write to register

Returns

[Result_t](#)

16.35.4.7 writeRegN()

```
virtual Result_t BBI2C::writeRegN (
    uint8_t devAddr,
    uint32_t reg,
    uint8_t * buf,
    uint32_t blen) [virtual]
```

Write a number of 8-bit values to an [I2C](#) slave to the specified register address.

Executing this function will send the following: a start bit, the slave device address and read/write bit, the 8-bit data bytes, and finally a stop bit. Note that the address is not incremented with each 8-bit data value.

Parameters

<i>devAddr</i>	Address of I2C device
<i>reg</i>	Register address to write
<i>*buf</i>	Pointer to buffer containing data to write
<i>blen</i>	Number of bytes to write

Returns

[Result_t](#)

The documentation for this class was generated from the following file:

- [bb_i2c.h](#)

16.36 DHCP::BlockAllocator Class Reference

Basic allocator that handles multiple leases in a contiguous IP block.

```
#include <dhcpd.h>
```

Inherits [DHCP::LeaseAllocator](#).

Inherited by [DHCP::MacPrefixAllocator](#), and [MyModAlloc](#).

16.36.1 Detailed Description

Basic allocator that handles multiple leases in a contiguous IP block.

The documentation for this class was generated from the following file:

- [dhcpd.h](#)

16.37 buffer_list Class Reference

Doubly-linked list management for buffers.

```
#include <buffers.h>
```

Public Member Functions

- void [InsertHead](#) ([PoolPtr](#) buffer)
Insert buffer at the head of the list.
- void [InsertTail](#) ([PoolPtr](#) buffer)
Insert buffer at the tail of the list.
- void [InsertBefore](#) ([PoolPtr](#) buf2insert, [PoolPtr](#) b4buffer)
Insert buffer before specified buffer in list.
- void [InsertAfter](#) ([PoolPtr](#) buf2insert, [PoolPtr](#) after_buffer)
Insert buffer after specified buffer in list.
- void [Remove](#) ([PoolPtr](#) buffer)
Remove specified buffer from the list.
- [PoolPtr](#) [RemoveHead](#) ()
Remove and return the head buffer.
- [PoolPtr](#) [RemoveTail](#) ()
Remove and return the tail buffer.
- [uint16_t](#) [GetCount](#) ()
Get current number of buffers in the list.

Public Attributes

- [PoolPtr](#) **m_Head**
First buffer in the list.
- [PoolPtr](#) **m_Tail**
Last buffer in the list.
- `uint16_t` **m_wElements**
Current number of elements in list.

16.37.1 Detailed Description

Doubly-linked list management for buffers.

Provides efficient insertion, removal, and traversal operations for managing lists of [pool_buffer](#) structures. Used internally by the buffer system and FIFO storage classes.

16.37.2 Member Function Documentation**16.37.2.1 GetCount()**

```
uint16_t buffer_list::GetCount () [inline]
```

Get current number of buffers in the list.

Returns

Number of buffers

16.37.2.2 InsertAfter()

```
void buffer_list::InsertAfter (
    PoolPtr buf2insert,
    PoolPtr after_buffer)
```

Insert buffer after specified buffer in list.

Parameters

<i>buf2insert</i>	Buffer to insert
<i>after_buffer</i>	Buffer to insert after

16.37.2.3 InsertBefore()

```
void buffer_list::InsertBefore (
    PoolPtr buf2insert,
    PoolPtr b4buffer)
```

Insert buffer before specified buffer in list.

Parameters

<i>buf2insert</i>	Buffer to insert
<i>b4buffer</i>	Buffer to insert before

16.37.2.4 InsertHead()

```
void buffer_list::InsertHead (
    PoolPtr buffer)
```

Insert buffer at the head of the list.

Parameters

<i>buffer</i>	Buffer to insert

16.37.2.5 InsertTail()

```
void buffer_list::InsertTail (  
    PoolPtr buffer)
```

Insert buffer at the tail of the list.

Parameters

<i>buffer</i>	Buffer to insert

16.37.2.6 Remove()

```
void buffer_list::Remove (  
    PoolPtr buffer)
```

Remove specified buffer from the list.

Parameters

<i>buffer</i>	Buffer to remove

16.37.2.7 RemoveHead()

```
PoolPtr buffer_list::RemoveHead ()
```

Remove and return the head buffer.

Returns

Head buffer, or NULL if list is empty

16.37.2.8 RemoveTail()

```
PoolPtr buffer_list::RemoveTail ()
```

Remove and return the tail buffer.

Returns

Tail buffer, or NULL if list is empty

The documentation for this class was generated from the following file:

- buffers.h

16.38 buffer_object Class Reference

Base class for web client response buffers.

```
#include <web_buffers.h>
```

Inherited by [ParsedJsonDataSet](#), [SimpleBufferObject](#), and [StringBufferObject](#).

16.38.1 Detailed Description

Base class for web client response buffers.

See also

[ParsedJsonDataSet](#), [SimpleBufferObject](#), [StringBufferObject](#)

The documentation for this class was generated from the following file:

- web_buffers.h

16.39 BuyPassAcmeServletObject Class Reference

ACME Client for BuyPass.

#include <acmeRFC8555Servlet.h>

Inherits [AcmeServletObject](#).

Public Member Functions

- **BuyPassAcmeServletObject** ()
Enable the BuyPass ACME client.

Public Member Functions inherited from [AcmeServletObject](#)

- void [Delete_Everything_Restart](#) ()
Re-request the ACME certificate.
- void **SetDiag** (bool v)
Turn on diagnostic printf messages to the console.
- [NBString](#) **GetGlobalStateString** ()
Returns the detailed state of the ACME process.
- [NBString](#) **GetStateString** ()
Returns the specific state condition of the acmeservlet.
- const char * [GetStateCC](#) ()
Returns the specific state condition of the acmeservlet.
- [AcmeServletObject](#) (const char *pUrlDir)
ACME Client Constructor.

16.39.1 Detailed Description

ACME Client for BuyPass.

The documentation for this class was generated from the following file:

- acmeRFC8555Servlet.h

16.40 CallbackFunctionPageHandler Class Reference

Implements the [HtmlPageHandler](#) class as a function pointer callback for GET requests.

#include <http.h>

Inherits [HtmlPageHandler](#).

Public Member Functions

- virtual int [ProcessRaw](#) (int sock, [HTTP_Request](#) &pdt)
Pure virtual method that must be implemented to process HTTP requests.
- [CallbackFunctionPageHandler](#) (const char *pUrl, [http_gethandlerfunc](#) *pFunction, [HTTP_RequestTypes](#) reqType=[tGet](#), int accessGroup=0, bool beforeFiles=false)
Constructor for HTTP GET callback function.
- [CallbackFunctionPageHandler](#) (const char *pUrl, [http_gethandlerfunc](#) *pFunction, [http_matchhandlerfunc](#) *pMatchFunction, [HTTP_RequestTypes](#) reqType=[tGet](#), int accessGroup=0, bool beforeFiles=false)
Constructor for HTTP GET callback function, includes option for function to match the requested name.
- virtual bool **Match** ([HTTP_Request](#) &req)
Determines if this handler matches the incoming request.

Public Member Functions inherited from [HtmlPageHandler](#)

- [HtmlPageHandler](#) (const char *url, [HTTP_RequestTypes](#) rt=tGet, int accessGroup=0, bool BeforeFiles=false)
Constructs and registers an HTTP request handler for a specific URL pattern and method.
- [~HtmlPageHandler](#) ()
Destructor that automatically unregisters the handler from the global chain.
- void [MoveHandlerToLast](#) ()
Moves this handler to the end of the handler chain, making it function as a 404 handler.
- int [GetGroup](#) ()
Returns the access control group identifier for this handler.

Additional Inherited Members

Static Public Member Functions inherited from [HtmlPageHandler](#)

- static [HtmlPageHandler](#) * [FindHandler](#) ([HTTP_Request](#) &req, bool bBeforeFiles)
Locates the appropriate handler for an incoming HTTP request.

Protected Member Functions inherited from [HtmlPageHandler](#)

- void [InsertSort](#) ([HtmlPageHandler](#) *&ph)
Inserts handler into sorted linked list based on priority.
- int [SortValue](#) ([HtmlPageHandler](#) *pv)
Compares handler priorities for sorting (-1, 0, 1)
- void [Remove](#) ()
Removes handler from the linked list.

Protected Attributes inherited from [HtmlPageHandler](#)

- [HtmlPageHandler](#) * [m_pNextHandler](#)
Pointer to next handler in the linked list for efficient traversal.
- const char * [m_pUrlName](#)
URL pattern to match. Empty string or NULL matches all requests.
- int [m_access_group](#)
Access control group identifier for authorization checks.
- [HTTP_RequestTypes](#) [m_requestTypes](#)
HTTP method types this handler responds to (GET, POST, etc.)

16.40.1 Detailed Description

Implements the [HtmlPageHandler](#) class as a function pointer callback for GET requests.

16.40.2 Constructor & Destructor Documentation

16.40.2.1 [CallBackFunctionPageHandler\(\)](#) [1/2]

```
CallBackFunctionPageHandler::CallBackFunctionPageHandler (
    const char * pUrl,
    http_gethandlerfunc * pFunction,
    HTTP_RequestTypes reqType = tGet,
    int accessGroup = 0,
    bool beforeFiles = false) [inline]
```

Constructor for HTTP GET callback function.

Parameters

<i>pUrl</i>	Pointer to the URL to intercept from normal system processing
<i>pFunction</i>	Pointer to the function to call for the requested URL
<i>reqType</i>	Optional type of request to intercept, HTTP_RequestTypes . Default is tGet
<i>accessGroup</i>	Optional password group access level. Default is 0 = no password, maximum is 255.
<i>beforeFiles</i>	Optional parameter to specify a check for just a name, or the name of an actual file in the system. true = intercept before checking for a file of the same name false = check for any files of the requested name before intercepting

16.40.2.2 CallbackFunctionPageHandler() [2/2]

```

CallbackFunctionPageHandler::CallbackFunctionPageHandler (
    const char * pUrl,
    http_gethandlerfunc * pFunction,
    http_matchhandlerfunc * pMatchFunction,
    HTTP_RequestTypes reqType = tGet,
    int accessGroup = 0,
    bool beforeFiles = false) [inline]

```

Constructor for HTTP GET callback function, includes option for function to match the requested name.

Parameters

<i>pUrl</i>	Pointer to the URL to intercept from normal system processing
<i>pFunction</i>	Pointer to the function to call for the requested URL
<i>pMatchFunction</i>	Pointer to function to perform the match check
<i>reqType</i>	Optional type of request to intercept, HTTP_RequestTypes . Default is tGet
<i>accessGroup</i>	Optional parameter to set a password group access level. Default is 0 = no password.
<i>beforeFiles</i>	Optional parameter to process a file name before checking for a compiled-in file of the same name. Default is false = check for any files of the requested name before intercepting

16.40.3 Member Function Documentation

16.40.3.1 ProcessRaw()

```

virtual int CallbackFunctionPageHandler::ProcessRaw (
    int sock,
    HTTP_Request & pd) [inline], [virtual]

```

Pure virtual method that must be implemented to process HTTP requests.

This method is called when a matching request is received and must contain the actual request processing logic. Implementations should read request data, perform necessary processing, and send appropriate HTTP responses.

Parameters

<i>sock</i>	Socket file descriptor for the client connection
<i>pd</i>	HTTP request object containing parsed request details (headers, URL, method, body data, etc.)

Returns

Non-zero if the request was successfully processed and a response was sent, or 0 if the request was not processed (allows fallback to other handlers)

Implements [HtmlPageHandler](#).

The documentation for this class was generated from the following file:

- [http.h](#)

16.41 CallbackFunctionPostHandler Class Reference

Implements the HtmlPostHandler class as a function pointer callback for POST requests.

```
#include <httppost.h>
```

Inherits HtmlPostHandler.

Public Member Functions

- virtual int [ProcessRaw](#) (int sock, [HTTP_Request](#) &pdt)
Pure virtual method that must be implemented to process HTTP requests.

16.41.1 Detailed Description

Implements the HtmlPostHandler class as a function pointer callback for POST requests.

16.41.2 Member Function Documentation

16.41.2.1 ProcessRaw()

```
virtual int CallbackFunctionPostHandler::ProcessRaw (
    int sock,
    HTTP\_Request & pd) [inline], [virtual]
```

Pure virtual method that must be implemented to process HTTP requests.

This method is called when a matching request is received and must contain the actual request processing logic. Implementations should read request data, perform necessary processing, and send appropriate HTTP responses.

Parameters

<i>sock</i>	Socket file descriptor for the client connection
<i>pd</i>	HTTP request object containing parsed request details (headers, URL, method, body data, etc.)

Returns

Non-zero if the request was successfully processed and a response was sent, or 0 if the request was not processed (allows fallback to other handlers)

Implements [HtmlPageHandler](#).

The documentation for this class was generated from the following file:

- httppost.h

16.42 canMCF5441x::CanRxMessage Class Reference

Class to hold received CAN messages.

```
#include <multican.h>
```

Public Member Functions

- uint8_t [GetLength](#) ()
Returns the amount of data stored in the message.
- uint8_t [GetData](#) (uint8_t *buffer, uint8_t max_len)
Copy the data in the message up to max_len.
- uint32_t [GetId](#) ()
Returns the ID of the message.
- uint16_t [GetTimeStamp](#) ()
Returns the time stamp of the message.
- BOOL [IsValid](#) ()

Check to verify the [CanRxMessage](#) is a valid message.

- [CanRxMessage](#) (OS_FIFO *pFifo, uint16_t timeout)

Build a [CanRxMessage](#) from a FIFO.

- [CanRxMessage](#) (int moduleNum, uint32_t id, uint16_t timeout)

Constructor that sends out a RTR request to ID and waits for a response.

- [~CanRxMessage](#) ()

Destructor.

- BOOL [GetNewMessage](#) (OS_FIFO *pFifo, uint16_t timeout)

Get a new message from the FIFO. If no message is available, wait up to the timeout for one to be received.

16.42.1 Detailed Description

Class to hold received CAN messages.

16.42.2 Constructor & Destructor Documentation

16.42.2.1 CanRxMessage() [1/2]

```
canMCF5441x::CanRxMessage::CanRxMessage (
    OS_FIFO * pFifo,
    uint16_t timeout)
```

Build a [CanRxMessage](#) from a FIFO.

The FIFO must be registered to listen for incoming messages. If no messages are received in the timeout interval, the returned [CanRxMessage](#) will be marked as not valid. A Timeout value of 0 will wait forever.

Parameters

<i>pFifo</i>	Pointer to an OS_FIFO to store the message
<i>timeout</i>	Message timeout in TICKS_PER_SECOND

16.42.2.2 CanRxMessage() [2/2]

```
canMCF5441x::CanRxMessage::CanRxMessage (
    int moduleNum,
    uint32_t id,
    uint16_t timeout)
```

Constructor that sends out a RTR request to ID and waits for a response.

The CAN system uses any unused channel to send and receive the buffer. The constructor can return an invalid message for two reasons:

1. The timeout interval has expired
2. There were no free CAN channels available to send the request

Parameters

<i>moduleNum</i>	CAN module number
<i>id</i>	CAN ID number
<i>timeout</i>	Message timeout in TICKS_PER_SECOND. A value of 0 will wait forever.

16.42.3 Member Function Documentation

16.42.3.1 GetData()

```
uint8_t canMCF5441x::CanRxMessage::GetData (  
    uint8_t * buffer,  
    uint8_t max_len)
```

Copy the data in the message up to *max_len*.

Parameters

<i>buffer</i>	Pointer to destination storage buffer
<i>max_len</i>	Maximum number of bytes to copy

Returns

The number of bytes copied

16.42.3.2 GetNewMessage()

```

BOOL canMCF5441x::CanRxMessage::GetNewMessage (
    OS_FIFO * pFifo,
    uint16_t timeout)

```

Get a new message from the FIFO. If no message is available, wait up to the timeout for one to be received.

Parameters

<i>pFifo</i>	Pointer to the OS_FIFO.
<i>timeout</i>	Timeout to wait if no messages are currently available. Value is in TICKS_PER_SECOND, a value of 0 will wait forever.

16.42.3.3 IsValid()

```

BOOL canMCF5441x::CanRxMessage::IsValid ()

```

Check to verify the [CanRxMessage](#) is a valid message.

Please refer to the CanRxMessage() constructor for further information on valid messages.

The documentation for this class was generated from the following file:

- multican.h

16.43 mcanMODM7AE70::CanRxMessage Class Reference

Class to hold received CAN messages.

```
#include <mcan.h>
```

Public Member Functions

- uint8_t [GetLength](#) ()
Returns the amount of data stored in the message.
- uint8_t [CopyData](#) (uint8_t *buffer, uint8_t max_len)
Copy the data in the message up to max_len.
- const uint8_t * [GetData](#) ()
Returns a pointer to the message data.
- uint32_t [GetId](#) ()
Returns the ID of the message.
- uint16_t [GetTimeStamp](#) ()
Returns the time stamp of the message.
- BOOL [IsValid](#) ()
Check to verify the [CanRxMessage](#) is a valid message.
- [CanRxMessage](#) (OS_FIFO *pFifo, uint32_t timeout=WAIT_FOREVER)
Build a [CanRxMessage](#) from a FIFO.
- [~CanRxMessage](#) ()
Destructor.
- BOOL [GetNewMessage](#) (OS_FIFO *pFifo, uint32_t timeout=WAIT_FOREVER)
Get a new message from the FIFO. If no message is available, wait up to the timeout for one to be received.

16.43.1 Detailed Description

Class to hold received CAN messages.

16.43.2 Constructor & Destructor Documentation

16.43.2.1 CanRxMessage()

```
mcanMODM7AE70::CanRxMessage::CanRxMessage (
    OS_FIFO * pFifo,
    uint32_t timeout = WAIT_FOREVER)
```

Build a [CanRxMessage](#) from a FIFO.

The FIFO must be registered to listen for incoming messages. If no messages are received in the timeout interval, the returned [CanRxMessage](#) will be marked as not valid. A Timeout value of 0 will wait forever.

Parameters

<i>pFifo</i>	Pointer to an OS_FIFO to store the message
<i>timeout</i>	Optional message timeout in TICKS_PER_SECOND

16.43.3 Member Function Documentation

16.43.3.1 CopyData()

```
uint8_t mcanMODM7AE70::CanRxMessage::CopyData (
    uint8_t * buffer,
    uint8_t max_len)
```

Copy the data in the message up to *max_len*.

Parameters

<i>buffer</i>	Pointer to destination storage buffer
<i>max_len</i>	Maximum number of bytes to copy

Returns

The number of bytes copied

16.43.3.2 GetData()

```
const uint8_t * mcanMODM7AE70::CanRxMessage::GetData ()
```

Returns a pointer to the message data.

Warning

Invalid after destructor is called

16.43.3.3 GetId()

```
uint32_t mcanMODM7AE70::CanRxMessage::GetId ()
```

Returns the ID of the message.

Returns

The message ID

16.43.3.4 GetLength()

```
uint8_t mcanMODM7AE70::CanRxMessage::GetLength ()
```

Returns the amount of data stored in the message.

Returns

Length of data stored

16.43.3.5 GetNewMessage()

```
BOOL mcanMODM7AE70::CanRxMessage::GetNewMessage (
    OS_FIFO * pFifo,
    uint32_t timeout = WAIT_FOREVER)
```

Get a new message from the FIFO. If no message is available, wait up to the timeout for one to be received.

Parameters

<i>pFifo</i>	Pointer to the OS_FIFO
<i>timeout</i>	Optional parameter to wait if no messages are currently available. Value is in TICKS_PER_SECOND, a value of 0 will wait forever.

Returns

True on success, otherwise false

16.43.3.6 GetTimeStamp()

```
uint16_t mcanMODM7AE70::CanRxMessage::GetTimeStamp ()
```

Returns the time stamp of the message.

For timestamp generation the MCAN supplies a 16-bit wrap-around counter. A prescaler TSCC.TCP can be configured to clock the counter in multiples of CAN bit times (1-16). The counter is readable via MCAN_TSCV.TSC. A write access to the Timestamp Counter Value register (MCAN_TSCV) resets the counter to zero. When the timestamp counter wraps around, interrupt flag MCAN_IR.TSW is set. On start of frame reception / transmission the counter value is captured and stored into the timestamp section of an Rx Buffer / Rx FIFO (RXTS[15:0]) or Tx Event FIFO (TXTS[15:0]) element.

The internal/external Timestamp Counter value is captured on start of frame (both Receive and Transmit). When MCAN_TSCC.TSS = 1, the Timestamp Counter is incremented in multiples of CAN bit times [1-16] depending on the configuration of MCAN_TSCC.TCP. A wrap around sets interrupt flag MCAN_IR.TSW. Write access resets the counter to zero.

Returns

Time stamp of the message

16.43.3.7 IsValid()

```
BOOL mcanMODM7AE70::CanRxMessage::IsValid ()
```

Check to verify the [CanRxMessage](#) is a valid message.

Please refer to the CanRxMessage() constructor for further information on valid messages.

Returns

True if valid, otherwise false

The documentation for this class was generated from the following file:

- mcan.h

16.44 CertData Class Reference

```
#include <certgen.h>
```

Inherits [config_obj](#).

Additional Inherited Members

Public Member Functions inherited from [config_obj](#)

- [config_obj](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_obj](#) (const char *name, const char *desc)
Object constructor.
- virtual void [GetTextValue](#) ([NBString](#) &s) override
Get the object value as a text string to the specified [NBString](#) object.
- virtual void [GetTypeValue](#) ([NBString](#) &s) override
Assigns the object type value to the specified [NBString](#) object.

16.44.1 Detailed Description

The documentation for this class was generated from the following file:

- certgen.h

16.45 CertGenData Struct Reference

Used to store the information that is passed in when enabling onboard generated certificates, as well as the information needed to determine when they need to be regenerated.

```
#include <certgen.h>
```

Public Attributes

- const char * **m_commonName**
Common Name.
- int **m_yrsValid**
Number of years.
- [NBString](#) **m_altNamesString**
Alternate names string.
- time_t **m_certExpTime**
Certificate expiration time.

16.45.1 Detailed Description

Used to store the information that is passed in when enabling onboard generated certificates, as well as the information needed to determine when they need to be regenerated.

The documentation for this struct was generated from the following file:

- certgen.h

16.46 MQTT::Client Class Reference

Advanced low-level [MQTT Client](#) for interacting with Mqtt Brokers. The client handles all network communications with the Broker and handles local message queueing and dispatching to topic handlers.

```
#include <mqtt.h>
```

Inherits [TimeOutManager](#).

Inherited by [MQTT::ConfiguredClient](#).

Public Types

- enum [eSessionLimit](#) {
[eLimit_Size_Tx](#) , [eLimit_Size_Rx](#) , [eLimit_InFlight_Tx](#) , [eLimit_InFlight_Rx](#) ,
[eLimit_Subscriptions](#) , [eLimit_SubAliases](#) , [eLimit_PublishAliases](#) , [eLimit_QoS](#) ,
[eLimit_SessionExpiry](#) }
Limit types for the [Client](#) interactions with the Broker.
- enum [eRateLimitType](#) { [eRateLimit_Publish](#) , [eRateLimit_Subscribe](#) }
Rate limit types for [Client](#) messaging.
- enum [eRateLimitAdjustMethod](#) { [eRateLimitAdj_None](#) , [eRateLimitAdj_Set](#) , [eRateLimitAdj_Add](#) ,
[eRateLimitAdj_Subtract](#) }
How to adjust rate limit buckets when modifying the current rate limit.
- enum [eConnectionStatus](#) {
[eConnStat_Error](#) , [eConnStat_No_Broker_Configured](#) , [eConnStat_Resolving](#) , [eConnStat_InProgress](#) ,
[eConnStat_Established](#) , [eConnStat_Refused](#) , [eConnStat_Disconnected](#) }
Connection state values.

Public Member Functions

- [Client](#) (const char *brokerURI, const char *pClientId, uint16_t maxKeepAlive=[MQTT_DEFAULT_KEEP_ALIVE](#), bool bCleanStart=true, [PublishRequest](#) *lwtMsg=NULL, const char *pConnMsg=NULL, const char *pDisconnMsg=NULL, const char *pUsername=NULL, const char *pPassword=NULL, RequestProperty *addProps=NULL, bool ConnectOnStartup=true)
Constructor. This version takes a raw string for the broker URI.
- [Client](#) (const [ParsedURI](#) *brokerURI, const char *pClientId, uint16_t maxKeepAlive=[MQTT_DEFAULT_KEEP_ALIVE](#), bool bCleanStart=true, [PublishRequest](#) *lwtMsg=NULL, const char *pConnMsg=NULL, const char *pDisconnMsg=NULL, const char *pUsername=NULL, const char *pPassword=NULL, RequestProperty *addProps=NULL, bool ConnectOnStartup=true)
Constructor. This version takes a [ParsedURI](#) object for the broker URI.
- int [Connect](#) (bool bWaitForConnection, const char *pNewBrokerURI, ConnAckCb_t notifyCb=NULL, void *notifyCtx=NULL)
Begin the process of Connecting the client to the Broker.
- int [Connect](#) (bool bWaitForConnection=true, const [ParsedURI](#) *pNewBrokerURI=NULL, ConnAckCb_t notifyCb=NULL, void *notifyCtx=NULL)
Begin the process of Connecting the client to the Broker.
- int [Disconnect](#) ()
Disconnect the [Client](#) from the Broker.
- void [SetConnectNotify](#) (ConnAckCb_t notifyCb, void *notifyCtx)
Set the connection notification callback and it's context information.
- int [SetRateLimit](#) ([eRateLimitType](#) type, uint8_t bucketMax, uint8_t fillCount, uint8_t fillTicks, [eRateLimitAdjustMethod](#) adjMethod, uint8_t adjCount)
Set the value for a given rate limit.
- int [SetRateLimit_Publish](#) (int reqPerSecond)
Set the Publish rate limit. This method automatically determines the fillCount and fillTicks to use for refilling the credit bucket.
- int [SetRateLimit_Subscribe](#) (int reqPerSecond)
Set the Subscribe rate limit. This method automatically determines the fillCount and fillTicks to use for refilling the credit bucket.
- [eConnectionStatus](#) [GetConnectionStatus](#) ()
Get the current state of the [Client](#)'s connection to the Broker.
- void [SetClientCert](#) (uint8_t *cert, uint32_t len, uint8_t *privkey, uint32_t privlen)
Set the [Client](#) Certificate and Private Key to use when connecting to a Broker via TLS.
- void [GetClientCert](#) (uint8_t **cert, uint32_t *len, uint8_t **privkey, uint32_t *privlen)
Set the [Client](#) Certificate and Private Key to use when connecting to a Broker via TLS.

- int [QueuePublish](#) ([mqtt_leaf](#) &leaf)
Queue an MqttObj to be published.
- int [QueueSubscribe](#) ([mqtt_leaf](#) &leaf)
Queue an MqttObj to subscribe.
- int [QueueUnsubscribe](#) ([mqtt_leaf](#) &leaf)
Queue an MqttObj to unsubscribe.
- int [QueueAllSubscriptions](#) ()
Queue all [Client](#)'s registered MqttObjs to be subscribed and published according to the object's operation flags.
- int [Publish](#) (const char *topic, uint8_t *pData, uint32_t len, int prio, TickTimeout &t, uint8_t qos=0)
Publish a binary payload message.
- int [Publish](#) (const char *topic, const char *pStr, int prio, TickTimeout &t, uint8_t qos=0)
Publish a message.
- int [Publish](#) ([PublishRequest](#) *req, int prio, TickTimeout &t)
Publish a fully defined request.
- int [QueuePublish](#) ([PublishRequest](#) *req, int *prio)
Queue a publish message from a fully defined request.
- int [Subscribe](#) ([TopicHandler](#) &handler, int *handlerIdx, TickTimeout &t, uint8_t subOpts=Pkt::eSubOpt_No↔ Local)
Register a [TopicHandler](#) and subscribe its Topic on the Broker.
- int [Subscribe](#) ([SubscribeRequest](#) *req, int *handlerIdx, TickTimeout &t)
Register a [TopicHandler](#) (via fully specified [SubscribeRequest](#)) and subscribe its Topic on the Broker.
- int [QueueSubscribe](#) (SubUnsubQueueMsg *qmsg)
Register a [TopicHandler](#) and subscribe its Topic on the Broker.
- int [Unsubscribe](#) (int handlerIdx, TickTimeout &t, const char *topicFilter=NULL)
Unsubscribe and deregister a [TopicHandler](#).
- int [Unsubscribe](#) (const char *topicFilter, TickTimeout &t)
Unsubscribe and deregister a [TopicHandler](#).
- int [QueueUnsubscribe](#) (SubUnsubQueueMsg *qmsg)
Queue unsubscribing and deregistering a [TopicHandler](#).
- int [RegisterTopicHandler](#) ([TopicHandler](#) *newHandler, int *handlerIdx)
Register a [TopicHandler](#) with the [Client](#).
- int [RemoveTopicHandler](#) ([TopicHandler](#) *handler)
Remove/deregister a [TopicHandler](#) from the [Client](#) by reference.
- int [RemoveTopicHandler](#) (int handlerIdx)
Remove/deregister a [TopicHandler](#) from the [Client](#) by index.
- void [RegisterMqttObj](#) ([mqtt_leaf](#) &leaf, uint32_t interval=0)
Register an MqttObj with the [Client](#).
- void [RemoveObj](#) ([mqtt_leaf](#) &leaf)
Deregister/remove an MqttObj from the [Client](#).
- int [SendNextPubReq](#) ()
Send the next queued Publish Request.
- int [SendNextSubReq](#) ()
Send the next queued SubUnSub Request.

Static Public Member Functions

- static void [StartTask](#) (uint8_t taskPrio=MQTT_TASK_PRIO)
Start the [Client](#) Task.
- static [Client](#) * [GetFirstClient](#) ()
Start the [Client](#) Task.

16.46.1 Detailed Description

Advanced low-level [MQTT Client](#) for interacting with Mqtt Brokers. The client handles all network communications with the Broker and handles local message queueing and dispatching to topic handlers.

Basic users should use the [ConfiguredClient](#) for easier implementation.

16.46.2 Member Enumeration Documentation

16.46.2.1 eConnectionStatus

enum [MQTT::Client::eConnectionStatus](#)

Connection state values.

Enumerator

eConnStat_Error	General error with client connection.
eConnStat_No_Broker_Configured	Error, no broker configured to be used.
eConnStat_Resolving	Currently resolving broker address.
eConnStat_InProgress	Connection to broker in progress.
eConnStat_Established	Connection with broker established.
eConnStat_Refused	Connection refused by broker.

16.46.2.2 eRateLimitAdjustMethod

enum [MQTT::Client::eRateLimitAdjustMethod](#)

How to adjust rate limit buckets when modifying the current rate limit.

Enumerator

eRateLimitAdj_None	Do not modify the current limit token count.
eRateLimitAdj_Set	Set the current limit token count to the adjustment value.
eRateLimitAdj_Add	Add the adjustment value to the current token count.
eRateLimitAdj_Subtract	Subtract the adjustment value from the current token count.

16.46.2.3 eRateLimitType

enum [MQTT::Client::eRateLimitType](#)

Rate limit types for [Client](#) messaging.

Enumerator

eRateLimit_Publish	Limit for Publish requests per second.
eRateLimit_Subscribe	Limit for Subscribe requests per second.

16.46.2.4 eSessionLimit

enum [MQTT::Client::eSessionLimit](#)

Limit types for the [Client](#) interactions with the Broker.

Enumerator

eLimit_Size_Tx	Maximum receive message size accepted.
eLimit_Size_Rx	Maximum transmit message size allowed.
eLimit_InFlight_Tx	Maximum number of outstanding tx messages with QoS > 0.

Enumerator

eLimit_InFlight_Rx	Maximum number of outstanding rx messages with QoS > 0.
eLimit_Subscriptions	Maximum number of topic subscriptions.
eLimit_SubAliases	Maximum number of subscription aliases.
eLimit_PublishAliases	Maximum number of Publish topic name aliases.
eLimit_QoS	Maximum QoS for messages.
eLimit_SessionExpiry	Maximum time for Session before expiring.

16.46.3 Constructor & Destructor Documentation

16.46.3.1 Client() [1/2]

```
MQTT::Client::Client (
    const char * brokerURI,
    const char * pClientId,
    uint16_t maxKeepAlive = MQTT_DEFAULT_KEEP_ALIVE,
    bool bCleanStart = true,
    PublishRequest * lwtMsg = NULL,
    const char * pConnMsg = NULL,
    const char * pDisconnMsg = NULL,
    const char * pUsername = NULL,
    const char * pPassword = NULL,
    RequestProperty * addProps = NULL,
    bool ConnectOnStartup = true)
```

Constructor. This version takes a raw string for the broker URI.

Parameters

<i>*brokerURI</i>	The URI for the broker to use. (Must be a URI with protocol, like <code>mqtt://1.2.3.4</code>)
<i>*pClientId</i>	A string containing the Client ID to use with the broker
<i>maxKeepAlive</i>	The number of seconds when, after multiple failed MQTT keepalive messages, the connection will be automatically restarted (Note: the server may impose a different maximum keepalive)
<i>bCleanStart</i>	Whether the broker start a fresh session and remove any previous subscriptions.
<i>*lwtMsg</i>	Pointer to a PublishRequest containing the Last Will and Testament to use for this client's connections
<i>*pConnMsg</i>	Message payload to send to the LWT topic after connecting to the broker.
<i>*pDisconnMsg</i>	Message payload to send to the LWT topic just prior to intentionally disconnecting from the broker.
<i>*pUsername</i>	Username to use when connecting to the broker.
<i>*pPassword</i>	Password to use when connecting to the broker.
<i>*addProps</i>	Additional message Properties to send with Connection Requests
<i>ConnectOnStartup</i>	Whether the client should automatically connect after startup.

16.46.3.2 Client() [2/2]

```
MQTT::Client::Client (
    const ParsedURI * brokerURI,
    const char * pClientId,
    uint16_t maxKeepAlive = MQTT_DEFAULT_KEEP_ALIVE,
    bool bCleanStart = true,
    PublishRequest * lwtMsg = NULL,
```

```

const char * pConnMsg = NULL,
const char * pDisconnMsg = NULL,
const char * pUsername = NULL,
const char * pPassword = NULL,
RequestProperty * addProps = NULL,
bool ConnectOnStartup = true)

```

Constructor. This version takes a [ParsedURI](#) object for the broker URI.

Parameters

<i>*brokerURI</i>	The URI for the broker to use. (Must be a URI with protocol, like <code>mqtt://1.2.3.4</code>)
<i>*pClientId</i>	A string containing the Client ID to use with the broker
<i>maxKeepAlive</i>	The number of seconds when, after multiple failed MQTT keepalive messages, the connection will be automatically restarted (Note: the server may impose a different maximum keepalive)
<i>bCleanStart</i>	Whether the broker start a fresh session and remove any previous subscriptions.
<i>*lwtMsg</i>	Pointer to a PublishRequest containing the Last Will and Testament to use for this client's connections
<i>*pConnMsg</i>	Message payload to send to the LWT topic after connecting to the broker.
<i>*pDisconnMsg</i>	Message payload to send to the LWT topic just prior to intentionally disconnecting from the broker.
<i>*pUsername</i>	Username to use when connecting to the broker.
<i>*pPassword</i>	Password to use when connecting to the broker.
<i>*addProps</i>	Additional message Properties to send with Connection Requests
<i>ConnectOnStartup</i>	Whether the client should automatically connect after startup.

16.46.4 Member Function Documentation

16.46.4.1 Connect() [1/2]

```

int MQTT::Client::Connect (
    bool bWaitForConnection,
    const char * pNewBrokerURI,
    ConnAckCb_t notifyCb = NULL,
    void * notifyCtx = NULL)

```

Begin the process of Connecting the client to the Broker.

Parameters

<i>bWaitForConnection</i>	Wait for the client to finish connecting to the broker before returning.
<i>pNewBrokerURI</i>	If non-null, a pointer to the new Broker URI to use
<i>notifyCb</i>	Pointer to a notification callback for connection state events.
<i>notifyCtx</i>	Context to pass to the notification callback as additional data.

Returns

`eResult_t` indicating result

16.46.4.2 Connect() [2/2]

```

int MQTT::Client::Connect (
    bool bWaitForConnection = true,
    const ParsedURI * pNewBrokerURI = NULL,
    ConnAckCb_t notifyCb = NULL,
    void * notifyCtx = NULL)

```

Begin the process of Connecting the client to the Broker.

Parameters

<i>bWaitForConnection</i>	Wait for the client to finish connecting to the broker before returning.
<i>pNewBrokerURI</i>	If non-null, a pointer to the new Broker ParsedURI to use
<i>notifyCb</i>	Pointer to a notification callback for connection state events.
<i>notifyCtx</i>	Context to pass to the notification callback as additional data.

Returns

`eResult_t` indicating result

16.46.4.3 Disconnect()

```
int MQTT::Client::Disconnect ()
```

Disconnect the [Client](#) from the Broker.

Returns

`eResult_t` indicating result

16.46.4.4 GetClientCert()

```
void MQTT::Client::GetClientCert (
    uint8_t ** cert,
    uint32_t * len,
    uint8_t ** privkey,
    uint32_t * privlen) [inline]
```

Set the [Client](#) Certificate and Private Key to use when connecting to a Broker via TLS.

Parameters

<i>**cert</i>	Pointer to return pointer for the certificate buffer address
<i>*len</i>	Pointer to return value for the length of the certificate
<i>**privkey</i>	Pointer to return pointer for the the private key buffer address
<i>*privlen</i>	Pointer to return value for the length of the private key

See also

[Client::GetClientCert](#)

16.46.4.5 GetConnectionStatus()

```
eConnectionStatus MQTT::Client::GetConnectionStatus ()
```

Get the current state of the [Client](#)'s connection to the Broker.

Returns

`eConnectionStatus` The current status of the client's connection to the Broker

See also

[Client::Connect](#), [Client::Disconnect](#), [Client::GetConnectionStatusString](#)

16.46.4.6 GetFirstClient()

```
static Client * MQTT::Client::GetFirstClient () [inline], [static]
```

Start the [Client](#) Task.

Returns

Pointer to the first registered [Client](#)

16.46.4.7 Publish() [1/3]

```
int MQTT::Client::Publish (
    const char * topic,
    const char * pStr,
    int prio,
    TickTimeout & t,
    uint8_t qos = 0)
```

Publish a message.

Parameters

<i>*topic</i>	Topic to publish message to
<i>*pStr</i>	String containing data to be published
<i>prio</i>	Priority level of the message to be published (see MQTT_OBJ_PRIORITY_MAX and MQTT_OBJ_PRIORITY_MIN)
<i>&t</i>	Timeout indicating how long to block before returning timeout error
<i>qos</i>	Quality of Service level to use for the resulting Publish message

Returns

eResult_t return code

See also

[Client::QueuePublish](#)

16.46.4.8 Publish() [2/3]

```
int MQTT::Client::Publish (
    const char * topic,
    uint8_t * pData,
    uint32_t len,
    int prio,
    TickTimeout & t,
    uint8_t qos = 0)
```

Publish a binary payload message.

Parameters

<i>*topic</i>	Topic to publish message to
<i>*pData</i>	Data to be published
<i>len</i>	Length of data to be published
<i>prio</i>	Priority level of the message to be published (see MQTT_OBJ_PRIORITY_MAX and MQTT_OBJ_PRIORITY_MIN)
<i>&t</i>	Timeout indicating how long to block before returning timeout error
<i>qos</i>	Quality of Service level to use for the resulting Publish message

Returns

eResult_t return code

See also

[Client::QueuePublish](#)

16.46.4.9 Publish() [3/3]

```
int MQTT::Client::Publish (  
    PublishRequest * req,  
    int prio,  
    TickTimeout & t)
```

Publish a fully defined request.

Parameters

<i>*req</i>	Publish Request describing the message to be sent
<i>prio</i>	Priority level of the message to be published (see MQTT_OBJ_PRIORITY_MAX and MQTT_OBJ_PRIORITY_MIN)
<i>&t</i>	Timeout indicating how long to block before returning timeout error

Returns

eResult_t return code

See also

[Client::QueuePublish](#)

16.46.4.10 QueueAllSubscriptions()

```
int MQTT::Client::QueueAllSubscriptions ()
```

Queue all [Client](#)'s registered MqttObjs to be subscribed and published according to the object's operation flags.

Returns

eResult_t return code

See also

[Client::QueuePublish](#), [Client::QueueSubscribe](#)

16.46.4.11 QueuePublish() [1/2]

```
int MQTT::Client::QueuePublish (  
    mqtt_leaf & leaf)
```

Queue an MqttObj to be published.

Parameters

<i>&leaf</i>	Reference to the object to publish.

Returns

eResult_t return code

See also

[Client::QueueSubscribe](#), [Client::QueueUnsubscribe](#)

16.46.4.12 QueuePublish() [2/2]

```
int MQTT::Client::QueuePublish (  
    PublishRequest * req,  
    int * prio)
```

Queue a publish message from a fully defined request.

Parameters

<i>*req</i>	Publish Request describing the message to be sent
<i>prio</i>	Priority level of the message to be published (see MQTT_OBJ_PRIORITY_MAX and MQTT_OBJ_PRIORITY_MIN)

Returns

eResult_t return code

See also

[Client::Publish](#)

16.46.4.13 QueueSubscribe() [1/2]

```
int MQTT::Client::QueueSubscribe (
    mqtt_leaf & leaf)
```

Queue an MqttObj to subscribe.

Parameters

<i>&leaf</i>	Reference to the object to subscribe.

Returns

eResult_t return code

See also

[Client::QueuePublish](#), [Client::QueueUnsubscribe](#)

16.46.4.14 QueueSubscribe() [2/2]

```
int MQTT::Client::QueueSubscribe (
    SubUnsubQueueMsg * qmsg)
```

Register a [TopicHandler](#) and subscribe its Topic on the Broker.

Parameters

<i>*qmsg</i>	Pointer to asynchronous Subscription Request structure

Returns

eResult_t return code

See also

[Client::Subscribe](#)

16.46.4.15 QueueUnsubscribe() [1/2]

```
int MQTT::Client::QueueUnsubscribe (
    mqtt_leaf & leaf)
```

Queue an MqttObj to unsubscribe.

Parameters

<i>&leaf</i>	Reference to the object to unsubscribe.

Returns

eResult_t return code

See also

[Client::QueuePublish](#), [Client::QueueSubscribe](#)

16.46.4.16 QueueUnsubscribe() [2/2]

```
int MQTT::Client::QueueUnsubscribe (
    SubUnsubQueueMsg * qmsg)
```

Queue unsubscribing and deregistering a [TopicHandler](#).

Parameters

<i>*qmsg</i>	Pointer to fully defined unsubscription request

Returns

eResult_t return code

See also

[Client::Unsubscribe](#)

16.46.4.17 RegisterMqttObj()

```
void MQTT::Client::RegisterMqttObj (
    mqtt_leaf & leaf,
    uint32_t interval = 0)
```

Register an MqttObj with the [Client](#).

Parameters

<i>&leaf</i>	Reference to the MqttObj to be registered with the Client
<i>interval</i>	Publication interval (in TimeTicks) for the registered object

See also

[Client::RemoveObj](#)

16.46.4.18 RegisterTopicHandler()

```
int MQTT::Client::RegisterTopicHandler (
    TopicHandler * newHandler,
    int * handlerIdx)
```

Register a [TopicHandler](#) with the [Client](#).

Parameters

<i>*newHandler</i>	Pointer to the TopicHanler to Register
<i>*handlerIdx</i>	Pointer to return value of the index the handler is registered at

Returns

eResult_t return code

See also

[Client::Subscribe](#)

16.46.4.19 RemoveObj()

```
void MQTT::Client::RemoveObj (
    mqtt_leaf & leaf)
```

Deregister/remove an MqttObj from the [Client](#).

Parameters

<i>&leaf</i>	Reference to the MqttObj to be removed from the Client

See also

[Client::RemoveObj](#)

16.46.4.20 RemoveTopicHandler() [1/2]

```
int MQTT::Client::RemoveTopicHandler (
    int handlerIdx)
```

Remove/deregister a [TopicHandler](#) from the [Client](#) by index.

Parameters

<i>handlerIdx</i>	Registration index of the topic handler to be removed

Returns

eResult_t return code

See also

[Client::Unsubscribe](#)

16.46.4.21 RemoveTopicHandler() [2/2]

```
int MQTT::Client::RemoveTopicHandler (
    TopicHandler * handler)
```

Remove/deregister a [TopicHandler](#) from the [Client](#) by reference.

Parameters

<i>*handler</i>	Pointer to the TopicHandler to be removed

Returns

eResult_t return code

See also

[Client::Unsubscribe](#)

16.46.4.22 SendNextPubReq()

```
int MQTT::Client::SendNextPubReq ()
```

Send the next queued Publish Request.

Returns

eResult_t return code

See also

[Client::SendNextSubReq](#)

16.46.4.23 SendNextSubReq()

```
int MQTT::Client::SendNextSubReq ()
```

Send the next queued SubUnSub Request.

Returns

eResult_t return code

See also

[Client::SendNextPubReq](#)

16.46.4.24 SetClientCert()

```
void MQTT::Client::SetClientCert (
    uint8_t * cert,
    uint32_t len,
    uint8_t * privkey,
    uint32_t privlen) [inline]
```

Set the [Client](#) Certificate and Private Key to use when connecting to a Broker via TLS.

Parameters

<i>*cert</i>	Pointer to the certificate buffer to use
<i>len</i>	Length of the certificate
<i>*privkey</i>	Pointer to the private key buffer
<i>privlen</i>	Length of the private key

See also

[Client::GetClientCert](#)

16.46.4.25 SetConnectNotify()

```
void MQTT::Client::SetConnectNotify (
    ConnAckCb_t notifyCb,
    void * notifyCtx)
```

Set the connection notification callback and it's context information.

Parameters

<i>notifyCb</i>	Notification callback for connection state events.
<i>notifyCtx</i>	Context to pass to the notification callback as additional data.

16.46.4.26 SetRateLimit()

```
int MQTT::Client::SetRateLimit (
    eRateLimitType type,
    uint8_t bucketMax,
    uint8_t fillCount,
    uint8_t fillTicks,
    eRateLimitAdjustMethod adjMethod,
    uint8_t adjCount)
```

Set the value for a given rate limit.

Parameters

<i>type</i>	The rate limit to set the value of.
<i>bucketMax</i>	The maximum count for the limit bucket (i.e. burst maximum)
<i>fillCount</i>	The count to add to the bucket each time interval
<i>fillTicks</i>	The interval, in TimeTicks, between each filling of the bucket.
<i>adjMethod</i>	The adjustment type to perform to the current bucket count when setting the new rate limit.
<i>adjCount</i>	The count to use when adjusting the current bucket count.

Returns

eResult_t Return code

See also

[Client::SetRateLimit_Publish](#), [Client::SetRateLimit_Subscribe](#)

16.46.4.27 SetRateLimit_Publish()

```
int MQTT::Client::SetRateLimit_Publish (
    int reqPerSecond)
```

Set the Publish rate limit. This method automatically determines the fillCount and fillTicks to use for refilling the credit bucket.

Parameters

<i>reqPerSecond</i>	The maximum number of allowed Publish messages per second.

Returns

eResult_t Return code

See also

[Client::SetRateLimit](#), [Client::SetRateLimit_Subscribe](#)

16.46.4.28 SetRateLimit_Subscribe()

```
int MQTT::Client::SetRateLimit_Subscribe (
    int reqPerSecond)
```

Set the Subscribe rate limit. This method automatically determines the fillCount and fillTicks to use for refilling the credit bucket.

Parameters

<i>reqPerSecond</i>	The maximum number of allowed Publish messages per second.

Returns

eResult_t Return code

See also

[Client::SetRateLimit](#), [Client::SetRateLimit_Publish](#)

16.46.4.29 StartTask()

```
static void MQTT::Client::StartTask (
    uint8_t taskPrio = MQTT_TASK_PRIO) [static]
```

Start the [Client](#) Task.

Parameters

<i>taskPrio</i>	The OS Task Priority to assign to the Client Task

16.46.4.30 Subscribe() [1/2]

```
int MQTT::Client::Subscribe (
    SubscribeRequest * req,
    int * handlerIdx,
    TickTimeout & t)
```

Register a [TopicHandler](#) (via fully specified [SubscribeRequest](#)) and subscribe its Topic on the Broker.

Parameters

<i>*req</i>	Fully described subscription request to be sent to the Broker
<i>*handlerIdx</i>	Pointer to int to return the registration index of the TopicHandler
<i>&t</i>	Timeout indicating how long to block before returning timeout error

Returns

eResult_t return code

See also

[Client::QueueSubscribe](#)

16.46.4.31 Subscribe() [2/2]

```
int MQTT::Client::Subscribe (
    TopicHandler & handler,
    int * handlerIdx,
    TickTimeout & t,
    uint8_t subOpts = Pkt::eSubOpt_NoLocal)
```

Register a [TopicHandler](#) and subscribe its Topic on the Broker.

Parameters

<i>&handler</i>	TopicHandler to be registered and subscribed
<i>*handlerIdx</i>	Pointer to int to return the registration index of the TopicHandler
<i>&t</i>	Timeout indicating how long to block before returning timeout error
<i>subOpts</i>	Flags describing Subscription options to request when Subscribing on the Broker

Returns

eResult_t return code

See also

[Client::QueueSubscribe](#)

16.46.4.32 Unsubscribe() [1/2]

```
int MQTT::Client::Unsubscribe (  
    const char * topicFilter,  
    TickTimeout & t)
```

Unsubscribe and deregister a [TopicHandler](#).

Parameters

<i>*topicFilter</i>	The Topic Filter of the registered TopicHandler and on the broker to be Unsubscribed from
<i>&t</i>	Timeout indicating how long to block before returning timeout error

Returns

eResult_t return code

See also

[Client::QueueUnsubscribe](#)

16.46.4.33 Unsubscribe() [2/2]

```
int MQTT::Client::Unsubscribe (  
    int handlerIdx,  
    TickTimeout & t,  
    const char * topicFilter = NULL)
```

Unsubscribe and deregister a [TopicHandler](#).

Parameters

<i>handlerIdx</i>	The registration index of the TopicHandler to unsubscribe and deregister
<i>&t</i>	Timeout indicating how long to block before returning timeout error
<i>*topicFilter</i>	The Topic Filter on the broker to be Unsubscribed from

Returns

eResult_t return code

See also

[Client::QueueUnsubscribe](#)

The documentation for this class was generated from the following file:

- mqtt.h

16.47 config_bool Class Reference

Boolean Configuration Variable.

#include <config_obj.h>

Inherits [config_value](#).

Inherited by reboot_obj.

Public Member Functions

- virtual void [GetTextValue](#) (NBString &s) override
Copy the object value as a text string to the specified NBString object.
- [config_bool](#) (config_obj &owner, bool def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_bool](#) (bool def_val, const char *name, const char *desc=NULL)
Object constructor.
- [operator bool](#) () const
Return the object value.
- [config_bool](#) & [operator=](#) (const bool v)
Assign the config_bool object value to the specified bool value.
- [config_bool](#) & [operator=](#) (const [config_bool](#) &cb)
Copy one config_bool object to another.
- [config_bool](#) & [operator=](#) (const int i)
Assign a config_bool object value to the specified integer value.
- virtual void [GetTypeValue](#) (NBString &s) override
Copy the object type value to the specified NBString object.

Additional Inherited Members

Protected Member Functions inherited from [config_value](#)

- [config_value](#) (config_obj &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.47.1 Detailed Description

Boolean Configuration Variable.

Note

All modifications to configuration objects are marked as pending. A call to [SaveConfigToStorage](#) is required to save changes to flash memory.

16.47.2 Constructor & Destructor Documentation

16.47.2.1 config_bool() [1/2]

```
config_bool::config_bool (
    config_obj & owner,
    bool def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Parameters

<i>owner</i>	Reference to parent leaf
<i>def_val</i>	Default value
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.47.2.2 config_bool() [2/2]

```
config_bool::config_bool (
    bool def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor.

Parameters

<i>def_val</i>	Default value
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description, used for info/tool tips

16.47.3 Member Function Documentation

16.47.3.1 GetTextValue()

```
virtual void config_bool::GetTextValue (
    NBString & s) [inline], [override], [virtual]
```

Copy the object value as a text string to the specified [NBString](#) object.

Writes either "true" or "false". This is a virtual function that can be overridden in a derived class.

Parameters

<i>s</i>	Reference to a NBString object to store the result

Implements [config_value](#).

16.47.3.2 GetTypeValue()

```
virtual void config_bool::GetTypeValue (
    NBString & s) [inline], [override], [virtual]
```

Copy the object type value to the specified [NBString](#) object.

Parameters

<i>s</i>	NBString object to hold the type value string

16.47.3.3 operator bool()

```
config_bool::operator bool () const [inline]
```

Return the object value.

Returns

The object value as a bool

Example: for a variable named `configInt` of type [config_double](#):

```
bool boolValue = bool(NV_Settings.m_nvBool);
iprintf("boolValue: %d\r\n", boolValue);
```

16.47.3.4 operator=() [1/3]

```
config_bool & config_bool::operator= (
    const bool v) [inline]
```

Assign the [config_bool](#) object value to the specified bool value.

Parameters

<i>v</i>	Value of type bool

16.47.3.5 operator=() [2/3]

```
config_bool & config_bool::operator= (
    const config_bool & cb) [inline]
```

Copy one [config_bool](#) object to another.

Parameters

<i>cb</i>	Reference to a config_bool object

16.47.3.6 operator=() [3/3]

```
config_bool & config_bool::operator= (
    const int i) [inline]
```

Assign a [config_bool](#) object value to the specified integer value.

Parameters

<i>i</i>	Integer value, either 0 (false) or 1 (true)

The documentation for this class was generated from the following file:

- [config_obj.h](#)

16.48 config_chooser Class Reference

Chooser Configuration Variable - Select From a List of Items.

```
#include <config_obj.h>
```

Inherits [config_obj](#).

Public Member Functions

- [config_chooser](#) ([config_obj](#) &owner, const char *name, const char *in_value, const char *in_choices, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_chooser](#) (const char *name, const char *in_value, const char *in_choices, const char *desc=NULL)
Object constructor.
- bool [IsSelected](#) (const char *choice)
Check if a particular choice option is selected.
- bool [IsSelected](#) (const [NBString](#) &s)
Check if a particular choice option is selected.
- bool [IsInChoices](#) (const char *str, size_t strLen)
Check if a string is in the list of possible choices. A comparison will continue until a null character is found, or the length of the longer of the two strings being compared is reached (strLen is used to determine if str is longer than the current element being compared.)
- bool [IsInChoices](#) (const [NBString](#) &str, size_t strLen)
Check if a string is in the list of possible choices. A comparison will continue until a null character is found, or the length of the longer of the two strings being compared is reached (strLen is used to determine if str is longer than the current element being compared.)
- const [config_string](#) & [GetChoices](#) ()

- *Get the list of choices.*
const [config_string](#) & [SetChoices](#) (const char *in_choices)
- *Set the list of choices.*
operator [NBString](#) () const
Returns the object value.
- [config_chooser](#) & operator= (const char *p)
Assign the selected list item from a const char value.*
- [config_chooser](#) & operator= (const [NBString](#) &s)
Assign the [config_string](#) object value from a [NBString](#) object.
- [config_chooser](#) & operator= (const [config_chooser](#) &ci)
Copy one [config_chooser](#) object to another.
- virtual void [GetTypeValue](#) ([NBString](#) &s) override
Assigns the object type value to the specified [NBString](#) object.

Public Member Functions inherited from [config_obj](#)

- [config_obj](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_obj](#) (const char *name, const char *desc)
Object constructor.
- virtual void [GetTextValue](#) ([NBString](#) &s) override
Get the object value as a text string to the specified [NBString](#) object.

16.48.1 Detailed Description

Chooser Configuration Variable - Select From a List of Items.

The [config_chooser](#) class provides a list of options with the ability to select one. If used on a web page it would appear as a drop down box. For example, the choices could be "one", "two", or "three", and "two" could be the item selected. Since the items are JSON, the selected item is identified by the option name, not a list index.

Note

All modifications to configuration objects are marked as pending. A call to [SaveConfigToStorage](#) is required to save changes to flash memory.

16.48.2 Constructor & Destructor Documentation

16.48.2.1 [config_chooser](#)() [1/2]

```
config_chooser::config_chooser (
    config\_obj & owner,
    const char * name,
    const char * in_value,
    const char * in_choices,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>name</i>	Field name assigned to the value
<i>in_value</i>	The option value selected
<i>in_choices</i>	The list of option choices
<i>desc</i>	Description used for info/tool tip

16.48.2.2 config_chooser() [2/2]

```
config_chooser::config_chooser (
    const char * name,
    const char * in_value,
    const char * in_choices,
    const char * desc = NULL) [inline]
```

Object constructor.

Parameters

<i>name</i>	Field name assigned to the value
<i>in_value</i>	The option value selected
<i>in_choices</i>	The list of option choices
<i>desc</i>	Description used for info/tool tip

16.48.3 Member Function Documentation**16.48.3.1 GetChoices()**

```
const config\_string & config_chooser::GetChoices () [inline]
```

Get the list of choices.

Returns

A [config_string](#) object containing the list of choices

16.48.3.2 GetTypeValue()

```
virtual void config_chooser::GetTypeValue (
    NBString & s) [inline], [override], [virtual]
```

Assigns the object type value to the specified [NBString](#) object.
The type value "object" is written to the [NBString](#) object.

Parameters

<i>s</i>	NBString object to hold the type value string

Reimplemented from [config_obj](#).

16.48.3.3 IsInChoices() [1/2]

```
bool config_chooser::IsInChoices (
    const char * str,
    size_t strLen) [inline]
```

Check if a string is in the list of possible choices. A comparison will continue until a null character is found, or the length of the longer of the two strings being compared is reached (strLen is used to determine if str is longer than the current element being compared.)

Parameters

<i>str</i>	NBString to attempt to find inside the list of choices
<i>strLen</i>	Length of str. This value is used to determine the amount of characters to compare for each element in the list of choices.

Return values

<i>true</i>	if str is found as an option in list of choices
<i>false</i>	if str is not found in the list of choices

16.48.3.4 IsInChoices() [2/2]

```
bool config_chooser::IsInChoices (
    const NBString & str,
    size_t strLen) [inline]
```

Check if a string is in the list of possible choices. A comparison will continue until a null character is found, or the length of the longer of the two strings being compared is reached (strLen is used to determine if str is longer than the current element being compared.)

Parameters

<i>str</i>	NBString to attempt to find inside the list of choices
<i>strLen</i>	Length of str. This value is used to determine the amount of characters to compare for each element in the list of choices.

Return values

<i>true</i>	if str is found as an option in list of choices
<i>false</i>	if str is not found in the list of choices

16.48.3.5 IsSelected() [1/2]

```
bool config_chooser::IsSelected (
    const char * choice) [inline]
```

Check if a particular choice option is selected.

Parameters

<i>choice</i>	Choice to test as a string type

Return values

<i>true</i>	Selected
<i>false</i>	Not Selected

16.48.3.6 IsSelected() [2/2]

```
bool config_chooser::IsSelected (
    const NBString & s) [inline]
```

Check if a particular choice option is selected.

Parameters

<i>s</i>	Choice to test as a NBString type

Return values

<i>true</i>	Selected
<i>false</i>	Not Selected

16.48.3.7 operator NSString()

```
config_chooser::operator NSString () const [inline]
```

Returns the object value.

Returns

The currently selected list option as a [NSString](#)

16.48.3.8 operator=() [1/3]

```
config_chooser & config_chooser::operator= (  
    const char * p) [inline]
```

Assign the selected list item from a const char* value.

Parameters

<i>p</i>	Selected list item to assign

16.48.3.9 operator=() [2/3]

```
config_chooser & config_chooser::operator= (  
    const config_chooser & ci) [inline]
```

Copy one [config_chooser](#) object to another.

Parameters

<i>ci</i>	Reference to a config_chooser object

16.48.3.10 operator=() [3/3]

```
config_chooser & config_chooser::operator= (  
    const NSString & s) [inline]
```

Assign the [config_string](#) object value from a [NSString](#) object.

Parameters

<i>s</i>	Reference to a NSString object

16.48.3.11 SetChoices()

```
const config_string & config_chooser::SetChoices (  
    const char * in_choices) [inline]
```

Set the list of choices.

Parameters

<i>in_choices</i>	The list of option choices

Returns

A [config_string](#) object containing the list of choices

The documentation for this class was generated from the following file:

- [config_obj.h](#)

16.49 [config_double](#) Class Reference

Double Float Configuration Variable.

`#include <config_obj.h>`

Inherits [config_value](#).

Inherited by [config_double_limit](#).

Public Member Functions

- virtual void [GetTextValue](#) ([NBString](#) &s) override
Copy the object value as a text string to the specified [NBString](#) object.
- [config_double](#) ([config_obj](#) &owner, double def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_double](#) (double def_val, const char *name, const char *desc=NULL)
Object constructor.
- [operator int](#) () const
Return the object value as an int.
- [operator float](#) () const
Return the object value as a float.
- [operator double](#) () const
Returns the object value as a double float.
- [config_double](#) & [operator=](#) (const double d)
Assign the [config_double](#) object value from a double value.
- [config_double](#) & [operator=](#) (const [config_double](#) &ci)
Copy one [config_double](#) object to another.
- virtual void [GetTypeValue](#) ([NBString](#) &s) override
Copy the object type value in the specified [NBString](#) object.

Additional Inherited Members**Protected Member Functions inherited from [config_value](#)**

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.49.1 Detailed Description

Double Float Configuration Variable.

Note

All modifications to configuration objects are marked as pending. A call to [SaveConfigToStorage](#) is required to save changes to flash memory.

16.49.2 Constructor & Destructor Documentation

16.49.2.1 config_double() [1/2]

```
config_double::config_double (
    config_obj & owner,
    double def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Parameters

<i>owner</i>	Reference to parent leaf
<i>def_val</i>	Default value
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.49.2.2 config_double() [2/2]

```
config_double::config_double (
    double def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor.

Parameters

<i>def_val</i>	Default value
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.49.3 Member Function Documentation

16.49.3.1 GetTextValue()

```
virtual void config_double::GetTextValue (
    NBString & s) [inline], [override], [virtual]
```

Copy the object value as a text string to the specified [NBString](#) object.

This is a virtual function that can be overridden in any derived class.

Parameters

<i>s</i>	Reference to a NBString object to store the result

Implements [config_value](#).

16.49.3.2 GetTypeValue()

```
virtual void config_double::GetTypeValue (
    NBString & s) [inline], [override], [virtual]
```

Copy the object type value in the specified [NBString](#) object.

The type value "string" is written to the [NBString](#) object, since browsers don't handle decimal numbers well with a "number" type unless the "step" attribute is also specified

Parameters

<code>s</code>	Reference to a NBString object

Reimplemented in [config_double_limit](#).

16.49.3.3 operator double()

```
config_double::operator double () const [inline]
```

Returns the object value as a double float.

Returns

The double float value of the object

Example: for a variable named `configInt` of type [config_double](#):

```
double value = double(configInt);
iprintf("int = %f\r\n", value);
```

16.49.3.4 operator float()

```
config_double::operator float () const [inline]
```

Return the object value as a float.

Returns

The float value of the object

Example: for a variable named `configFloat` of type [config_double](#):

```
float value = float(configFloat);
iprintf("int = %f\r\n", value);
```

16.49.3.5 operator int()

```
config_double::operator int () const [inline]
```

Return the object value as an int.

Returns

The integer value of the object

Example: for a variable named `configInt` of type [config_double](#):

```
int value = int(configInt);
iprintf("int = %d\r\n", value);
```

16.49.3.6 operator=() [1/2]

```
config_double & config_double::operator= (
    const config_double & ci) [inline]
```

Copy one [config_double](#) object to another.

Parameters

<code>ci</code>	Reference to a config_double object

16.49.3.7 operator=() [2/2]

```
config_double & config_double::operator= (
    const double d) [inline]
```

Assign the [config_double](#) object value from a double value.

Parameters

<i>d</i>	Value of type double

The documentation for this class was generated from the following file:

- config_obj.h

16.50 config_double_limit Class Reference

A [config_double](#) with minimum and/or maximum values Attempting to set a value lower than the minimum or greater than the maximum will fail with an error both programmatically and in the Config Web UI.

```
#include <config_obj.h>
```

Inherits [config_double](#).

Public Member Functions

- [config_double_limit](#) ([config_obj](#) &owner, double def_val, double minv=-DBL_MAX, double maxv=DBL_MAX, double stepv=0.01, const char *name=NULL, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter and limits.
- [config_double_limit](#) (double def_val, double minv=-DBL_MAX, double maxv=DBL_MAX, double stepv=0.01, const char *name=NULL, const char *desc=NULL)
Object constructor with limits.
- [config_double_limit](#) & [operator=](#) (const [config_double](#) &ci)
Copy a [config_double](#) object into the [config_double_limit](#).
- [config_double_limit](#) & [operator=](#) (const [config_double_limit](#) &cil)
Copy one [config_double_limit](#) object to another.
- virtual void [GetTypeValue](#) ([NBString](#) &s) override
Copy the object type value in the specified [NBString](#) object.

Public Member Functions inherited from [config_double](#)

- virtual void [GetTextValue](#) ([NBString](#) &s) override
Copy the object value as a text string to the specified [NBString](#) object.
- [config_double](#) ([config_obj](#) &owner, double def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_double](#) (double def_val, const char *name, const char *desc=NULL)
Object constructor.
- [operator int](#) () const
Return the object value as an int.
- [operator float](#) () const
Return the object value as a float.
- [operator double](#) () const
Returns the object value as a double float.
- [config_double](#) & [operator=](#) (const double d)
Assign the [config_double](#) object value from a double value.
- [config_double](#) & [operator=](#) (const [config_double](#) &ci)
Copy one [config_double](#) object to another.

Additional Inherited Members

Protected Member Functions inherited from [config_value](#)

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.50.1 Detailed Description

A [config_double](#) with minimum and/or maximum values Attempting to set a value lower than the minimum or greater than the maximum will fail with an error both programmatically and in the Config Web UI.

16.50.2 Constructor & Destructor Documentation

16.50.2.1 [config_double_limit\(\)](#) [1/2]

```
config_double_limit::config_double_limit (
    config\_obj & owner,
    double def_val,
    double minv = -DBL_MAX,
    double maxv = DBL_MAX,
    double stepv = 0.01,
    const char * name = NULL,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter and limits.

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value
<i>minv</i>	Minimum value
<i>maxv</i>	Maximum value
<i>stepv</i>	Step value (for HTML-side validation, not strictly enforced)
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.50.2.2 [config_double_limit\(\)](#) [2/2]

```
config_double_limit::config_double_limit (
    double def_val,
    double minv = -DBL_MAX,
    double maxv = DBL_MAX,
    double stepv = 0.01,
    const char * name = NULL,
    const char * desc = NULL) [inline]
```

Object constructor with limits.

Parameters

<i>def_val</i>	Default value
<i>minv</i>	Minimum value
<i>maxv</i>	Maximum value
<i>stepv</i>	Step value (for HTML-side validation, not strictly enforced)
<i>name</i>	Field name assigned to the value

<i>desc</i>	Description used for info/tool tip

16.50.3 Member Function Documentation

16.50.3.1 GetTypeValue()

```
virtual void config_double_limit::GetTypeValue (
    NBString & s) [inline], [override], [virtual]
```

Copy the object type value in the specified [NBString](#) object.
The type value "number" is written to the [NBString](#) object.

Parameters

<i>s</i>	Reference to a NBString object

Reimplemented from [config_double](#).

16.50.3.2 operator=() [1/2]

```
config_double_limit & config_double_limit::operator= (
    const config_double & ci) [inline]
```

Copy a [config_double](#) object into the [config_double_limit](#).

Parameters

<i>ci</i>	Reference to a config_double object

16.50.3.3 operator=() [2/2]

```
config_double_limit & config_double_limit::operator= (
    const config_double_limit & cil) [inline]
```

Copy one [config_double_limit](#) object to another.

Parameters

<i>cil</i>	Reference to a config_double_limit object

The documentation for this class was generated from the following file:

- [config_obj.h](#)

16.51 config_int Class Reference

Signed 32-bit Integer Configuration Variable.

#include <config_obj.h>

Inherits [config_value](#).

Inherited by [config_int_limit](#), and [version_obj](#).

Public Member Functions

- virtual void [GetTextValue](#) ([NBString](#) &s) override
Copy the object value to the specified [NBString](#) object.
- [config_int](#) ([config_obj](#) &owner, int def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.

- [config_int](#) (int def_val, const char *name, const char *desc=NULL)
Object constructor.
- [operator int](#) () const
Return the object value as an int.
- [config_int](#) & [operator=](#) (const int i)
Assign the [config_int](#) object value to the specified int value.
- [config_int](#) & [operator=](#) (const [config_int](#) &ci)
Copy one [config_int](#) object to another.
- virtual void [GetTypeValue](#) (NBString &s) override
Copy the object type value in the specified [NBString](#) object.

Additional Inherited Members

Protected Member Functions inherited from [config_value](#)

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.51.1 Detailed Description

Signed 32-bit Integer Configuration Variable.

Note

All modifications to configuration objects are marked as pending. A call to [SaveConfigToStorage](#) is required to save changes to flash memory.

16.51.2 Constructor & Destructor Documentation

16.51.2.1 [config_int](#)() [1/2]

```
config_int::config_int (
    config\_obj & owner,
    int def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.51.2.2 [config_int](#)() [2/2]

```
config_int::config_int (
    int def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor.

Parameters

<i>def_val</i>	Default value
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.51.3 Member Function Documentation**16.51.3.1 GetTextValue()**

```
virtual void config_int::GetTextValue (
    NBString & s) [inline], [override], [virtual]
```

Copy the object value to the specified [NBString](#) object.

This is a virtual function that can be overridden in any derived class.

Parameters

<i>s</i>	Reference to a NBString object to store the result

Implements [config_value](#).

16.51.3.2 GetTypeValue()

```
virtual void config_int::GetTypeValue (
    NBString & s) [inline], [override], [virtual]
```

Copy the object type value in the specified [NBString](#) object.

The type value "integer" is written to the [NBString](#) object.

Parameters

<i>s</i>	Reference to a NBString object

16.51.3.3 operator int()

```
config_int::operator int () const [inline]
```

Return the object value as an int.

Returns

The integer value of the object

Example: for a variable named configInt of type [config_int](#):

```
int value = int(configInt);
iprintf("int = %d\r\n", value);
```

16.51.3.4 operator=() [1/2]

```
config_int & config_int::operator= (
    const config_int & ci) [inline]
```

Copy one [config_int](#) object to another.

Parameters

<i>ci</i>	Reference to a config_int object

16.51.3.5 operator=() [2/2]

```
config_int & config_int::operator= (  
    const int i) [inline]
```

Assign the [config_int](#) object value to the specified int value.

Parameters

<i>i</i>	Integer value

The documentation for this class was generated from the following file:

- config_obj.h

16.52 config_int_limit Class Reference

A [config_int](#) with minimum and/or maximum values Attempting to set a value lower than the minimum or greater than the maximum will fail with an error both programmatically and in the Config Web UI.

```
#include <config_obj.h>
```

Inherits [config_int](#).

Public Member Functions

- [config_int_limit](#) ([config_obj](#) &owner, int def_val, int minv=INT_MIN, int maxv=INT_MAX, const char *name=NULL, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter and limits.
- [config_int_limit](#) (int def_val, int minv=INT_MIN, int maxv=INT_MAX, const char *name=NULL, const char *desc=NULL)
Object constructor with limits.
- [config_int_limit](#) & [operator=](#) (const [config_int](#) &ci)
Copy a [config_int](#) object into the [config_int_limit](#).
- [config_int_limit](#) & [operator=](#) (const [config_int_limit](#) &cil)
Copy one [config_int_limit](#) object to another.

Public Member Functions inherited from [config_int](#)

- virtual void [GetTextValue](#) (NSString &s) override
Copy the object value to the specified [NSString](#) object.
- [config_int](#) ([config_obj](#) &owner, int def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_int](#) (int def_val, const char *name, const char *desc=NULL)
Object constructor.
- [operator int](#) () const
Return the object value as an int.
- [config_int](#) & [operator=](#) (const int i)
Assign the [config_int](#) object value to the specified int value.
- [config_int](#) & [operator=](#) (const [config_int](#) &ci)
Copy one [config_int](#) object to another.
- virtual void [GetTypeValue](#) (NSString &s) override
Copy the object type value in the specified [NSString](#) object.

Additional Inherited Members

Protected Member Functions inherited from [config_value](#)

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.52.1 Detailed Description

A [config_int](#) with minimum and/or maximum values Attempting to set a value lower than the minimum or greater than the maximum will fail with an error both programmatically and in the Config Web UI.

16.52.2 Constructor & Destructor Documentation

16.52.2.1 [config_int_limit\(\)](#) [1/2]

```
config_int_limit::config_int_limit (
    config\_obj & owner,
    int def_val,
    int minv = INT_MIN,
    int maxv = INT_MAX,
    const char * name = NULL,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter and limits.

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value
<i>minv</i>	Minimum value
<i>maxv</i>	Maximum value
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.52.2.2 [config_int_limit\(\)](#) [2/2]

```
config_int_limit::config_int_limit (
    int def_val,
    int minv = INT_MIN,
    int maxv = INT_MAX,
    const char * name = NULL,
    const char * desc = NULL) [inline]
```

Object constructor with limits.

Parameters

<i>def_val</i>	Default value
<i>minv</i>	Minimum value
<i>maxv</i>	Maximum value
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.52.3 Member Function Documentation

16.52.3.1 [operator=\(\)](#) [1/2]

```
config\_int\_limit & config\_int\_limit::operator= (
    const config\_int & ci) [inline]
```

Copy a [config_int](#) object into the [config_int_limit](#).

Parameters

<i>ci</i>	Reference to a config_int object

16.52.3.2 operator=() [2/2]

```
config_int_limit & config_int_limit::operator= (
    const config_int_limit & cil) [inline]
```

Copy one [config_int_limit](#) object to another.

Parameters

<i>cil</i>	Reference to a config_int_limit object

The documentation for this class was generated from the following file:

- [config_obj.h](#)

16.53 config_IPADDR Class Reference

Configuration Variable for IPADDR (IPv6) object type.

```
#include <config_obj.h>
```

Inherits [config_value](#).

Public Member Functions

- virtual void [GetTextValue](#) ([NBString](#) &s) override
Get the object value as a text string with quotations to the specified [NBString](#) object.
- [config_IPADDR](#) ([config_obj](#) &owner, [IPADDR](#) def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_IPADDR](#) ([IPADDR](#) def_val, const char *name, const char *desc=NULL)
Object constructor.
- [config_IPADDR](#) ([config_obj](#) &owner, const char *def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_IPADDR](#) (const char *def_val, const char *name, const char *desc=NULL)
Object constructor.
- [operator IPADDR](#) () const
Returns the object value.
- bool [IsNull](#) () const
Check if the IP address is null.
- bool [NotNull](#) () const
Check if the IP address is not null.
- void [SetNull](#) ()
Set the IP address value of an [config_IPADDR](#) object to null.
- [config_IPADDR](#) & [operator=](#) (const [IPADDR](#) &i6)
Copy an IPADDR object value to a [config_IPADDR](#) object.
- [config_IPADDR](#) & [operator=](#) (const [config_IPADDR](#) &ci)
Copy one [config_IPADDR](#) object to another.
- virtual void [GetTypeValue](#) ([NBString](#) &s) override
Copy the object type value to the specified [NBString](#) object.

Additional Inherited Members

Protected Member Functions inherited from [config_value](#)

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.53.1 Detailed Description

Configuration Variable for IPADDR (IPv6) object type.

See also

[config_IPADDR4](#)

Note

All modifications to configuration objects are marked as pending. A call to [SaveConfigToStorage](#) is required to save changes to flash memory.

16.53.2 Constructor & Destructor Documentation

16.53.2.1 config_IPADDR() [1/4]

```
config_IPADDR::config_IPADDR (
    config_obj & owner,
    IPADDR def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.
Default value assigned with an IPADDR object

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value of type IPADDR
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.53.2.2 config_IPADDR() [2/4]

```
config_IPADDR::config_IPADDR (
    IPADDR def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor.
Default value assigned with an IPADDR object

Parameters

<i>def_val</i>	Default value of type NBString
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.53.2.3 config_IPADDR() [3/4]

```
config_IPADDR::config_IPADDR (
    config_obj & owner,
    const char * def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.
Default value assigned with an const char * string. IP address is converted using the SetFromAscii() IPADDR member function.

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value of type char * in IPv6 format
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.53.2.4 config_IPADDR() [4/4]

```
config_IPADDR::config_IPADDR (
    const char * def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor.

Default value assigned with an const char * string. IP address is converted using the SetFromAscii() IPADDR member function.

Parameters

<i>def_val</i>	Default value of type char * in IPv6 format
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.53.3 Member Function Documentation**16.53.3.1 GetTextValue()**

```
virtual void config_IPADDR::GetTextValue (
    NBString & s) [inline], [override], [virtual]
```

Get the object value as a text string with quotations to the specified [NBString](#) object.

This is a virtual function that can be overridden in any derived class.

Parameters

<i>s</i>	Reference to a NBString object to store the result

Implements [config_value](#).

16.53.3.2 GetTypeValue()

```
virtual void config_IPADDR::GetTypeValue (
    NBString & s) [inline], [override], [virtual]
```

Copy the object type value to the specified [NBString](#) object.

The type value "string" is written to the [NBString](#) object.

Parameters

<i>s</i>	NBString object to hold the type value string

16.53.3.3 IsNull()

```
bool config_IPADDR::IsNull () const [inline]
```

Check if the IP address is null.

Return values

<i>true</i>	if the IP address value is null

See also

[NotNull\(\)](#)**16.53.3.4 NotNull()**

```
bool config_IPADDR::NotNull () const [inline]
```

Check if the IP address is not null.

Return values

<i>true</i>	if the IP address value is not null

See also

[IsNull\(\)](#)**16.53.3.5 operator IPADDR()**

```
config_IPADDR::operator IPADDR () const [inline]
```

Returns the object value.

Returns

The value of the object as an IPADDR object

16.53.3.6 operator=() [1/2]

```
config_IPADDR & config_IPADDR::operator= (  
    const config_IPADDR & ci) [inline]
```

Copy one [config_IPADDR](#) object to another.

Parameters

<i>ci</i>	Reference to a config_IPADDR object

16.53.3.7 operator=() [2/2]

```
config_IPADDR & config_IPADDR::operator= (  
    const IPADDR & i6) [inline]
```

Copy an IPADDR object value to a [config_IPADDR](#) object.

Parameters

<i>i6</i>	Reference to an IPADDR object

The documentation for this class was generated from the following file:

- [config_obj.h](#)

16.54 config_IPADDR4 Class Reference

Configuration Variable for [IPADDR4](#) (IPv4) object types.

#include <config_obj.h>

Inherits [config_value](#).

Public Member Functions

- virtual void [GetTextValue](#) (NBString &s) override
Get the object value as a text string with quotations to the specified NBString object.
- [config_IPADDR4](#) (config_obj &owner, IPADDR4 def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_IPADDR4](#) (IPADDR4 def_val, const char *name, const char *desc=NULL)
Object constructor.
- [config_IPADDR4](#) (config_obj &owner, const char *def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_IPADDR4](#) (const char *def_val, const char *name, const char *desc=NULL)
Object constructor.
- [operator IPADDR4](#) () const
Returns the object value.
- bool [IsNull](#) () const
Check if the IP address is null.
- bool [NotNull](#) () const
Check if the IP address is not null.
- void [SetNull](#) ()
Set the IP address to null.
- [config_IPADDR4](#) & [operator=](#) (const IPADDR4 &i4)
Copy an IPADDR4 object value to a config_IPADDR4 object.
- [config_IPADDR4](#) & [operator=](#) (const config_IPADDR4 &ci)
Copy one config_IPADDR4 object to another.
- virtual void [GetTypeValue](#) (NBString &s) override
Copy the object type value to the specified NBString object.

Additional Inherited Members

Protected Member Functions inherited from [config_value](#)

- [config_value](#) (config_obj &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.54.1 Detailed Description

Configuration Variable for [IPADDR4](#) (IPv4) object types.

See also

[config_IPADDR](#)

Note

All modifications to configuration objects are marked as pending. A call to [SaveConfigToStorage](#) is required to save changes to flash memory.

16.54.2 Constructor & Destructor Documentation

16.54.2.1 config_IPADDR4() [1/4]

```
config_IPADDR4::config_IPADDR4 (  
    config_obj & owner,  
    IPADDR4 def_val,  
    const char * name,  
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Default value assigned with an [IPADDR4](#) object

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value of type IPADDR4
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.54.2.2 config_IPADDR4() [2/4]

```
config_IPADDR4::config_IPADDR4 (  
    IPADDR4 def_val,  
    const char * name,  
    const char * desc = NULL) [inline]
```

Object constructor.

Default value assigned with an [IPADDR4](#) object

Parameters

<i>def_val</i>	Default value of type NBString
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.54.2.3 config_IPADDR4() [3/4]

```
config_IPADDR4::config_IPADDR4 (  
    config_obj & owner,  
    const char * def_val,  
    const char * name,  
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Default value assigned with an const char * string. IP address is converted using the [IPADDR4](#) SetFromAscii() member function.

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value of type char *. Format is: "xxx.xxx.xxx.xxx". For example, "10.1.1.100"
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.54.2.4 config_IPADDR4() [4/4]

```
config_IPADDR4::config_IPADDR4 (
    const char * def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor.

Default value assigned with an const char * string. IP address is converted using the [IPADDR4](#) SetFromAscii() member function.

Parameters

<i>def_val</i>	Default value of type char *. Format is: "xxx.xxx.xxx.xxx". For example, "10.1.1.100"
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.54.3 Member Function Documentation**16.54.3.1 GetTextValue()**

```
virtual void config_IPADDR4::GetTextValue (
    NBString & s) [inline], [override], [virtual]
```

Get the object value as a text string with quotations to the specified [NBString](#) object.

The format of the string is: "xxx.xxx.xxx.xxx". This is a virtual function that can be overridden in any derived class.

Parameters

<i>s</i>	Reference to a NBString object to store the result

Implements [config_value](#).

16.54.3.2 GetTypeValue()

```
virtual void config_IPADDR4::GetTypeValue (
    NBString & s) [inline], [override], [virtual]
```

Copy the object type value to the specified [NBString](#) object.

The type value "string" is written to the [NBString](#) object.

Parameters

<i>s</i>	NBString object to hold the type value string

16.54.3.3 IsNull()

```
bool config_IPADDR4::IsNull () const [inline]
```

Check if the IP address is null.

Return values

<i>true</i>	if the IP address value is null

See also

[NotNull\(\)](#), [SetNull\(\)](#)

16.54.3.4 NotNull()

```
bool config_IPADDR4::NotNull () const [inline]
```

Check if the IP address is not null.

Return values

<code>true</code>	if the IP address value is not null

See also

[IsNull\(\)](#), [SetNull\(\)](#)

16.54.3.5 operator IPADDR4()

```
config_IPADDR4::operator IPADDR4 () const [inline]
```

Returns the object value.

Returns

The value of the object as an [IPADDR4](#)

16.54.3.6 operator=() [1/2]

```
config_IPADDR4 & config_IPADDR4::operator= (
    const config_IPADDR4 & ci) [inline]
```

Copy one [config_IPADDR4](#) object to another.

Parameters

<code>ci</code>	Reference to a config_IPADDR4 object

16.54.3.7 operator=() [2/2]

```
config_IPADDR4 & config_IPADDR4::operator= (
    const IPADDR4 & i4) [inline]
```

Copy an [IPADDR4](#) object value to a [config_IPADDR4](#) object.

Parameters

<code>i4</code>	Reference to an IPADDR4 object

16.54.3.8 SetNull()

```
void config_IPADDR4::SetNull () [inline]
```

Set the IP address to null.

See also

[IsNull\(\)](#), [NotNull\(\)](#)

The documentation for this class was generated from the following file:

- `config_obj.h`

16.55 config_localname Class Reference

Configure device network name.

```
#include <config_netobj.h>
```

Inherits [config_string](#).

Public Member Functions

- [config_localname](#) ([config_obj](#) &owner, [NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_localname](#) ([NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor.
- [config_localname](#) ([config_obj](#) &owner, const char *def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_localname](#) (const char *def_val, const char *name, const char *desc=NULL)
Object constructor.
- [operator NBString](#) () const
Returns the object value.
- [config_localname](#) & [operator=](#) (const char *p)
Assign the [config_localname](#) object value from a const char value.*
- [config_localname](#) & [operator=](#) (const [NBString](#) &s)
Assign the [config_localname](#) object value from a [NBString](#) object.
- [config_localname](#) & [operator=](#) (const [config_string](#) &ci)
Copy a [config_string](#) object to a [config_localname](#) object.
- [config_localname](#) & [operator=](#) (const [config_localname](#) &ci)
Copy one [config_localname](#) object to another.

Public Member Functions inherited from [config_string](#)

- virtual void [GetTextValue](#) ([NBString](#) &s) override
Get the object value (as a text string with quotations) to the specified [NBString](#) object.
- [config_string](#) ([config_obj](#) &owner, [NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_string](#) ([NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor.
- [config_string](#) ([config_obj](#) &owner, const char *def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_string](#) (const char *def_val, const char *name, const char *desc=NULL)
Object constructor.
- void [SetEnumList](#) ([NBString](#) s)
Renders the data used to explain the schema/descriptions for the list of choices.
- [operator NBString](#) () const
Return the object value.
- [config_string](#) & [operator=](#) (const char *p)
Assign the [config_string](#) object value from a const char string.*
- [config_string](#) & [operator=](#) (const [NBString](#) &s)
Assign the [config_string](#) object value from a [NBString](#) object.
- [config_string](#) & [operator=](#) (const [config_string](#) &ci)
Copy one [config_string](#) object to another.
- const char * [c_str](#) () const
Returns the object value as a string.
- size_t [length](#) () const
Returns the string length in bytes.
- const char & [operator\[\]](#) (size_t pos) const
Return the value of a character in the string.
- virtual void [GetTypeValue](#) ([NBString](#) &s) override
Copy the object type value to the specified [NBString](#) object.

Additional Inherited Members

Protected Member Functions inherited from [config_value](#)

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.55.1 Detailed Description

Configure device network name.

16.55.2 Constructor & Destructor Documentation

16.55.2.1 config_localname() [1/4]

```
config_localname::config_localname (  
    config\_obj & owner,  
    NBString def_val,  
    const char * name,  
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.
Default value is assigned with a [NBString](#)

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value of type NBString
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.55.2.2 config_localname() [2/4]

```
config_localname::config_localname (  
    NBString def_val,  
    const char * name,  
    const char * desc = NULL) [inline]
```

Object constructor.
Default value is assigned with a [NBString](#)

Parameters

<i>def_val</i>	Default value of type NBString
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.55.2.3 config_localname() [3/4]

```
config_localname::config_localname (  
    config\_obj & owner,  
    const char * def_val,  
    const char * name,  
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.
Default value is assigned with a character string

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value of type char *
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.55.2.4 config_localname() [4/4]

```
config_localname::config_localname (  
    const char * def_val,  
    const char * name,  
    const char * desc = NULL) [inline]
```

Object constructor.

Default value is assigned with a character string

Parameters

<i>def_val</i>	Default value of type char *
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.55.3 Member Function Documentation**16.55.3.1 operator NBString()**

```
config_localname::operator NBString () const [inline]
```

Returns the object value.

Returns

The value of the object as a [NBString](#)

16.55.3.2 operator=() [1/4]

```
config_localname & config_localname::operator= (  
    const char * p) [inline]
```

Assign the [config_localname](#) object value from a const char* value.

Parameters

<i>p</i>	String to assign

16.55.3.3 operator=() [2/4]

```
config_localname & config_localname::operator= (  
    const config_localname & ci) [inline]
```

Copy one [config_localname](#) object to another.

Parameters

<i>ci</i>	Reference to a config_localname object

16.55.3.4 operator=() [3/4]

```
config_localname & config_localname::operator= (
    const config_string & ci) [inline]
```

Copy a [config_string](#) object to a [config_localname](#) object.

Parameters

<i>ci</i>	Reference to a config_string object

16.55.3.5 operator=() [4/4]

```
config_localname & config_localname::operator= (
    const NBString & s) [inline]
```

Assign the [config_localname](#) object value from a [NBString](#) object.

Parameters

<i>s</i>	Reference to a NBString object

The documentation for this class was generated from the following file:

- [config_netobj.h](#)

16.56 config_MACADR Class Reference

Configuration Variable for [MACADR](#) object type.

```
#include <config_obj.h>
```

Inherits [config_value](#).

Public Member Functions

- virtual void [GetTextValue](#) ([NBString](#) &s) override
Get the object value as a text string with quotations to the specified [NBString](#) object.
- [config_MACADR](#) ([config_obj](#) &owner, [MACADR](#) def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_MACADR](#) ([MACADR](#) def_val, const char *name, const char *desc=NULL)
Object constructor.
- [config_MACADR](#) ([config_obj](#) &owner, const char *def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_MACADR](#) (const char *def_val, const char *name, const char *desc=NULL)
Object constructor.
- [operator](#) [MACADR](#) () const
Returns the object value.
- [config_MACADR](#) & [operator](#)= (const [config_MACADR](#) &ci)
Copy one [config_MACADR](#) object to another.
- [config_MACADR](#) & [operator](#)= (const [MACADR](#) &ci)
Copy a [MACADR](#) object value to a [MACADR](#) object.
- virtual void [GetTypeValue](#) ([NBString](#) &s) override
Copy the object type value to the specified [NBString](#) object.

Additional Inherited Members

Protected Member Functions inherited from [config_value](#)

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.56.1 Detailed Description

Configuration Variable for [MACADR](#) object type.

Note

All modifications to configuration objects are marked as pending. A call to [SaveConfigToStorage](#) is required to save changes to flash memory.

16.56.2 Constructor & Destructor Documentation

16.56.2.1 [config_MACADR\(\)](#) [1/4]

```
config_MACADR::config_MACADR (
    config\_obj & owner,
    MACADR def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Default value assigned with an [MACADR](#) object

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value of type MACADR
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.56.2.2 [config_MACADR\(\)](#) [2/4]

```
config_MACADR::config_MACADR (
    MACADR def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor.

Default value assigned with an [MACADR](#) object

Parameters

<i>def_val</i>	Default value of type MACADR
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.56.2.3 config_MACADR() [3/4]

```
config_MACADR::config_MACADR (
    config_obj & owner,
    const char * def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Default value is assigned with a const char * string. The [AsciiToMac\(\)](#) function is used to convert the string to a [MACADR](#).

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default MAC Address value of type char *. The hexidical format must be 6 octets, with or without ':' separators. For example, "0123456789AB", or "01:23:45:67:89:AB"
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.56.2.4 config_MACADR() [4/4]

```
config_MACADR::config_MACADR (
    const char * def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor.

Default value is assigned with a const char * string. The [AsciiToMac\(\)](#) function is used to convert the string to a [MACADR](#).

Parameters

<i>def_val</i>	Default MAC Address value of type char *. The hexidical format must be 6 octets, with or without ':' separators. For example, "0123456789AB", or "01:23:45:67:89:AB"
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.56.3 Member Function Documentation**16.56.3.1 GetTextValue()**

```
virtual void config_MACADR::GetTextValue (
    NBString & s) [inline], [override], [virtual]
```

Get the object value as a text string with quotations to the specified [NBString](#) object.

The format is the hexadecimal address: "xx:xx:xx:xx:xx:xx" This is a virtual function that can be overridden in any derived class.

Parameters

<i>s</i>	Reference to a NBString object to store the result

Implements [config_value](#).

16.56.3.2 GetTypeValue()

```
virtual void config_MACADR::GetTypeValue (
    NBString & s) [inline], [override], [virtual]
```

Copy the object type value to the specified [NSString](#) object.
The type value "string" is written to the [NSString](#) object.

Parameters

s	NBString object to hold the type value string

16.56.3.3 operator MACADR()

```
config_MACADR::operator MACADR () const [inline]
```

Returns the object value.

Returns

The value of the object as a [MACADR](#)

16.56.3.4 operator=() [1/2]

```
config_MACADR & config_MACADR::operator= (
    const config_MACADR & ci) [inline]
```

Copy one [config_MACADR](#) object to another.

Parameters

ci	Reference to a config_IPADDR object

16.56.3.5 operator=() [2/2]

```
config_MACADR & config_MACADR::operator= (
    const MACADR & ci) [inline]
```

Copy a [MACADR](#) object value to a [MACADR](#) object.

Parameters

ci	Reference to an MACADR object

The documentation for this class was generated from the following file:

- [config_obj.h](#)

16.57 config_obj Class Reference

Base class used to create configuration objects.

```
#include <config_obj.h>
```

Inherits [config_leaf](#).

Inherited by [CertData](#), [ControlSet](#), [EmailNotify](#), [I4Record](#), [I6Record](#), [InterfaceBlock](#), [LLDPEntity](#), [MonitorRecord](#), [NV_OneI2CPortSetting](#), [NV_OnePortSetting](#), [NV_SettingsStruct](#), [OneConfigRecord](#), [PIDSet](#), [SysRecord](#), [Temp↔Range](#), [TempRange](#), [TempRange](#), [TempRange](#), [ThermoGroup](#), [Thermostat1](#), [WebUIConfig](#), [cfg_class_slot](#), [cfg_↔class_slots](#), [classSerialSettings](#), [classSerialSettings](#), [config_MqttClient](#), [config_MqttLWT](#), [config_chooser](#), [config_↔preserver_obj](#), [config_uart](#), [config_x509Cert](#), [detached_root_obj](#), [empty_config_obj](#), and [root_obj](#).

Public Member Functions

- [config_obj](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_obj](#) (const char *name, const char *desc)
Object constructor.
- virtual void [GetTextValue](#) ([NBString](#) &s) override
Get the object value as a text string to the specified [NBString](#) object.
- virtual void [GetTypeValue](#) ([NBString](#) &s) override
Assigns the object type value to the specified [NBString](#) object.

16.57.1 Detailed Description

Base class used to create configuration objects.

Base that can be used to create your own custom configuration objects in a JSON compatible format. The object can hold any number of JSON compatible types: integer, boolean, null, string, object or array.

A [config_obj](#) can hold multiple values in a JSON object. A [config_value](#) holds only a single JSON value.

See also

[config_value](#)

The example below demonstrates how to create a thermostat object with a temperature range derived from a [config_obj](#):

Note

Please check the Configuration Class example in the `\nburn\examples` folder for the latest code updates.

The use of config objects should be done at a global scope. They can contain any number of members, which should be given a [default](#) value and a name used as an identifier. The name value [for each](#) member variable should be unique, otherwise it can lead to issues inside the config tree.

```
class TempRange : public config_obj
{
public:
    // Class variables
    config_int m_maxTemp{100, "MaxTemp"};
    config_int m_setTemp{30, "SetTemp"};
    config_int m_minTemp{10, "MinTemp"};
    ConfigEndMarker; // No new data members below this line

    // Class constructors
    TempRange(const char *name, const char *desc = nullptr) : config_obj(name, desc){};
    TempRange(config_obj &owner, const char *name, const char *desc = nullptr) : config_obj(owner, name,
        desc){};
};

class Thermostat : public TempRange
{
public:
    config_int m_tempRec1{101, "TempRecord_1"};
    config_int m_tempRec2{102, "TempRecord_2"};
    config_bool m_active{true, "Active"};
    config_string m_location{"Warehouse 1", "Thermostat Location"};
    config_chooser m_tempScale{"Temperature Scale", "Fahrenheit", "Fahrenheit", "Celsius", "Kelvin"};
    ConfigEndMarker; // No new data members below this line

    Thermostat(const char *name, const char *desc = nullptr) : TempRange(name, desc){};
    Thermostat(config_obj &owner, const char *name, const char *desc = nullptr) : TempRange(owner, name,
        desc){};
};

static Thermostat thermo(appdata, "Thermostat", "The primary thermostat");
```

Note

All modifications to configuration objects are marked as pending. A call to [SaveConfigToStorage](#) is required to save changes to flash memory.

16.57.2 Constructor & Destructor Documentation

16.57.2.1 config_obj() [1/2]

```
config_obj::config_obj (
    config_obj & owner,
    const char * name,
    const char * desc) [inline]
```

Object constructor with the parent/owner leaf parameter.

For example,

```
myObject(config_obj &owner, const char *name, const char *desc = nullptr) : config_obj(owner, name, desc){};
```

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>name</i>	Field name

<i>desc</i>	Description used for info/tool tip

16.57.2.2 config_obj() [2/2]

```
config_obj::config_obj (
    const char * name,
    const char * desc) [inline]
```

Object constructor.

This constructor should be called as part of the derived class constructor. For example,
`myObject(const char *name, const char *desc = nullptr) : config_obj(name, desc){}`

Parameters

<i>name</i>	Field name
<i>desc</i>	Description used for info/tool tip

16.57.3 Member Function Documentation

16.57.3.1 GetTextValue()

```
virtual void config_obj::GetTextValue (
    NBString & s) [override], [virtual]
```

Get the object value as a text string to the specified [NBString](#) object.
 This is a virtual function that can be overridden in any derived class.

Parameters

<i>s</i>	Reference to a NBString object to store the result

16.57.3.2 GetTypeValue()

```
virtual void config_obj::GetTypeValue (
    NBString & s) [inline], [override], [virtual]
```

Assigns the object type value to the specified [NBString](#) object.
 The type value "object" is written to the [NBString](#) object.

Parameters

<i>s</i>	NBString object to hold the type value string

Reimplemented in [config_chooser](#).

The documentation for this class was generated from the following file:

- `config_obj.h`

16.58 config_pass Class Reference

Password string Configuration Variable.

`#include <config_obj.h>`

Inherits [config_string](#).

Inherited by [config_pass_limit](#).

Public Member Functions

- [config_pass](#) ([config_obj](#) &owner, [NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_pass](#) ([NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor.
- [config_pass](#) ([config_obj](#) &owner, const char *def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_pass](#) (const char *def_val, const char *name, const char *desc=NULL)
Object constructor.
- virtual void [GetTextValue](#) ([NBString](#) &s) override
Get the [config_pass](#) object value as a text string.
- virtual void [GetRawValue](#) ([NBString](#) &s) override
Copy the raw [config_string](#) object value to the [NBString](#) object.
- [operator NBString](#) () const
Returns the object value.
- [config_pass](#) & [operator=](#) (const char *p)
Assign the [config_pass](#) object value from a const char value.*
- [config_pass](#) & [operator=](#) (const [NBString](#) &s)
Assign the [config_pass](#) object value from a [NBString](#) object.
- [config_pass](#) & [operator=](#) (const [config_string](#) &ci)
Copy a [config_string](#) object to a [config_pass](#) object.
- [config_pass](#) & [operator=](#) (const [config_pass](#) &ci)
Copy one [config_pass](#) object to another.

Public Member Functions inherited from [config_string](#)

- [config_string](#) ([config_obj](#) &owner, [NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_string](#) ([NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor.
- [config_string](#) ([config_obj](#) &owner, const char *def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_string](#) (const char *def_val, const char *name, const char *desc=NULL)
Object constructor.
- void [SetEnumList](#) ([NBString](#) s)
Renders the data used to explain the schema/descriptions for the list of choices.
- [operator NBString](#) () const
Return the object value.
- [config_string](#) & [operator=](#) (const char *p)
Assign the [config_string](#) object value from a const char string.*
- [config_string](#) & [operator=](#) (const [NBString](#) &s)
Assign the [config_string](#) object value from a [NBString](#) object.
- [config_string](#) & [operator=](#) (const [config_string](#) &ci)
Copy one [config_string](#) object to another.
- const char * [c_str](#) () const
Returns the object value as a string.
- size_t [length](#) () const
Returns the string length in bytes.
- const char & [operator\[\]](#) (size_t pos) const
Return the value of a character in the string.
- virtual void [GetTypeValue](#) ([NBString](#) &s) override
Copy the object type value to the specified [NBString](#) object.

Additional Inherited Members

Protected Member Functions inherited from [config_value](#)

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.58.1 Detailed Description

Password string Configuration Variable.

A [config_pass](#) is like a [config_string](#), but the data is hidden by '*' characters.

Note

All modifications to configuration objects are marked as pending. A call to [SaveConfigToStorage](#) is required to save changes to flash memory.

16.58.2 Constructor & Destructor Documentation

16.58.2.1 config_pass() [1/4]

```
config_pass::config_pass (
    config\_obj & owner,
    NBString def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Default value is assigned with a [NBString](#)

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value of type NBString
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.58.2.2 config_pass() [2/4]

```
config_pass::config_pass (
    NBString def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor.

Default value is assigned with a [NBString](#)

Parameters

<i>def_val</i>	Default value of type NBString
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.58.2.3 config_pass() [3/4]

```
config_pass::config_pass (
    config_obj & owner,
    const char * def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Default value is assigned with a character string

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value of type char *
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.58.2.4 config_pass() [4/4]

```
config_pass::config_pass (
    const char * def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor.

Default value is assigned with a character string

Parameters

<i>def_val</i>	Default value of type char *
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.58.3 Member Function Documentation

16.58.3.1 GetRawValue()

```
virtual void config_pass::GetRawValue (
    NBString & s) [override], [virtual]
```

Copy the raw [config_string](#) object value to the [NBString](#) object.

Parameters

<i>s</i>	Reference to a NBString object

This is a virtual function that can be overridden in any derived class.

16.58.3.2 GetTextValue()

```
virtual void config_pass::GetTextValue (
    NBString & s) [override], [virtual]
```

Get the [config_pass](#) object value as a text string.

This is a virtual function that can be overridden in any derived class.

Parameters

<i>s</i>	Reference to a NBString object to store the result

Reimplemented from [config_string](#).

16.58.3.3 operator NSString()

```
config_pass::operator NSString () const [inline]
```

Returns the object value.

Returns

The value of the object as a [NSString](#)

16.58.3.4 operator=() [1/4]

```
config_pass & config_pass::operator= (  
    const char * p) [inline]
```

Assign the [config_pass](#) object value from a const char* value.

Parameters

<i>p</i>	String to assign

16.58.3.5 operator=() [2/4]

```
config_pass & config_pass::operator= (  
    const config_pass & ci) [inline]
```

Copy one [config_pass](#) object to another.

Parameters

<i>ci</i>	Reference to a config_pass object

16.58.3.6 operator=() [3/4]

```
config_pass & config_pass::operator= (  
    const config_string & ci) [inline]
```

Copy a [config_string](#) object to a [config_pass](#) object.

Parameters

<i>ci</i>	Reference to a config_string object

16.58.3.7 operator=() [4/4]

```
config_pass & config_pass::operator= (  
    const NSString & s) [inline]
```

Assign the [config_pass](#) object value from a [NSString](#) object.

Parameters

<i>s</i>	Reference to a NSString object

The documentation for this class was generated from the following file:

- [config_obj.h](#)

16.59 config_pass_limit Class Reference

A [config_pass](#) with minimum and/or maximum lengths Attempting to set a value shorter than the minimum or longer than the maximum will fail with an error both programmatically and in the Config Web UI.

```
#include <config_obj.h>
```

Inherits [config_pass](#).

Public Member Functions

- [config_pass_limit](#) ([config_obj](#) &owner, [NBString](#) def_val, size_t minl=0, size_t maxl=0, const char *name=NULL, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter and limits.
- [config_pass_limit](#) ([NBString](#) def_val, size_t minl=0, size_t maxl=0, const char *name=NULL, const char *desc=NULL)
Object constructor with limits.
- [config_pass_limit](#) & operator= (const [config_pass](#) &ci)
Copy a [config_pass](#) object into the [config_pass_limit](#).
- [config_pass_limit](#) & operator= (const [config_pass_limit](#) &cil)
Copy one [config_pass_limit](#) object to another.

Public Member Functions inherited from [config_pass](#)

- [config_pass](#) ([config_obj](#) &owner, [NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_pass](#) ([NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor.
- [config_pass](#) ([config_obj](#) &owner, const char *def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_pass](#) (const char *def_val, const char *name, const char *desc=NULL)
Object constructor.
- virtual void [GetTextValue](#) ([NBString](#) &s) override
Get the [config_pass](#) object value as a text string.
- virtual void [GetRawValue](#) ([NBString](#) &s) override
Copy the raw [config_string](#) object value to the [NBString](#) object.
- operator [NBString](#) () const
Returns the object value.
- [config_pass](#) & operator= (const char *p)
Assign the [config_pass](#) object value from a const char value.*
- [config_pass](#) & operator= (const [NBString](#) &s)
Assign the [config_pass](#) object value from a [NBString](#) object.
- [config_pass](#) & operator= (const [config_string](#) &ci)
Copy a [config_string](#) object to a [config_pass](#) object.
- [config_pass](#) & operator= (const [config_pass](#) &ci)
Copy one [config_pass](#) object to another.

Public Member Functions inherited from [config_string](#)

- [config_string](#) ([config_obj](#) &owner, [NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_string](#) ([NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor.
- [config_string](#) ([config_obj](#) &owner, const char *def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.

- [config_string](#) (const char *def_val, const char *name, const char *desc=NULL)
Object constructor.
- void [SetEnumList](#) (NBString s)
Renders the data used to explain the schema/descriptions for the list of choices.
- [operator NBString](#) () const
Return the object value.
- [config_string](#) & [operator=](#) (const char *p)
Assign the [config_string](#) object value from a const char string.*
- [config_string](#) & [operator=](#) (const NBString &s)
Assign the [config_string](#) object value from a NBString object.
- [config_string](#) & [operator=](#) (const [config_string](#) &ci)
Copy one [config_string](#) object to another.
- const char * [c_str](#) () const
Returns the object value as a string.
- size_t [length](#) () const
Returns the string length in bytes.
- const char & [operator\[\]](#) (size_t pos) const
Return the value of a character in the string.
- virtual void [GetTypeValue](#) (NBString &s) override
Copy the object type value to the specified NBString object.

Additional Inherited Members

Protected Member Functions inherited from [config_value](#)

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.59.1 Detailed Description

A [config_pass](#) with minimum and/or maximum lengths Attempting to set a value shorter than the minimum or longer than the maximum will fail with an error both programmatically and in the Config Web UI.

16.59.2 Constructor & Destructor Documentation

16.59.2.1 config_pass_limit() [1/2]

```
config_pass_limit::config_pass_limit (
    config\_obj & owner,
    NBString def_val,
    size_t minl = 0,
    size_t maxl = 0,
    const char * name = NULL,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter and limits.

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value
<i>minl</i>	Minimum length
<i>maxl</i>	Maximum length
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.59.2.2 config_pass_limit() [2/2]

```
config_pass_limit::config_pass_limit (
    NBString def_val,
    size_t minl = 0,
    size_t maxl = 0,
    const char * name = NULL,
    const char * desc = NULL) [inline]
```

Object constructor with limits.

Parameters

<i>def_val</i>	Default value
<i>minl</i>	Minimum length
<i>maxl</i>	Maximum length
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.59.3 Member Function Documentation

16.59.3.1 operator=() [1/2]

```
config_pass_limit & config_pass_limit::operator= (
    const config_pass & ci) [inline]
```

Copy a [config_pass](#) object into the [config_pass_limit](#).

Parameters

<i>ci</i>	Reference to a config_pass object

16.59.3.2 operator=() [2/2]

```
config_pass_limit & config_pass_limit::operator= (
    const config_pass_limit & cil) [inline]
```

Copy one [config_pass_limit](#) object to another.

Parameters

<i>cil</i>	Reference to a config_pass_limit object

The documentation for this class was generated from the following file:

- [config_obj.h](#)

16.60 config_string Class Reference

String Configuration Variable.

```
#include <config_obj.h>
```

Inherits [config_value](#).

Inherited by [config_color_brightlimit](#), [config_fs_bulkstore](#), [config_localname](#), [config_pass](#), [config_string_limit](#), and [config_time_t](#).

Public Member Functions

- virtual void [GetTextValue](#) (NBString &s) override
Get the object value (as a text string with quotations) to the specified NBString object.
- [config_string](#) (config_obj &owner, NBString def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_string](#) (NBString def_val, const char *name, const char *desc=NULL)
Object constructor.
- [config_string](#) (config_obj &owner, const char *def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_string](#) (const char *def_val, const char *name, const char *desc=NULL)
Object constructor.
- void [SetEnumList](#) (NBString s)
Renders the data used to explain the schema/descriptions for the list of choices.
- [operator NBString](#) () const
Return the object value.
- [config_string](#) & [operator=](#) (const char *p)
Assign the config_string object value from a const char string.*
- [config_string](#) & [operator=](#) (const NBString &s)
Assign the config_string object value from a NBString object.
- [config_string](#) & [operator=](#) (const config_string &ci)
Copy one config_string object to another.
- const char * [c_str](#) () const
Returns the object value as a string.
- size_t [length](#) () const
Returns the string length in bytes.
- const char & [operator\[\]](#) (size_t pos) const
Return the value of a character in the string.
- virtual void [GetTypeValue](#) (NBString &s) override
Copy the object type value to the specified NBString object.

Friends

- class [config_pass](#)
- class [config_localname](#)
- class [config_chooser](#)

Additional Inherited Members**Protected Member Functions inherited from [config_value](#)**

- [config_value](#) (config_obj &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.60.1 Detailed Description

String Configuration Variable.

Note

All modifications to configuration objects are marked as pending. A call to [SaveConfigToStorage](#) is required to save changes to flash memory.

16.60.2 Constructor & Destructor Documentation

16.60.2.1 config_string() [1/4]

```
config_string::config_string (  
    config_obj & owner,  
    NBString def_val,  
    const char * name,  
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Default value assigned with a [NBString](#)

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value of type NBString
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.60.2.2 config_string() [2/4]

```
config_string::config_string (  
    NBString def_val,  
    const char * name,  
    const char * desc = NULL) [inline]
```

Object constructor.

Default value assigned with a [NBString](#)

Parameters

<i>def_val</i>	Default value of type NBString
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.60.2.3 config_string() [3/4]

```
config_string::config_string (  
    config_obj & owner,  
    const char * def_val,  
    const char * name,  
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Default value assigned with a character string

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value of type char *
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.60.2.4 config_string() [4/4]

```
config_string::config_string (  
    const char * def_val,  
    const char * name,  
    const char * desc = NULL) [inline]
```

Object constructor.

Default value assigned with a character string

Parameters

<i>def_val</i>	Default value of type char *
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.60.3 Member Function Documentation

16.60.3.1 c_str()

```
const char * config_string::c_str () const [inline]
```

Returns the object value as a string.

Returns

String value of object

16.60.3.2 GetTextValue()

```
virtual void config_string::GetTextValue (  
    NBString & s) [override], [virtual]
```

Get the object value (as a text string with quotations) to the specified [NBString](#) object.

This is a virtual function that can be overridden in any derived class.

Parameters

<i>s</i>	Reference to a NBString object to store the result

Implements [config_value](#).

Reimplemented in [config_pass](#).

16.60.3.3 GetTypeValue()

```
virtual void config_string::GetTypeValue (  
    NBString & s) [inline], [override], [virtual]
```

Copy the object type value to the specified [NBString](#) object.

The type value "string" is written to the [NBString](#) object.

Parameters

<i>s</i>	NBString object to hold the type value string

Reimplemented in [config_time_t](#).

16.60.3.4 length()

```
size_t config_string::length () const [inline]
```

Returns the string length in bytes.

Returns

String length

16.60.3.5 operator NBString()

```
config_string::operator NBString () const [inline]
```

Return the object value.

Returns

The value of the object as a [NBString](#)

16.60.3.6 operator=() [1/3]

```
config_string & config_string::operator= (  
    const char * p) [inline]
```

Assign the [config_string](#) object value from a const char* string.

Parameters

<i>p</i>	String to assign

16.60.3.7 operator=() [2/3]

```
config_string & config_string::operator= (  
    const config_string & ci) [inline]
```

Copy one [config_string](#) object to another.

Parameters

<i>ci</i>	Reference to a config_string object

16.60.3.8 operator=() [3/3]

```
config_string & config_string::operator= (  
    const NBString & s) [inline]
```

Assign the [config_string](#) object value from a [NBString](#) object.

Parameters

<i>s</i>	Reference to a NBString object

16.60.3.9 operator[]()

```
const char & config_string::operator[] (  
    size_t pos) const [inline]
```

Return the value of a character in the string.

Parameters

<i>pos</i>	Character index/position in the string

Returns

Character value at the specified index

16.60.3.10 SetEnumList()

```
void config_string::SetEnumList (
    NBString s) [inline]
```

Renders the data used to explain the schema/descriptions for the list of choices.

Parameters

s	Enumerated list that is a string with choices separated by commas (',') with no spaces. For example, "one,two,three"

The documentation for this class was generated from the following file:

- config_obj.h

16.61 config_string_limit Class Reference

A [config_string](#) with minimum and/or maximum lengths Attempting to set a value shorter than the minimum or longer than the maximum will fail with an error both programmatically and in the Config Web UI.

```
#include <config_obj.h>
```

Inherits [config_string](#).

Public Member Functions

- [config_string_limit](#) ([config_obj](#) &owner, [NBString](#) def_val, [size_t](#) minl=0, [size_t](#) maxl=0, const char *name=NULL, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter and limits.
- [config_string_limit](#) ([NBString](#) def_val, [size_t](#) minl=0, [size_t](#) maxl=0, const char *name=NULL, const char *desc=NULL)
Object constructor with limits.
- [config_string_limit](#) & operator= (const [config_string](#) &ci)
Copy a [config_string](#) object into the [config_string_limit](#).
- [config_string_limit](#) & operator= (const [config_string_limit](#) &cil)
Copy one [config_string_limit](#) object to another.

Public Member Functions inherited from [config_string](#)

- virtual void [GetTextValue](#) ([NBString](#) &s) override
Get the object value (as a text string with quotations) to the specified [NBString](#) object.
- [config_string](#) ([config_obj](#) &owner, [NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_string](#) ([NBString](#) def_val, const char *name, const char *desc=NULL)
Object constructor.
- [config_string](#) ([config_obj](#) &owner, const char *def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_string](#) (const char *def_val, const char *name, const char *desc=NULL)
Object constructor.
- void [SetEnumList](#) ([NBString](#) s)
Renders the data used to explain the schema/descriptions for the list of choices.
- [operator NBString](#) () const
Return the object value.
- [config_string](#) & operator= (const char *p)
Assign the [config_string](#) object value from a const char string.*
- [config_string](#) & operator= (const [NBString](#) &s)

- Assign the `config_string` object value from a `NBString` object.
- `config_string & operator= (const config_string &ci)`
Copy one `config_string` object to another.
- `const char * c_str () const`
Returns the object value as a string.
- `size_t length () const`
Returns the string length in bytes.
- `const char & operator[] (size_t pos) const`
Return the value of a character in the string.
- virtual void `GetTypeValue (NBString &s)` override
Copy the object type value to the specified `NBString` object.

Additional Inherited Members

Protected Member Functions inherited from `config_value`

- `config_value (config_obj &owner, const char *name, const char *desc)`
Object constructor with the parent/owner leaf parameter.
- `config_value (const char *name, const char *desc)`
Object constructor.

16.61.1 Detailed Description

A `config_string` with minimum and/or maximum lengths Attempting to set a value shorter than the minimum or longer than the maximum will fail with an error both programmatically and in the Config Web UI.

16.61.2 Constructor & Destructor Documentation

16.61.2.1 `config_string_limit()` [1/2]

```
config_string_limit::config_string_limit (
    config_obj & owner,
    NBString def_val,
    size_t minl = 0,
    size_t maxl = 0,
    const char * name = NULL,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter and limits.

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value
<i>minl</i>	Minimum length
<i>maxl</i>	Maximum length
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.61.2.2 `config_string_limit()` [2/2]

```
config_string_limit::config_string_limit (
    NBString def_val,
    size_t minl = 0,
    size_t maxl = 0,
```

```
const char * name = NULL,  
const char * desc = NULL) [inline]
```

Object constructor with limits.

Parameters

<i>def_val</i>	Default value
<i>minl</i>	Minimum length
<i>maxl</i>	Maximum length
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.61.3 Member Function Documentation

16.61.3.1 operator=() [1/2]

```
config_string_limit & config_string_limit::operator= (
    const config_string & ci) [inline]
```

Copy a [config_string](#) object into the [config_string_limit](#).

Parameters

<i>ci</i>	Reference to a config_string object

16.61.3.2 operator=() [2/2]

```
config_string_limit & config_string_limit::operator= (
    const config_string_limit & cil) [inline]
```

Copy one [config_string_limit](#) object to another.

Parameters

<i>cil</i>	Reference to a config_string_limit object

The documentation for this class was generated from the following file:

- [config_obj.h](#)

16.62 config_time_t Class Reference

Class to hold time configuration.

```
#include <config_time.h>
```

Inherits [config_string](#).

Public Member Functions

- [config_time_t](#) ([config_obj](#) &owner, time_t def_val, const char *name, const char *desc=NULL, bool bUserSeesGmt=false)
Object constructor with the parent/owner leaf parameter.
- [config_time_t](#) ([config_obj](#) &owner, const char *def_val, const char *name, const char *desc=NULL, bool bUserSeesGmt=false)
Object constructor with the parent/owner leaf parameter.
- [config_time_t](#) (const char *def_val, const char *name, const char *desc=NULL, bool bUserSeesGmt=false)
Object constructor.
- [config_time_t](#) (time_t def_val, const char *name, const char *desc=NULL)
Object constructor.
- virtual void **GetTypeValue** ([NBString](#) &s)

- `operator time_t () const`
- `config_time_t & operator= (time_t t)`
Assign the `config_time_t` object value from a another `NBString` object.
- `config_time_t & operator= (const config_time_t &ci)`
Copy one `config_string` object to another.
- `NBString GetAsGmtString ()`
- `NBString GetAsLocalString ()`
- virtual int **Missing_ConfigEndMarker** (void *&startp)
This is named so error messages make sense to user, old name was `GetExtent`.
- void **RenderValue** (int fd, int len, const char *extra)
- void **RenderInput** (int fd, int len, const char *extra)
- bool **ProcessValue** (const char *pValue)

Public Member Functions inherited from `config_string`

- virtual void `GetTextValue` (`NBString` &s) override
Get the object value (as a text string with quotations) to the specified `NBString` object.
- `config_string` (`config_obj` &owner, `NBString` def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- `config_string` (`NBString` def_val, const char *name, const char *desc=NULL)
Object constructor.
- `config_string` (`config_obj` &owner, const char *def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- `config_string` (const char *def_val, const char *name, const char *desc=NULL)
Object constructor.
- void `SetEnumList` (`NBString` s)
Renders the data used to explain the schema/descriptions for the list of choices.
- `operator NBString () const`
Return the object value.
- `config_string & operator= (const char *p)`
Assign the `config_string` object value from a const char string.*
- `config_string & operator= (const NBString &s)`
Assign the `config_string` object value from a `NBString` object.
- `config_string & operator= (const config_string &ci)`
Copy one `config_string` object to another.
- const char * `c_str` () const
Returns the object value as a string.
- size_t `length` () const
Returns the string length in bytes.
- const char & `operator[]` (size_t pos) const
Return the value of a character in the string.

Static Public Member Functions

- static [NBString](#) **TimetoNBString** (time_t tv, bool bUtc=true)
- static time_t **ParseTime** (const char *cp, bool bUtc=true)
- static time_t **ParseTime** ([NBString](#) &ns, bool bUtc=true)

Additional Inherited Members

Protected Member Functions inherited from [config_value](#)

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.62.1 Detailed Description

Class to hold time configuration.

16.62.2 Constructor & Destructor Documentation

16.62.2.1 config_time_t() [1/4]

```
config_time_t::config_time_t (
    config\_obj & owner,
    time_t def_val,
    const char * name,
    const char * desc = NULL,
    bool bUserSeesGmt = false) [inline]
```

Object constructor with the parent/owner leaf parameter.
Default value assigned with a time_t

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value of type time_t
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip
<i>bUserSeesGmt</i>	If true, the user sees GMT time when displaying/reading time values. If false, localtime is used.

16.62.2.2 config_time_t() [2/4]

```
config_time_t::config_time_t (
    config\_obj & owner,
    const char * def_val,
    const char * name,
    const char * desc = NULL,
    bool bUserSeesGmt = false) [inline]
```

Object constructor with the parent/owner leaf parameter.
Default value assigned with a time_t

Parameters

<i>owner</i>	Reference to owner/parent leaf

Parameters

<i>def_val</i>	Default value of type time_t
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip
<i>bUserSeesGmt</i>	If true, the user sees GMT time when displaying/reading time values. If false, localtime is used.

16.62.2.3 config_time_t() [3/4]

```
config_time_t::config_time_t (
    const char * def_val,
    const char * name,
    const char * desc = NULL,
    bool bUserSeesGmt = false) [inline]
```

Object constructor.

Default value assigned with a time_t

Parameters

<i>def_val</i>	Default value of type time_t
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip
<i>bUserSeesGmt</i>	If true, the user sees GMT time when displaying/reading time values. If false, localtime is used.

16.62.2.4 config_time_t() [4/4]

```
config_time_t::config_time_t (
    time_t def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor.

Parameters

<i>def_val</i>	Default value of type time_t
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.62.3 Member Function Documentation**16.62.3.1 operator time_t()**

```
config_time_t::operator time_t () const [inline]
```

Return the object value

Returns

The value of the object as a time_t

16.62.3.2 operator=() [1/2]

```
config_time_t & config_time_t::operator= (
    const config_time_t & ci) [inline]
```

Copy one [config_string](#) object to another.

Parameters

<i>ci</i>	Reference to a config_string object

16.62.3.3 operator=() [2/2]

```
config_time_t & config_time_t::operator= (
    time_t t) [inline]
```

Assign the [config_time_t](#) object value from a anotherNBString object.

Parameters

<i>t</i>	time_t ↔
	<i>_t</i>

The documentation for this class was generated from the following file:

- [config_time.h](#)

16.63 config_uint Class Reference

Unsigned 32-bit Integer Configuration Variable.

```
#include <config_obj.h>
```

Inherits [config_value](#).

Inherited by [config_hex_uint](#), and [config_uint_limit](#).

Public Member Functions

- virtual void [GetTextValue](#) (NBString &s) override
Copy the object value to the specified NBString object.
- [config_uint](#) (config_obj &owner, uint32_t def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_uint](#) (uint32_t def_val, const char *name, const char *desc=NULL)
Object constructor.
- [operator uint32_t](#) () const
Return the object value.
- [config_uint](#) & [operator=](#) (const uint32_t i)
Assign the config_uint object value from a uint32_t value.
- [config_uint](#) & [operator=](#) (const [config_uint](#) &ci)
Copy one config_uint object to another.
- virtual void [GetTypeValue](#) (NBString &s) override
Copy the object type value to the specified NBString object.

Additional Inherited Members**Protected Member Functions inherited from [config_value](#)**

- [config_value](#) (config_obj &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.63.1 Detailed Description

Unsigned 32-bit Integer Configuration Variable.

Note

All modifications to configuration objects are marked as pending. A call to [SaveConfigToStorage](#) is required to save changes to flash memory.

16.63.2 Constructor & Destructor Documentation

16.63.2.1 config_uint() [1/2]

```
config_uint::config_uint (
    config_obj & owner,
    uint32_t def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter.

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.63.2.2 config_uint() [2/2]

```
config_uint::config_uint (
    uint32_t def_val,
    const char * name,
    const char * desc = NULL) [inline]
```

Object constructor.

Parameters

<i>def_val</i>	Default value
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.63.3 Member Function Documentation

16.63.3.1 GetTextValue()

```
virtual void config_uint::GetTextValue (
    NBString & s) [inline], [override], [virtual]
```

Copy the object value to the specified [NBString](#) object.

This is a virtual function that can be overridden in any derived class.

Parameters

<i>s</i>	Reference to a NBString object to store the result

Implements [config_value](#).

16.63.3.2 GetTypeValue()

```
virtual void config_uint::GetTypeValue (
    NBString & s) [inline], [override], [virtual]
```

Copy the object type value to the specified [NBString](#) object.
The type value "integer" is written to the [NBString](#) object.

Parameters

<code>s</code>	NBString object to hold the type value string

16.63.3.3 operator uint32_t()

```
config_uint::operator uint32_t () const [inline]
```

Return the object value.

Returns

The value of the object as a `uint32_t`

Example: for a variable named `configuint` of type [config_uint](#):

```
uint32_t value = uint32_t(configuint);
iprintf("int = %ld\r\n", value);
```

16.63.3.4 operator=() [1/2]

```
config_uint & config_uint::operator= (
    const config_uint & ci) [inline]
```

Copy one [config_uint](#) object to another.

Parameters

<code>ci</code>	Reference to a config_uint object

16.63.3.5 operator=() [2/2]

```
config_uint & config_uint::operator= (
    const uint32_t i) [inline]
```

Assign the [config_uint](#) object value from a `uint32_t` value.

Parameters

<code>i</code>	<code>uint32_t</code> value

The documentation for this class was generated from the following file:

- `config_obj.h`

16.64 config_uint_limit Class Reference

A [config_uint](#) with minimum and/or maximum values Attempting to set a value lower than the minimum or greater than the maximum will fail with an error both programmatically and in the Config Web UI.

```
#include <config_obj.h>
```

Inherits [config_uint](#).

Public Member Functions

- [config_uint_limit](#) ([config_obj](#) &owner, uint32_t def_val, uint32_t minv=0, uint32_t maxv=UINT_MAX, const char *name=NULL, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter and limits.
- [config_uint_limit](#) (uint32_t def_val, uint32_t minv=0, uint32_t maxv=UINT_MAX, const char *name=NULL, const char *desc=NULL)
Object constructor with limits.
- [config_uint_limit](#) & [operator=](#) (const [config_uint](#) &ci)
Copy a [config_uint](#) object into the [config_uint_limit](#).
- [config_uint_limit](#) & [operator=](#) (const [config_uint_limit](#) &cil)
Copy one [config_uint_limit](#) object to another.

Public Member Functions inherited from [config_uint](#)

- virtual void [GetTextValue](#) ([NBString](#) &s) override
Copy the object value to the specified [NBString](#) object.
- [config_uint](#) ([config_obj](#) &owner, uint32_t def_val, const char *name, const char *desc=NULL)
Object constructor with the parent/owner leaf parameter.
- [config_uint](#) (uint32_t def_val, const char *name, const char *desc=NULL)
Object constructor.
- [operator uint32_t](#) () const
Return the object value.
- [config_uint](#) & [operator=](#) (const uint32_t i)
Assign the [config_uint](#) object value from a uint32_t value.
- [config_uint](#) & [operator=](#) (const [config_uint](#) &ci)
Copy one [config_uint](#) object to another.
- virtual void [GetTypeValue](#) ([NBString](#) &s) override
Copy the object type value to the specified [NBString](#) object.

Additional Inherited Members**Protected Member Functions inherited from [config_value](#)**

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.64.1 Detailed Description

A [config_uint](#) with minimum and/or maximum values Attempting to set a value lower than the minimum or greater than the maximum will fail with an error both programmatically and in the Config Web UI.

16.64.2 Constructor & Destructor Documentation**16.64.2.1 [config_uint_limit](#)() [1/2]**

```
config_uint_limit::config_uint_limit (
    config\_obj & owner,
    uint32_t def_val,
    uint32_t minv = 0,
    uint32_t maxv = UINT_MAX,
    const char * name = NULL,
    const char * desc = NULL) [inline]
```

Object constructor with the parent/owner leaf parameter and limits.

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>def_val</i>	Default value
<i>minv</i>	Minimum value
<i>maxv</i>	Maximum value
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.64.2.2 config_uint_limit() [2/2]

```
config_uint_limit::config_uint_limit (
    uint32_t def_val,
    uint32_t minv = 0,
    uint32_t maxv = UINT_MAX,
    const char * name = NULL,
    const char * desc = NULL) [inline]
```

Object constructor with limits.

Parameters

<i>def_val</i>	Default value
<i>minv</i>	Minimum value
<i>maxv</i>	Maximum value
<i>name</i>	Field name assigned to the value
<i>desc</i>	Description used for info/tool tip

16.64.3 Member Function Documentation**16.64.3.1 operator=() [1/2]**

```
config_uint_limit & config_uint_limit::operator= (
    const config_uint & ci) [inline]
```

Copy a [config_uint](#) object into the [config_uint_limit](#).

Parameters

<i>ci</i>	Reference to a config_uint object

16.64.3.2 operator=() [2/2]

```
config_uint_limit & config_uint_limit::operator= (
    const config_uint_limit & cil) [inline]
```

Copy one [config_uint_limit](#) object to another.

Parameters

<i>cil</i>	Reference to a config_uint_limit object

The documentation for this class was generated from the following file:

- [config_obj.h](#)

16.65 config_value Class Reference

Base class used to create a configuration value.

```
#include <config_obj.h>
```

Inherits config_leaf.

Inherited by [CUR_IPADDR4](#), [Dynamic_IPADDR](#), [config_IPADDR](#), [config_IPADDR4](#), [config_MACADR](#), [config_bool](#), [config_double](#), [config_int](#), [config_report](#), [config_string](#), and [config_uint](#).

Protected Member Functions

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.65.1 Detailed Description

Base class used to create a configuration value.

Base class that can be used to create custom configuration values in a JSON compatible format. The JSON compatible types are: integer, boolean, null, string, object or array.

The system already provides the common JSON value types, including: [config_int](#), [config_uint](#), [config_bool](#), [config_chooser](#), [config_string](#), [config_pass](#), [config_IPADDR4](#), [config_IPADDR](#), [config_MACADR](#),

A [config_obj](#) can hold multiple values in a JSON object. A [config_value](#) holds only a single JSON value.

See also

[config_obj](#)

16.65.2 Constructor & Destructor Documentation

16.65.2.1 config_value() [1/2]

```
config_value::config_value (
    config_obj & owner,
    const char * name,
    const char * desc) [inline], [protected]
```

Object constructor with the parent/owner leaf parameter.

For example,

```
myValue(config_obj &owner, const char *name, const char *desc = nullptr) : config_value(owner, name,
desc){};
```

Parameters

<i>owner</i>	Reference to owner/parent leaf
<i>name</i>	Field name
<i>desc</i>	Description used for info/tool tip

16.65.2.2 config_value() [2/2]

```
config_value::config_value (
    const char * name,
    const char * desc) [inline], [protected]
```

Object constructor.

This constructor should be called as part of the derived class constructor. For example,

```
myValue(const char *name, const char *desc = nullptr) : config_value(name, desc){}
```

Parameters

<i>name</i>	Field name

<i>desc</i>	Description used for info/tool tip

The documentation for this class was generated from the following file:

- config_obj.h

16.66 MQTT::ConfigExposer Class Reference

Easy MqttObj class for automatically publishing and subscribing NetBurner Config system objects to/from an [MQTT](#) topic. Use like a normal ConfigObj.

```
#include <mqtt.h>
```

Inherits [MQTT::mqtt_leaf](#).

Public Types

- enum [eRenderFlag_t](#) { [eRender_Hidden](#) , [eRender_ModifiedOnly](#) , [eRender_Pretty](#) }
Rendering options for [MQTT](#) output of Config objects.

Public Types inherited from [MQTT::mqtt_leaf](#)

- enum [eObj_State_t](#)
Mqtt Object Publication/Subscription States.

Public Member Functions

- [ConfigExposer](#) (config_leaf &leaf, const char *topicRoot, [Client](#) *mClient=NULL, uint16_t initFlags=0, uint32_t renderFlags=0)
Constructor for a standalone [ConfigExposer](#) object.
- [ConfigExposer](#) (mqtt_leaf &parent, config_leaf &leaf, const char *topicRoot, [Client](#) *mClient=NULL, uint16_t initFlags=0, uint32_t renderFlags=0)
Constructor for a [ConfigExposer](#) object that has a parent.
- virtual int [GetPubTopicName](#) (NBString &s) override
Get the Topic Name this object Publishes to.
- virtual int [GetSubTopicFilter](#) (NBString &s) override
Get the Subscription Topic Filter this object Subscribes to.
- virtual ValueTestResult [TestOrSetValue](#) (ParsedJsonDataSet &pjs, bool setVal) override
whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.
- virtual int [SubscribeEvent](#) (int mqttFd, [PacketInfo::Msg](#) *msg) override
Handle a message sent by the Broker triggered by this object's Subscription.
- virtual int [GetRenderLength](#) ([eObj_Serialize_t](#) form=[eObj_Serialize_CURRENT_FORM](#)) override
the number of bytes used to serialize this object in a particular format.
- virtual int [RenderToBuffer](#) ([eObj_Serialize_t](#) form, uint8_t *buf, int buflen) override
Render this object to an output buffer.
- virtual int [RenderToFd](#) ([eObj_Serialize_t](#) form, int fd) override
Render this object to an output socket.

Public Member Functions inherited from MQTT::mqtt_leaf

- `~mqtt_leaf ()`
~mqtt_leaf Destructor for base [mqtt_leaf](#)
- `void RemoveFromTree ()`
Remove this object from it's parent tree.
- `bool NameMatch (const char *cp)`
Determine whether a given name matches this object.
- `int Publish (int waitForCompletion=-1)`
Publishes the current [MQTT](#) object, optionally waiting for completion.
- `void SetPublishInterval (uint32_t intervalPeriodTicks)`
Sets or clears an automatic publishing interval for this [MQTT](#) object.
- `virtual void SetSerializationForm (eObj_Serialize_t form)`
Set the default serialization format for this object.
- `eObj_Serialize_t GetSerializationForm ()`
Get the default serialization format for this object.
- `virtual void SetFlags (uint16_t newFlags)`
Set additional behavioral flags for this object.
- `virtual void ClearFlags (uint16_t delFlags)`
Clear behavioral flags from this object.
- `uint16_t GetFlags ()`
Get the behavioral flags set for this object.
- `int getPriority ()`
Get the message priority for this object.

Protected Member Functions

- `virtual int ParseFd (int mqttFd) override`
Parse the data from a file descriptor according to this object's current default serialization method and assign the value.

Protected Member Functions inherited from MQTT::mqtt_leaf

- `mqtt_leaf (const char *topic, Client *mClient, uint16_t flags=(eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_N`
Constructor for [mqtt_leaf](#) objects.
- `mqtt_leaf (mqtt_leaf &parent, const char *topic, Client *mClient, uint16_t flags=(eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|`
Constructor for [mqtt_leaf](#) objects.
- `virtual mqtt_leaf * GetChildren ()`
Get the first child of this object.
- `virtual mqtt_leaf * AddChild (mqtt_leaf *child)`
Add an MqttObj as a child to this object.
- `virtual mqtt_leaf * RemoveFirstChild ()`
Remove the first child from this object.
- `void ForEachLeaf (LeafCallBack *p, void *pd)`
Call a function on each element of the MqttObj tree with this object as the root.
- `void RootParse (ParsedJsonDataSet &pjs)`
Assign to this object tree from a JSON dataset.
- `bool MatchId (ParsedJsonDataSet &pjs)`
Determine whether the JSON dataset's current index matches this object's name.
- `int QueuePublish ()`
Queue this object to be Published via the registered [Client](#).
- `int QueueSubscribe ()`

- *Queue this object to be Subscribed via the registered [Client](#).*
- `int QueueUnsubscribe ()`
Queue this object to be Unsubscribed via the registered [Client](#).
- `int Assign ()`
Perform post assignemnt operations.
- `uint8_t PubState ()`
Get the Publish state.
- `uint8_t SubState ()`
Get the Subscribe state of this object.
- `virtual void SetPubState (uint8_t newState)`
Set the Publish state.
- `virtual void SetSubState (uint8_t newState)`
Set the Subscribe state.
- `int HandleSubAck (int mqttFd, PacketInfo::Msg *msg)`
Handle a Subscribe Acknowledge message from the broker for our Subscribe request.
- `int HandleUnsubAck (int mqttFd, PacketInfo::Msg *msg)`
Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.
- `int HandlePubAck (int mqttFd, PacketInfo::Msg *msg)`
Handle an Publish Acknowledge message from the broker for our Publish request.

Additional Inherited Members

Static Public Member Functions inherited from [MQTT::mqtt_leaf](#)

- `static int MqttRxPublish (Client *mClient, int mqttFd, PacketInfo::Msg *msg, void *ctx)`
Static message handler for Publish messages received by the registered [Client](#).

Protected Attributes inherited from [MQTT::mqtt_leaf](#)

- `NBInterpolatedString topicName`
Topic name or additional branch.
- `eObj\_Serialize\_t publishForm`
Default serialization form.
- `uint16_t mFlags`
Behavioral flags.

16.66.1 Detailed Description

Easy MqttObj class for automatically publishing and subscribing NetBurner Config system objects to/from an [MQTT](#) topic. Use like a normal ConfigObj.

16.66.2 Member Enumeration Documentation

16.66.2.1 eRenderFlag_t

enum [MQTT::ConfigExposer::eRenderFlag_t](#)

Rendering options for [MQTT](#) output of Config objects.

Enumerator

<code>eRender_Hidden</code>	Hide this branch from renders of the object.
<code>eRender_ModifiedOnly</code>	Only render modified leaves of the branch.
<code>eRender_Pretty</code>	Do Pretty Print rendering.

16.66.3 Constructor & Destructor Documentation

16.66.3.1 ConfigExposer() [1/2]

```
MQTT::ConfigExposer::ConfigExposer (
    config_leaf & leaf,
    const char * topicRoot,
    Client * mClient = NULL,
    uint16_t initFlags = 0,
    uint32_t renderFlags = 0)
```

Constructor for a standalone [ConfigExposer](#) object.

Parameters

<i>&leaf</i>	Reference to the ConfigObject branch being exposed
<i>*topicRoot</i>	Mqtt Topic Name to use as the base for Publish/Subscribe.
<i>*mClient</i>	Pointer to Client to register with. If NULL, it will be registered with the first created Client .
<i>initFlags</i>	Behavioral flags to set on the created object
<i>renderFlags</i>	Rendering flags to set for the exposed ConfigObj

16.66.3.2 ConfigExposer() [2/2]

```
MQTT::ConfigExposer::ConfigExposer (
    mqtt_leaf & parent,
    config_leaf & leaf,
    const char * topicRoot,
    Client * mClient = NULL,
    uint16_t initFlags = 0,
    uint32_t renderFlags = 0)
```

Constructor for a [ConfigExposer](#) object that has a parent.

Parameters

<i>&parent</i>	Reference to parent MqttObj
<i>&leaf</i>	Reference to the ConfigObject branch being exposed
<i>*topicRoot</i>	Mqtt Topic Name to use as the base for Publish/Subscribe.
<i>*mClient</i>	Pointer to Client to register with. If NULL, it will be registered with the first created Client .
<i>initFlags</i>	Behavioral flags to set on the created object
<i>renderFlags</i>	Rendering flags to set for the exposed ConfigObj

16.66.4 Member Function Documentation

16.66.4.1 GetPubTopicName()

```
virtual int MQTT::ConfigExposer::GetPubTopicName (
    NBString & s) [override], [virtual]
```

Get the Topic Name this object Publishes to.

Parameters

<i>&s</i>	The NBString to copy the Topic Name into

Returns

eResult_t return code

Reimplemented from [MQTT::mqtt_leaf](#).

16.66.4.2 GetRenderLength()

```
virtual int MQTT::ConfigExposer::GetRenderLength (
    eObj_Serialize_t form = eObj_Serialize_CURRENT_FORM) [override], [virtual]
```

the number of bytes used to serialize this object in a particular format.

Parameters

<i>form</i>	Serialization format to use

Returns

Length of serialized data

Reimplemented from [MQTT::mqtt_leaf](#).

16.66.4.3 GetSubTopicFilter()

```
virtual int MQTT::ConfigExposer::GetSubTopicFilter (
    NBString & s) [override], [virtual]
```

Get the Subscription Topic Filter this object Subscribes to.

Parameters

&s	The NBString to copy the Subscription Topic Filter into

Returns

eResult_t return code

Reimplemented from [MQTT::mqtt_leaf](#).

16.66.4.4 ParseFd()

```
virtual int MQTT::ConfigExposer::ParseFd (
    int mqttFd) [override], [protected], [virtual]
```

Parse the data from a file descriptor according to this object's current default serialization method and assign the value.

Parameters

<i>mqttFd</i>	File descriptor to read from.

Returns

eResult_t return code

Implements [MQTT::mqtt_leaf](#).

16.66.4.5 RenderToBuffer()

```
virtual int MQTT::ConfigExposer::RenderToBuffer (
    eObj_Serialize_t form,
    uint8_t * buf,
    int buflen) [override], [virtual]
```

Render this object to an output buffer.

Parameters

<i>form</i>	Serialization format to render as

<i>*buf</i>	Pointer to output buffer
<i>buflen</i>	Size of the output buffer

Returns

Length of the rendered data

Implements [MQTT::mqtt_leaf](#).

16.66.4.6 RenderToFd()

```
virtual int MQTT::ConfigExposer::RenderToFd (  
    eObj_Serialize_t form,  
    int fd) [override], [virtual]
```

Render this object to an output socket.

Parameters

<i>form</i>	Serialization format to render as
<i>fd</i>	File descriptor to render to

Returns

Length of the rendered data

Implements [MQTT::mqtt_leaf](#).

16.66.4.7 SubscribeEvent()

```
virtual int MQTT::ConfigExposer::SubscribeEvent (  
    int mqttFd,  
    PacketInfo::Msg * msg) [override], [virtual]
```

Handle a message sent by the Broker triggered by this object's Subscription.

Parameters

<i>mqttFd</i>	File Descriptor to read any additional message data from
<i>*msg</i>	Pointer to message received from the Broker

Returns

eResult_t return code

Reimplemented from [MQTT::mqtt_leaf](#).

16.66.4.8 TestOrSetValue()

```
virtual ValueTestResult MQTT::ConfigExposer::TestOrSetValue (  
    ParsedJsonDataSet & pjs,  
    bool setVal) [override], [virtual]
```

whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.

Parameters

<i>&pjs</i>	The JSON dataset to use
<i>setVal</i>	true: Assign from the value at the current index, false: test if current index is valid for this object

Returns

ValueTestResult indicating the validity of the current index

Implements [MQTT::mqtt_leaf](#).

The documentation for this class was generated from the following file:

- mqtt.h

16.67 MQTT::ConfiguredClient Class Reference

Easy [MQTT Client](#) wrapper for interacting with [MQTT](#) brokers. The client handles all network communications with the Broker and handles local message queueing and dispatching to topic handlers. Adds ConfigObj for automatically configuring all [Client](#) settings via the Config System.

```
#include <mqtt.h>
```

Inherits [config_MqttClient](#), and [MQTT::Client](#).

Public Member Functions

- [ConfiguredClient](#) ([config_obj](#) &owner=sys, const char *name=NULL, const char *desc="Mqtt client connection configuration", RequestProperty *addProps=NULL, bool ConnectOnStartup=true)

[ConfiguredClient](#) constructor.

Public Member Functions inherited from [MQTT::Client](#)

- [Client](#) (const char *brokerURI, const char *pClientId, uint16_t maxKeepAlive=[MQTT_DEFAULT_KEEP_ALIVE](#), bool bCleanStart=true, [PublishRequest](#) *lwtMsg=NULL, const char *pConnMsg=NULL, const char *pDisconnMsg=NULL, const char *pUsername=NULL, const char *pPassword=NULL, RequestProperty *addProps=NULL, bool ConnectOnStartup=true)
Constructor. This version takes a raw string for the broker URI.
- [Client](#) (const [ParsedURI](#) *brokerURI, const char *pClientId, uint16_t maxKeepAlive=[MQTT_DEFAULT_KEEP_ALIVE](#), bool bCleanStart=true, [PublishRequest](#) *lwtMsg=NULL, const char *pConnMsg=NULL, const char *pDisconnMsg=NULL, const char *pUsername=NULL, const char *pPassword=NULL, RequestProperty *addProps=NULL, bool ConnectOnStartup=true)
Constructor. This version takes a [ParsedURI](#) object for the broker URI.
- int [Connect](#) (bool bWaitForConnection, const char *pNewBrokerURI, ConnAckCb_t notifyCb=NULL, void *notifyCtx=NULL)
Begin the process of Connecting the client to the Broker.
- int [Connect](#) (bool bWaitForConnection=true, const [ParsedURI](#) *pNewBrokerURI=NULL, ConnAckCb_t notifyCb=NULL, void *notifyCtx=NULL)
Begin the process of Connecting the client to the Broker.
- int [Disconnect](#) ()
Disconnect the [Client](#) from the Broker.
- void [SetConnectNotify](#) (ConnAckCb_t notifyCb, void *notifyCtx)
Set the connection notification callback and it's context information.
- int [SetRateLimit](#) ([eRateLimitType](#) type, uint8_t bucketMax, uint8_t fillCount, uint8_t fillTicks, [eRateLimitAdjustMethod](#) adjMethod, uint8_t adjCount)
Set the value for a given rate limit.
- int [SetRateLimit_Publish](#) (int reqPerSecond)
Set the Publish rate limit. This method automatically determines the fillCount and fillTicks to use for refilling the credit bucket.
- int [SetRateLimit_Subscribe](#) (int reqPerSecond)
Set the Subscribe rate limit. This method automatically determines the fillCount and fillTicks to use for refilling the credit bucket.
- [eConnectionStatus](#) [GetConnectionStatus](#) ()
Get the current state of the [Client](#)'s connection to the Broker.

- void [SetClientCert](#) (uint8_t *cert, uint32_t len, uint8_t *privkey, uint32_t privlen)
Set the [Client](#) Certificate and Private Key to use when connecting to a Broker via TLS.
- void [GetClientCert](#) (uint8_t **cert, uint32_t *len, uint8_t **privkey, uint32_t *privlen)
Set the [Client](#) Certificate and Private Key to use when connecting to a Broker via TLS.
- int [QueuePublish](#) ([mqtt_leaf](#) &leaf)
Queue an [MqttObj](#) to be published.
- int [QueueSubscribe](#) ([mqtt_leaf](#) &leaf)
Queue an [MqttObj](#) to subscribe.
- int [QueueUnsubscribe](#) ([mqtt_leaf](#) &leaf)
Queue an [MqttObj](#) to unsubscribe.
- int [QueueAllSubscriptions](#) ()
Queue all [Client](#)'s registered [MqttObjs](#) to be subscribed and published according to the object's operation flags.
- int [Publish](#) (const char *topic, uint8_t *pData, uint32_t len, int prio, TickTimeout &t, uint8_t qos=0)
Publish a binary payload message.
- int [Publish](#) (const char *topic, const char *pStr, int prio, TickTimeout &t, uint8_t qos=0)
Publish a message.
- int [Publish](#) ([PublishRequest](#) *req, int prio, TickTimeout &t)
Publish a fully defined request.
- int [QueuePublish](#) ([PublishRequest](#) *req, int *prio)
Queue a publish message from a fully defined request.
- int [Subscribe](#) ([TopicHandler](#) &handler, int *handlerIdx, TickTimeout &t, uint8_t subOpts=[Pkt::eSubOpt_NoLocal](#))
Register a [TopicHandler](#) and subscribe its Topic on the Broker.
- int [Subscribe](#) ([SubscribeRequest](#) *req, int *handlerIdx, TickTimeout &t)
Register a [TopicHandler](#) (via fully specified [SubscribeRequest](#)) and subscribe its Topic on the Broker.
- int [QueueSubscribe](#) ([SubUnsubQueueMsg](#) *qmsg)
Register a [TopicHandler](#) and subscribe its Topic on the Broker.
- int [Unsubscribe](#) (int handlerIdx, TickTimeout &t, const char *topicFilter=NULL)
Unsubscribe and deregister a [TopicHandler](#).
- int [Unsubscribe](#) (const char *topicFilter, TickTimeout &t)
Unsubscribe and deregister a [TopicHandler](#).
- int [QueueUnsubscribe](#) ([SubUnsubQueueMsg](#) *qmsg)
Queue unsubscribing and deregistering a [TopicHandler](#).
- int [RegisterTopicHandler](#) ([TopicHandler](#) *newHandler, int *handlerIdx)
Register a [TopicHandler](#) with the [Client](#).
- int [RemoveTopicHandler](#) ([TopicHandler](#) *handler)
Remove/deregister a [TopicHandler](#) from the [Client](#) by reference.
- int [RemoveTopicHandler](#) (int handlerIdx)
Remove/deregister a [TopicHandler](#) from the [Client](#) by index.
- void [RegisterMqttObj](#) ([mqtt_leaf](#) &leaf, uint32_t interval=0)
Register an [MqttObj](#) with the [Client](#).
- void [RemoveObj](#) ([mqtt_leaf](#) &leaf)
Deregister/remove an [MqttObj](#) from the [Client](#).
- int [SendNextPubReq](#) ()
Send the next queued Publish Request.
- int [SendNextSubReq](#) ()
Send the next queued SubUnSub Request.

Additional Inherited Members

Public Types inherited from MQTT::Client

- enum [eSessionLimit](#) {
[eLimit_Size_Tx](#) , [eLimit_Size_Rx](#) , [eLimit_InFlight_Tx](#) , [eLimit_InFlight_Rx](#) ,
[eLimit_Subscriptions](#) , [eLimit_SubAliases](#) , [eLimit_PublishAliases](#) , [eLimit_QoS](#) ,
[eLimit_SessionExpiry](#) }
Limit types for the [Client](#) interactions with the Broker.
- enum [eRateLimitType](#) { [eRateLimit_Publish](#) , [eRateLimit_Subscribe](#) }
Rate limit types for [Client](#) messaging.
- enum [eRateLimitAdjustMethod](#) { [eRateLimitAdj_None](#) , [eRateLimitAdj_Set](#) , [eRateLimitAdj_Add](#) ,
[eRateLimitAdj_Subtract](#) }
How to adjust rate limit buckets when modifying the current rate limit.
- enum [eConnectionStatus](#) {
[eConnStat_Error](#) , [eConnStat_No_Broker_Configured](#) , [eConnStat_Resolving](#) , [eConnStat_InProgress](#) ,
[eConnStat_Established](#) , [eConnStat_Refused](#) , [eConnStat_Disconnected](#) }
Connection state values.

Static Public Member Functions inherited from MQTT::Client

- static void [StartTask](#) (uint8_t taskPrio=[MQTT_TASK_PRIO](#))
Start the [Client](#) Task.
- static [Client](#) * [GetFirstClient](#) ()
Start the [Client](#) Task.

16.67.1 Detailed Description

Easy [MQTT Client](#) wrapper for interacting with [MQTT](#) brokers. The client handles all network communications with the Broker and handles local message queueing and dispatching to topic handlers. Adds ConfigObj for automatically configuring all [Client](#) settings via the Config System.

16.67.2 Constructor & Destructor Documentation

16.67.2.1 ConfiguredClient()

```
MQTT::ConfiguredClient::ConfiguredClient (
    config\_obj & owner = sys,
    const char * name = NULL,
    const char * desc = "Mqtt client connection configuration",
    RequestProperty * addProps = NULL,
    bool ConnectOnStartup = true)
```

[ConfiguredClient](#) constructor.

Parameters

<i>&owner</i>	Reference to the ConfigObj this Client should be registered below
<i>*name</i>	Name of this client in the Config System. If NULL, a default name will automatically be assigned.
<i>*desc</i>	Description of this client to provide within the Config System
<i>*addProps</i>	Additional message Properties to send with Connection Requests
<i>ConnectOnStartup</i>	Whether the client should automatically connect after startup.

The documentation for this class was generated from the following file:

- mqtt.h

16.68 MQTT::ConnectRequest Struct Reference

Structure used to specify details for a Publish call.

```
#include <mqtt_msg_reqs.h>
```

16.68.1 Detailed Description

Structure used to specify details for a Publish call.

The documentation for this struct was generated from the following file:

- mqtt_msg_reqs.h

16.69 CUR_IPADDR4 Class Reference

```
#include <config_netobj.h>
```

Inherits [config_value](#).

Public Member Functions

- [CUR_IPADDR4](#) (const char *name, const char *desc=NULL)
- virtual void [GetTextValue](#) (NBString &s)
- virtual ConfigTestResult [TestNewValue](#) (ParsedJsonDataSet &pjs)
- virtual void [CommitTestedValue](#) (uint32_t permission_mask)
- virtual void [ExtendedSchema](#) (int fd, int indent, bool pretty)
- virtual void [GetTypeValue](#) (NBString &s)
- [operator IPADDR4](#) ()
- bool [IsNull](#) () const
- bool [NotNull](#) () const
- void [SetNull](#) ()
- bool [IsLoopBack](#) () const
- bool [IsMultiCast](#) () const
- bool [IsGlobalBroadCast](#) () const
- bool [IsAutoIP](#) ()

Additional Inherited Members

Protected Member Functions inherited from [config_value](#)

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.69.1 Detailed Description

16.69.2 Constructor & Destructor Documentation

16.69.2.1 CUR_IPADDR4()

```
CUR_IPADDR4::CUR_IPADDR4 (
    const char * name,
    const char * desc = NULL) [inline]
```

16.69.3 Member Function Documentation

16.69.3.1 CommitTestedValue()

```
virtual void CUR_IPADDR4::CommitTestedValue (
    uint32_t permission_mask) [inline], [virtual]
```

16.69.3.2 ExtendedSchema()

```
virtual void CUR_IPADDR4::ExtendedSchema (
    int fd,
    int indent,
    bool pretty) [inline], [virtual]
```

16.69.3.3 GetTextValue()

```
virtual void CUR_IPADDR4::GetTextValue (
    NBString & s) [inline], [virtual]
Implements config\_value.
```

16.69.3.4 GetTypeValue()

```
virtual void CUR_IPADDR4::GetTypeValue (
    NBString & s) [inline], [virtual]
```

16.69.3.5 IsAutoIP()

```
bool CUR_IPADDR4::IsAutoIP () [inline]
```

16.69.3.6 IsGlobalBroadCast()

```
bool CUR_IPADDR4::IsGlobalBroadCast () const [inline]
```

16.69.3.7 IsLoopBack()

```
bool CUR_IPADDR4::IsLoopBack () const [inline]
```

16.69.3.8 IsMultiCast()

```
bool CUR_IPADDR4::IsMultiCast () const [inline]
```

16.69.3.9 IsNull()

```
bool CUR_IPADDR4::IsNull () const [inline]
```

16.69.3.10 NotNull()

```
bool CUR_IPADDR4::NotNull () const [inline]
```

16.69.3.11 operator IPADDR4()

```
CUR_IPADDR4::operator IPADDR4 () [inline]
```

16.69.3.12 SetNull()

```
void CUR_IPADDR4::SetNull () [inline]
```

16.69.3.13 TestNewValue()

```
virtual ConfigTestResult CUR_IPADDR4::TestNewValue (
    ParsedJsonDataSet & pjs) [inline], [virtual]
```

The documentation for this class was generated from the following file:

- `config_netobj.h`

16.70 DelayObject Class Reference

High-resolution microsecond delay timer class.

```
#include <HiResDelay.h>
```

Public Member Functions

- [DelayObject](#) (int Timer=FIRST_UNUSED_TIMER)
Construct a high-resolution delay timer object.
- [~DelayObject](#) ()
Destructor - releases the allocated hardware timer.
- void [DelayUsec](#) (uint32_t usec)
Perform a blocking delay with microsecond precision.
- bool [valid](#) ()
Verify the delay object is valid and ready to use.
- void [_wake](#) ()
Internal ISR callback to wake blocked task (internal use only)

16.70.1 Detailed Description

High-resolution microsecond delay timer class.

Provides blocking delays with microsecond precision by allocating and using a hardware system timer. The timer is automatically managed - allocated on construction and freed on destruction.

This class uses a hardware timer from the processor's timer pool to provide precise microsecond delays. The timer is allocated when the object is created and remains dedicated to this object until it is destroyed. An internal semaphore is used to block the calling task during the delay period.

Note

Hardware timers are a limited resource. Always verify successful allocation by checking [valid\(\)](#) after construction.

Thread Safety

Each [DelayObject](#) should be used by a single task. For multi-task use, create separate [DelayObject](#) instances or protect with a mutex.

Performance Considerations

- Object creation allocates a timer (overhead on first use)
- Use static or member variables to avoid repeated allocation
- Very short delays (<10us) have relatively higher overhead
- Delay accuracy affected by interrupt latency and system load

Example: Basic Usage

```
DelayObject delay;

if (delay.valid()) {
    delay.DelayUsec(500); // 500 microsecond delay
}
```

Example: Pulse Generation

```
static DelayObject pulseDelay;

void generatePulse(int pin, uint32_t pulseWidthUs) {
    Pins[pin].set(); // Set pin high
    pulseDelay.DelayUsec(pulseWidthUs); // Precise pulse width
    Pins[pin].clr(); // Set pin low
}
```

Example: Protocol Timing

```

static DelayObject protocolDelay;

void sendI2CStart() {
    SDA.set();
    protocolDelay.DelayUsec(5);    // Setup time
    SCL.set();
    protocolDelay.DelayUsec(5);    // High time
    SDA.clr();
    protocolDelay.DelayUsec(5);    // Hold time
}

```

See also

[Interval Timer](#) For non-blocking periodic timers

[Stopwatch Timer](#) For measuring elapsed time

[OSTimeDly\(\)](#) For standard RTOS delays

16.70.2 Constructor & Destructor Documentation**16.70.2.1 DelayObject()**

```

DelayObject::DelayObject (
    int Timer = FIRST_UNUSED_TIMER)

```

Construct a high-resolution delay timer object.

Allocates a hardware system timer from the processor's timer pool. The timer remains allocated for the lifetime of this object.

Parameters

<i>Timer</i>	Timer number to use. Default is FIRST_UNUSED_TIMER which automatically selects the first available timer. Specific timer numbers can be used for advanced applications but this is not recommended for general use.

Note

Always check [valid\(\)](#) after construction to ensure a timer was successfully allocated.

Warning

If no timers are available, the object will be invalid and [DelayUsec\(\)](#) should not be called. Check [valid\(\)](#) first.

Example

```

DelayObject delay; // Allocates first free timer
if (!delay.valid()) {
    iprintf("ERROR: No hardware timers available\n");
    return;
}

```

See also

[valid\(\)](#) To verify successful timer allocation

FIRST_UNUSED_TIMER Default timer selection constant

16.70.2.2 ~DelayObject()

```

DelayObject::~~DelayObject ()

```

Destructor - releases the allocated hardware timer.

Automatically frees the hardware timer back to the system pool when the object is destroyed or goes out of scope.

Note

The timer is freed even if a delay is in progress, though this should be avoided in normal operation.

16.70.3 Member Function Documentation

16.70.3.1 `_wake()`

```
void DelayObject::_wake () [inline]
```

Internal ISR callback to wake blocked task (internal use only)

This function is called by the timer interrupt service routine when the delay period expires. It posts the internal semaphore to unblock the task waiting in [DelayUsec\(\)](#).

Warning

This function is for internal use only and should never be called by user code. It is public only for ISR access.

Note

The leading underscore indicates internal/private usage

16.70.3.2 `DelayUsec()`

```
void DelayObject::DelayUsec (
    uint32_t usec)
```

Perform a blocking delay with microsecond precision.

Blocks the calling task for the specified number of microseconds using a hardware timer. The task is suspended until the timer expires.

Parameters

<i>usec</i>	Number of microseconds to delay (range: 0 to 4,294,967,295)

Precondition

The [DelayObject](#) must be valid (check with [valid\(\)](#))

Postcondition

The calling task resumes after approximately usec microseconds

Warning

Do not call this function if [valid\(\)](#) returns false

Very short delays (<10us) may have reduced accuracy due to function call and setup overhead

Note

The actual delay may be slightly longer than requested due to:

- Interrupt latency
- Task scheduling overhead
- System load

Delay Range Guidelines

- 1-10 us: Use only if hardware timing is critical
- 10-999 us: Ideal range for high-resolution delays
- 1-20 ms: Consider using [OSTimeDly\(\)](#) instead for efficiency
- >20 ms: Strongly recommend [OSTimeDly\(\)](#) to free hardware timer

Example: Short Delays

```
static DelayObject delay;
delay.DelayUsec(50);    // 50 microsecond delay
delay.DelayUsec(100);   // 100 microsecond delay
```

Example: Millisecond Delays

```
static DelayObject delay;
delay.DelayUsec(1000);  // 1 millisecond = 1000 microseconds
delay.DelayUsec(2500);  // 2.5 milliseconds
```

See also

[OSTimeDly\(\)](#) For delays ≥ 20 ms (more efficient)

[valid\(\)](#) Check before calling this function

16.70.3.3 [valid\(\)](#)

```
bool DelayObject::valid () [inline]
```

Verify the delay object is valid and ready to use.

Checks whether a hardware timer was successfully allocated during construction. Must be called before using [DelayUsec\(\)](#) to avoid undefined behavior.

Returns

true if a hardware timer was successfully allocated

false if no timer was available or allocation failed

Postcondition

If false is returned, [DelayUsec\(\)](#) must not be called

Note

A return value of false indicates all hardware timers are currently allocated. Consider:

- Freeing unused [DelayObject](#) instances
- Using [OSTimeDly\(\)](#) for less critical timing
- Redesigning to share timers between operations

Example: Proper Error Checking

```
DelayObject delay;

if (delay.valid()) {
    delay.DelayUsec(100); // Safe to use
} else {
    iprintf("WARNING: No hardware timers available\n");
    OSTimeDly(1); // Fallback to standard delay
}
```

Example: Critical Operation Guard

```
static DelayObject criticalDelay;

void performCriticalTiming() {
    if (!criticalDelay.valid()) {
        iprintf("FATAL: Cannot perform critical timing\n");
        return; // Abort operation
    }
    // Proceed with timing-critical code
    criticalDelay.DelayUsec(50);
}
```

See also

[DelayObject\(\)](#) Constructor that allocates the timer

The documentation for this class was generated from the following file:

- [HiResDelay.h](#)

16.71 DHCP::DhcpInfo Struct Reference

DHCP Info.

```
#include <dhcpd.h>
```

Public Attributes

- **IPADDR4 netmask**
Netmask for the lease offered.
- **IPADDR4 gateway**
gateway to be offered
- **IPADDR4 dns_1**
primary DNS Server
- **IPADDR4 dns_2**
secondary DNS Server
- **IPADDR4 logServ**
Syslog Server.
- **IPADDR4 smtpServ**
SMTP Server.
- **IPADDR4 ntpServ**
NTP Server.
- const char * **domain_name**
Domain name for hosts.
- char * **hostname**
Hostname.
- char * **tftp_name**
tftp server name/dotted ip, null terminated
- char * **bootfile**
tftp boot file, null terminated
- uint16_t **bonusLength**
Length of any additional options, pre written.
- uint8_t * **bonusOpts**
additional options, pre written

16.71.1 Detailed Description

DHCP Info.

The documentation for this struct was generated from the following file:

- dhcpd.h

16.72 DHCP::DhcpLeaseData Struct Reference

Lease Data.

```
#include <dhcpd.h>
```

Public Attributes

- **IPADDR4 ip**
the IP Address to be offered
- **MACADR mac**
source mac address of the request
- uint32_t **expiration**
The tick that the lease expires on.
- char **hostname** [DHCP_SERV_MAX_HOSTNAME_LEN+1]
buffer for client hostname

16.72.1 Detailed Description

Lease Data.

The documentation for this struct was generated from the following file:

- `dhcpcd.h`

16.73 DHCP::DhcpLeaseRequest Struct Reference

Lease Request.

```
#include <dhcpcd.h>
```

Public Attributes

- [IPADDR4](#) **ip**
the IP Address to be offered
- [MACADR](#) **mac**
source mac address of the request
- `uint32_t` **duration**
The duration of the lease, in seconds.
- `uint32_t` **xid**
transaction id of the [DHCP](#) message
- `char` **hostname** [[DHCP_SERV_MAX_HOSTNAME_LEN](#)+1]
buffer for client hostname

16.73.1 Detailed Description

Lease Request.

The documentation for this struct was generated from the following file:

- `dhcpcd.h`

16.74 DhcpObject Class Reference

[DHCP](#) client class.

```
#include <dhcpcclient.h>
```

Inherits [TimeOutElement](#).

Public Member Functions

- `void` [StartDHCP](#) ()
Manually start the [DHCP](#) Client.
- `void` [StopDHCP](#) ()
Manually stop the [DHCP](#) client and release the [DHCP](#) lease.
- `void` [RestartDHCP](#) ()
Manually stop are resetart the [DHCP](#) client.
- `bool` [bDoFallBack](#) ()
Returns true if a [DHCP](#) fallback event has occurred.
- `void` [RenewDHCP](#) ()
Manually force a [DHCP](#) renew of a lease.
- `void` [RebindDHCP](#) ()
Manually force a [DHCP](#) rebind of a lease.
- `BOOL` [ValidDhcpLease](#) ()
Returns the status of a [DHCP](#) lease.
- `uint32_t` [GetRemainingDhcpLeaseTime](#) ()

Returns number of seconds remaining for current lease.

- uint32_t [GetDhcpRenewTime](#) ()

Returns number of seconds remaining before the lease will be renewed.

- uint32_t [GetDhcpRebindTime](#) ()

Returns number of seconds remaining before the current lease will have to rebind.

- uint32_t [GetDhcpExpirationTime](#) ()

Returns number of seconds remaining before the current lease will expire.

- int32_t [GetDHCPState](#) ()

Returns current state of the [DHCP](#) lease.

16.74.1 Detailed Description

[DHCP](#) client class.

16.74.2 Member Function Documentation

16.74.2.1 bDoFallBack()

```
bool DhcpObject::bDoFallBack ()
```

Returns true if a [DHCP](#) fallback event has occurred.

The [init\(\)](#) function called at the beginning of an application will handle [DHCP](#) client services for all network interfaces.

It is rare that you would need to call these functions manually.

If the fallback functionality has been enabled in the device configuration, this function can be used to determine if a fallback to a static IP settings has occurred.

Returns

True if a fallback condition has occurred.

16.74.2.2 GetDhcpExpirationTime()

```
uint32_t DhcpObject::GetDhcpExpirationTime () [inline]
```

Returns number of seconds remaining before the current lease will expire.

Returns

The number of seconds remaining before the current lease will expire.

See also

[GetRemainingDhcpLeaseTime\(\)](#), [GetDhcpRenewTime\(\)](#), [GetDhcpRebindTime\(\)](#)

16.74.2.3 GetDhcpRebindTime()

```
uint32_t DhcpObject::GetDhcpRebindTime () [inline]
```

Returns number of seconds remaining before the current lease will have to rebind.

Returns

The number of seconds remaining before the current lease will have to rebind.

See also

[GetRemainingDhcpLeaseTime\(\)](#), [GetDhcpRenewTime\(\)](#), [GetDhcpExpirationTime\(\)](#)

16.74.2.4 GetDhcpRenewTime()

```
uint32_t DhcpObject::GetDhcpRenewTime () [inline]
```

Returns number of seconds remaining before the lease will be renewed.

Returns

The number of seconds remaining before the lease will be renewed.

See also

[GetRemainingDhcpLeaseTime\(\)](#), [GetDhcpRebindTime\(\)](#), [GetDhcpExpirationTime\(\)](#)

16.74.2.5 GetDHCPState()

```
int32_t DhcpObject::GetDHCPState ()
```

Returns current state of the [DHCP](#) lease.

Returns

[DHCP State](#)

See also

[ValidDhcpLease\(\)](#)

16.74.2.6 GetRemainingDhcpLeaseTime()

```
uint32_t DhcpObject::GetRemainingDhcpLeaseTime ()
```

Returns number of seconds remaining for current lease.

Returns

The number of seconds remaining for current lease.

See also

[GetDhcpRenewTime\(\)](#), [GetDhcpRebindTime\(\)](#), [GetDhcpExpirationTime\(\)](#)

16.74.2.7 RebindDHCP()

```
void DhcpObject::RebindDHCP ()
```

Manually force a [DHCP](#) rebind of a lease.

The [init\(\)](#) function called at the beginning of an application will handle [DHCP](#) client services for all network interfaces.

It is rare that you would need to call these functions manually.

See also

[RenewDHCP\(\)](#)

16.74.2.8 RenewDHCP()

```
void DhcpObject::RenewDHCP ()
```

Manually force a [DHCP](#) renew of a lease.

The [init\(\)](#) function called at the beginning of an application will handle [DHCP](#) client services for all network interfaces.

It is rare that you would need to call these functions manually.

See also

[RebindDHCP\(\)](#)

16.74.2.9 RestartDHCP()

```
void DhcpObject::RestartDHCP ()
```

Manually stop are resetar the [DHCP](#) client.

The [init\(\)](#) function called at the beginning of an application will handle [DHCP](#) client services for all network interfaces. It is rare that you would need to call these functions manually.

See also

[DhcpObject::StartDHCP\(\)](#), [DhcpObject::StopDHCP\(\)](#)

16.74.2.10 StartDHCP()

```
void DhcpObject::StartDHCP ()
```

Manually start the [DHCP](#) Client.

The [init\(\)](#) function called at the beginning of an application will handle [DHCP](#) client services for all network interfaces. It is rare that you would need to call these functions manually.

This function will start the [DHCP](#) client and return immediately. Check the [DHCP](#) status to verify you have successfully obtained a lease via [GetDHCPState\(\)](#)

See also

[DhcpObject::StopDHCP\(\)](#)

16.74.2.11 StopDHCP()

```
void DhcpObject::StopDHCP ()
```

Manually stop the [DHCP](#) client and release the [DHCP](#) lease.

The [init\(\)](#) function called at the beginning of an application will handle [DHCP](#) client services for all network interfaces. It is rare that you would need to call these functions manually.

See also

[DhcpObject::StartDHCP\(\)](#)

16.74.2.12 ValidDhcpLease()

```
BOOL DhcpObject::ValidDhcpLease ()
```

Returns the status of a [DHCP](#) lease.

Returns

True if the interface has a valid lease.

See also

[RenewDHCP\(\)](#)

The documentation for this class was generated from the following file:

- [dhcpclient.h](#)

16.75 DiagCallBack Class Reference

Callback-based diagnostic item that calls a custom function to generate diagnostic content.

```
#include <diagnostics.h>
```

Inherits [DiagItemClass](#).

Public Member Functions

- [DiagCallBack](#) (const char *name, [diagcallback](#) *pFunc)

Constructor for callback-based diagnostic item.

16.75.1 Detailed Description

Callback-based diagnostic item that calls a custom function to generate diagnostic content.

16.75.2 Constructor & Destructor Documentation

16.75.2.1 DiagCallBack()

```
DiagCallBack::DiagCallBack (
    const char * name,
    diagcallback * pFunc) [inline]
```

Constructor for callback-based diagnostic item.

Parameters

<i>name</i>	Name of the diagnostic item that will appear in the JSON output
<i>pFunc</i>	Pointer to the callback function that will generate the diagnostic content

The documentation for this class was generated from the following file:

- diagnostics.h

16.76 DiagItemClass Class Reference

Base class for all diagnostic items.

```
#include <diagnostics.h>
```

Inherited by [AcmeServletObject](#), [DiagCallBack](#), [DiagPtrMon](#), [DiagStrMon](#), and [DiagVarMon](#).

16.76.1 Detailed Description

Base class for all diagnostic items.

The documentation for this class was generated from the following file:

- diagnostics.h

16.77 DiagPtrMon Class Reference

Monitor and report pointer values.

```
#include <diagnostics.h>
```

Inherits [DiagItemClass](#).

Public Member Functions

- [DiagPtrMon](#) (const char *name, void *p)
Constructor for monitoring pointer values.
- virtual void **ServeContent** (int fd)
Output the pointer value as a hexadecimal string.

16.77.1 Detailed Description

Monitor and report pointer values.

16.77.2 Constructor & Destructor Documentation

16.77.2.1 DiagPtrMon()

```
DiagPtrMon::DiagPtrMon (  
    const char * name,  
    void * p) [inline]
```

Constructor for monitoring pointer values.

Parameters

<i>name</i>	Display name for this pointer in the diagnostic output
<i>p</i>	Pointer value to monitor

The documentation for this class was generated from the following file:

- `diagnostics.h`

16.78 DiagStrMon Class Reference

Monitor and report string values.

```
#include <diagnostics.h>
```

Inherits [DiagItemClass](#).

Public Member Functions

- [DiagStrMon](#) (const char *name, const char *pstr)
Constructor for monitoring C-style string.
- [DiagStrMon](#) (const char *name, [NBString](#) &s)
Constructor for monitoring [NBString](#) object.
- virtual void **ServeContent** (int fd)
Output the string value to the diagnostic stream.

16.78.1 Detailed Description

Monitor and report string values.

16.78.2 Constructor & Destructor Documentation

16.78.2.1 DiagStrMon() [1/2]

```
DiagStrMon::DiagStrMon (  
    const char * name,  
    const char * pstr) [inline]
```

Constructor for monitoring C-style string.

Parameters

<i>name</i>	Display name for this string in the diagnostic output
<i>pstr</i>	Pointer to the C-style string to monitor

16.78.2.2 DiagStrMon() [2/2]

```
DiagStrMon::DiagStrMon (  
    const char * name,  
    NBString & s) [inline]
```

Constructor for monitoring [NBString](#) object.

Parameters

<i>name</i>	Display name for this string in the diagnostic output
<i>s</i>	Reference to the NBString object to monitor

The documentation for this class was generated from the following file:

- `diagnostics.h`

16.79 DiagVarMon Class Reference

Monitor and report the value of numeric variables.

#include <diagnostics.h>

Inherits [DiagItemClass](#).

Public Member Functions

- [DiagVarMon](#) (const char *name, uint32_t &v)
Constructor for monitoring numeric variables.
- virtual void **ServeContent** (int fd)
Pure virtual function to output diagnostic content.

16.79.1 Detailed Description

Monitor and report the value of numeric variables.

16.79.2 Constructor & Destructor Documentation

16.79.2.1 DiagVarMon()

```
DiagVarMon::DiagVarMon (
    const char * name,
    uint32_t & v) [inline]
```

Constructor for monitoring numeric variables.

Creates a diagnostic monitor for various numeric data types. The constructor automatically determines the appropriate formatting based on the variable type:

- Signed integers are displayed as signed decimal numbers
- Unsigned integers are displayed as unsigned decimal numbers
- Floating point numbers are displayed in general format (g)

Parameters

<i>name</i>	Display name for this variable in the diagnostic output
<i>v</i>	Reference to the variable to monitor

The documentation for this class was generated from the following file:

- diagnostics.h

16.80 NB::Wifi::driverStatusStruct Struct Reference

#include <wifiDriver.h>

Public Attributes

- uint8_t **connected**
1 if connected, 0 if disconnected
- uint8_t **ssidLength**
Length of the SSID string.
- uint16_t **txPower**
Transmit power.
- int16_t **rss**

- *Recieved Signal Strength Indicator.*
- uint8_t **band**
Frequency band.
- uint8_t **channel**
802.11 channel
- uint32_t **maxTxRate**
Max transmit data throughput.
- uint8_t **security**
Security used: WEP, WPA, WPA2, open, unknown.
- uint8_t **cipher**
refer to [OPTIONLISTCIPH](#) for defenitions
- [MACADR](#) **bssid**
Basic Service Set Identifier.
- uint8_t **bssType**
refer to [OPTIONLISTBSSTYPE](#) for defenitions
- char **ssid** [SSID_MAX_LEN+1]
Placeholder for char array pointer.
- uint32_t **tickLastUpdated**
CPU tick count of when this struct was last updated.

16.80.1 Detailed Description

A struct used to describe the status of the current network connection.
The documentation for this struct was generated from the following file:

- wifiDriver.h

16.81 dspiDMAStruct Struct Reference

DSPI DMA structure.
`#include <dspi.h>`

Public Attributes

- bool **enabled**
Whether or not the driver is using DMA for the transfer.
- uint8_t **byteCount**
The bytes transferred per DMA call.
- bool **rxPresent**
Is a receive buffer present?
- bool **txPresent**
Is a transmit buffer present?
- dspistruct **savedDSPI**
- bool **csState**
- uint8_t **bytesLastRead**
- bool **isThereExtraRxBytes**
- DmaCh * **rxDma**
- DmaCh * **txDma**

- LPSPI_Type **savedSPI**
- bool **isPcsContinuous**
- uint32_t **extraBytesTCR**
Used to write SPI TCR via DMA for extra 1 or 2 data byte transfer. Used to de-assert CS if DEASSERT_AFTER_LAST.
- uint32_t **extraThirdByteTCR**
Used to write SPI TCR via DMA for extra 3rd data byte transfer. Used to de-assert CS if DEASSERT_AFTER_LAST.
- uint32_t **pcsTCR**
Used to write SPI TCR via DMA to de-assert CS if DEASSERT_AFTER_LAST at the end of the transfer.
- DmaTcd **softwareTCD** [6]
SoftwareTCD, used for scatter/gather to de-assert CS, internally used.

16.81.1 Detailed Description

DSPI DMA structure.

The documentation for this struct was generated from the following files:

- coldfire/cpu/MCF5441X/include/dspi.h
- cortex-m7/cpu/MIMXRT10xx/include/dspi.h

16.82 dspiDriverStruct Struct Reference

Major variables/configurations used for a DSPI transfer.

```
#include <dspi.h>
```

Public Attributes

- volatile uint8_t * **pDSPIRxbuf**
Track memory location where data will be read or written to the peripheral.
- volatile uint8_t * **pDSPITxbuf**
Track memory location where data will be read or written to the peripheral.
- uint8_t **BitsPerQueue**
Number of bits per transfer, (value = 8 - 32)
- uint32_t **DSPI_SizeLeft**
Number of bytes left in the transfer.
- uint16_t **Command_Mask**
Partial configuration for the queue's command reg.
- [csReturnType](#) **csReturnToInactive**
- OS_SEM * **DSPI_Sem**
Pointer to an external semaphore provided by [DSPIStart\(\)](#)
- volatile uint8_t **DSPI_INT_STATUS**
Status of the spi device.
- uint32_t **WordsToWrite**
Number of words to write to the command/TX Queues and Read from RX.
- uint32_t **LastWordsToWrite**
The amount from the previous ISR.
- volatile BOOL **DSPIfinished**
- [dspiDMAStruct](#) **dma**

16.82.1 Detailed Description

Major variables/configurations used for a DSPI transfer.

The documentation for this struct was generated from the following file:

- coldfire/cpu/MCF5441X/include/dspi.h

16.83 DSPIModule Class Reference

DSPIModule is a SPI communications driver. It is an object based driver, which allows for low overhead multiplexing between peripherals with different bus configurations.

```
#include <dspi.h>
```

Public Member Functions

- **DSPIModule** (uint8_t **SPIModule**)
*The minimum **DSPIModule** Constructor. Requires that configuration settings be applied after construction before use.*
- **DSPIModule** (uint8_t **SPIModule**, uint32_t baudRateInBps, uint8_t transferSizeInBits=8, uint8_t peripheralChipSelects=0x00, uint8_t chipSelectPolarity=0x0F, uint8_t clockPolarity=0, uint8_t clockPhase=1, BOOL doutHiz=TRUE, uint8_t csToClockDelay=0, uint8_t delayAfterTransfer=0)
***DSPIModule** Full **DSPIModule** Constructor. Will initialize all hardware settings.*
- uint8_t **Init** (uint32_t baudRateInBps=2000000, uint8_t transferSizeInBits=8, uint8_t peripheralChipSelects=0x00, uint8_t chipSelectPolarity=0x0F, uint8_t clockPolarity=0, uint8_t clockPhase=1, BOOL doutHiz=TRUE, uint8_t csToClockDelay=0, uint8_t delayAfterTransfer=0)
***DSPIModule** Full **DSPIModule** Constructor. Will initialize all hardware settings.*
- uint8_t **Start** (uint8_t *transmitBufferPtr, volatile uint8_t *receiveBufferPtr, uint32_t byteCount, int csReturnToInactive=**DEASSERT_AFTER_LAST**)
Start begins a SPI transaction using the provided buffers.
- uint8_t **Tx** (uint8_t *transmitBufferPtr, uint32_t byteCount, int csReturnToInactive=**DEASSERT_AFTER_LAST**)
Tx begins a transmit only SPI transaction using the provided buffer. Silently discards the received data.
- uint8_t **Rx** (uint8_t *receiveBufferPtr, uint32_t byteCount, int csReturnToInactive=**DEASSERT_AFTER_LAST**)
Rx begins a receive only SPI transaction using the provided buffer. Holds output line low during transfers.
- bool **EnableDMA** (bool enableDMA=true)
EnableDMA configures whether DMA is allowed for qualifying transactions.
- bool **DisableDMA** ()
DisableDMA disables DMA for all transactions.
- bool **RegisterSem** (OS_SEM *finishedSem)
Registers a semaphore to be posted to upon completion of transactions.
- bool **ClrSem** ()
Clears the semaphore registration.
- OS_SEM * **GetSem** ()
Gets a pointer to the registered Semaphore.
- bool **Done** ()
Returns whether the previously started transaction is complete.
- uint32_t **GetActualBaudrate** ()
Returns the actual bus speed to be used.
- bool **SetCS** (uint8_t CS)
Modifies the chip select configuration for the driver context such that only the requested chip select is active during transfers.

Static Public Member Functions

- static BOOL **Done** (uint8_t **SPIModule**)
Returns whether the previously started transaction for a given hardware module is complete.

Friends

- uint8_t [DSPIInit](#) (uint8_t [SPIModule](#)=[DEFAULT_DSPI_MODULE](#), uint32_t Baudrate=2000000, uint8_t QueueBitSize=8, uint8_t CS=0x00, uint8_t CSPol=0x0F, uint8_t ClkPolarity=0, uint8_t ClkPhase=1, BOOL Douthiz=TRUE, uint8_t QCD=0, uint8_t DTL=0)

Initialize a DSPI module.

16.83.1 Detailed Description

[DSPIModule](#) is a SPI communications driver. It is an object based driver, which allows for low overhead multiplexing between peripherals with different bus configurations.

16.83.2 Constructor & Destructor Documentation**16.83.2.1 DSPIModule() [1/2]**

```
DSPIModule::DSPIModule (
    uint8_t SPIModule)
```

The minimum [DSPIModule](#) Constructor. Requires that configuration settings be applied after construction before use.

Parameters

SPIModule	The DSPI module to use, from 1 to 3. Default is 1.

See also

[DSPIModule::DSPIModule\(uint8_t \[SPIModule\]\(#\), uint32_t baudRateInBps, uint8_t transferSizeInBits, uint8_t peripheralChipSelects\)](#)

16.83.2.2 DSPIModule() [2/2]

```
DSPIModule::DSPIModule (
    uint8_t SPIModule,
    uint32_t baudRateInBps,
    uint8_t transferSizeInBits = 8,
    uint8_t peripheralChipSelects = 0x00,
    uint8_t chipSelectPolarity = 0x0F,
    uint8_t clockPolarity = 0,
    uint8_t clockPhase = 1,
    BOOL douthiz = TRUE,
    uint8_t csToClockDelay = 0,
    uint8_t delayAfterTransfer = 0)
```

[DSPIModule](#) Full [DSPIModule](#) Constructor. Will initialize all hardware settings.

Parameters

SPIModule	The DSPI module to use, from 1 to 3. Default is 1.
baudRateInBps	Maximum master clock frequency for SPI Bus in bits per second.
transferSizeInBits	Bit width of the transfers to be made.
peripheralChipSelects	Bit mask of which chipselects are active during transfers.
chipSelectPolarity	Bit mask of the inactive state of chipselects.
clockPolarity	Inactive level for the clock signal.
clockPhase	Clock phase: • 0: data captured on leading edge of DSPI_CLK, changed on following edge. • 1: data changed on leading edge of DSPI_CLK, captured on following edge.

Parameters

<i>doutHiz</i>	Data output is high impedance between transfers (instead of driven).
<i>csToClockDelay</i>	Delay from chip select to valid clock. The default of 0 will set the delay as close to 1/2 DSPI_CLK without going under, keeping with the interface to QSPI driver.
<i>delayAfterTransfer</i>	Delay between data transfers. The default of 0 is 17/(system clock / 2), in keeping with interface to QSPI

See also

[DSPIModule::DSPIModule\(uint8_t SPIModule \)](#)

DSPIModule::Init(uint8_t SPIModule, uint32_t baudRateInBps, uint8_t transferSizeInBits, uint8_t peripheral↵
ChipSelects, uint8_t chipSelectPolarity, uint8_t clockPolarity, uint8_t clockPhase, BOOL doutHiz, uint8_t cs↵
ToClockDelay, uint8_t delayAfterTransfer)

16.83.3 Member Function Documentation

16.83.3.1 ClrSem()

```
bool DSPIModule::ClrSem () [inline]
```

Clears the semaphore registration.

Returns

Returns whether the registration was successfully cleared.

16.83.3.2 DisableDMA()

```
bool DSPIModule::DisableDMA () [inline]
```

DisableDMA disables DMA for all transactions.

Returns

Returns whether DMA is allowed

16.83.3.3 Done() [1/2]

```
bool DSPIModule::Done () [inline]
```

Returns whether the previously started transaction is complete.

Returns

Whether the previously started transaction is complete

16.83.3.4 Done() [2/2]

```
static BOOL DSPIModule::Done (
    uint8_t SPIModule) [static]
```

Returns whether the previously started transaction for a given hardware module is complete.

Parameters

SPIModule	The hardware instance to check the transaction state for.

Returns

Whether the previously started transaction is complete

16.83.3.5 EnableDMA()

```
bool DSPIModule::EnableDMA (  
    bool enableDMA = true)
```

EnableDMA configures whether DMA is allowed for qualifying transactions.

Parameters

<i>enableDMA</i>	Whether DMA is allowed

Returns

Returns whether DMA is allowed

16.83.3.6 GetActualBaudrate()

```
uint32_t DSPIModule::GetActualBaudrate () [inline]
```

Returns the actual bus speed to be used.

Returns

The actual bus speed to be used.

16.83.3.7 GetSem()

```
OS_SEM * DSPIModule::GetSem () [inline]
```

Gets a pointer to the registered Semaphore.

Returns

Pointer to the registered Semaphore

16.83.3.8 Init()

```
uint8_t DSPIModule::Init (
    uint32_t baudRateInBps = 2000000,
    uint8_t transferSizeInBits = 8,
    uint8_t peripheralChipSelects = 0x00,
    uint8_t chipSelectPolarity = 0x0F,
    uint8_t clockPolarity = 0,
    uint8_t clockPhase = 1,
    BOOL doutHiz = TRUE,
    uint8_t csToClockDelay = 0,
    uint8_t delayAfterTransfer = 0)
```

[DSPIModule](#) Full [DSPIModule](#) Constructor. Will initialize all hardware settings.

param [SPIModule](#) The instance number of the hardware module to use with this driver context.

Parameters

<i>baudRateInBps</i>	Maximum master clock frequency for SPI Bus
<i>transferSizeInBits</i>	Bit width of the transfers to be made
<i>peripheralChipSelects</i>	Bit mask of which chipselects are active during transfers
<i>chipSelectPolarity</i>	Bit mask of the inactive state of chipselects
<i>clockPolarity</i>	Inactive level for the clock signal
<i>clockPhase</i>	• 0: data captured on leading edge of DSPI_CLK, changed on following edge • 1: data changed on leading edge of DSPI_CLK, captured on following edge

Parameters

<i>doutHiz</i>	Data output is high impedance between transfers (instead of driven)

Parameters

<i>csToClockDelay</i>	Delay form chip select to valid clock. The default of 0 will set the delay as close to 1/2 DSPI_CLK without going under, keeping with the interface to QSPI driver.
<i>delayAfterTransfer</i>	Delay between data transfers. The default of 0 is 17/(system clock / 2), in keepin with interface to QSPI

See also

[DSPIModule::DSPIModule\(uint8_t SPIModule, uint32_t baudRateInBps, uint8_t transferSizeInBits, uint8_t peripheralChipSelect\)](#)

16.83.3.9 RegisterSem()

```
bool DSPIModule::RegisterSem (
    OS_SEM * finishedSem)
```

Registers a semaphore to be posted to upon completion of transactions.

Parameters

<i>finishedSem</i>	A pointer to the semaphore to be posted to.

Returns

Returns whether the semaphore was successfully registered.

16.83.3.10 Rx()

```
uint8_t DSPIModule::Rx (
    uint8_t * receiveBufferPtr,
    uint32_t byteCount,
    int csReturnToInactive = DEASSERT_AFTER_LAST) [inline]
```

Rx begins a receive only SPI transaction using the provided buffer. Holds output line low during transfers.

Parameters

<i>receiveBufferPtr</i>	A pointer to the buffer to write received data to. If the buffer is NULL, then receive data will silently be discarded.
<i>byteCount</i>	The number of bytes to send and receive in this transaction. The value must be a multiple of the number of bytes in each individual transfer.
<i>csReturnToInactive</i>	Configures when the ChipSelect that is used for the transaction should be deasserted and return to being inactive.

See also

[csReturnType](#)

Returns

Non-zero on error

16.83.3.11 SetCS()

```
bool DSPIModule::SetCS (
    uint8_t CS) [inline]
```

Modifies the chip select configuration for the driver context such that only the requested chip select is active during transfers.

Parameters

CS	The chip select number to assert during transfers.

Returns

Whether the active chip select was successfully updated.

16.83.3.12 Start()

```
uint8_t DSPIModule::Start (
    uint8_t * transmitBufferPtr,
    volatile uint8_t * receiveBufferPtr,
    uint32_t byteCount,
    int csReturnToInactive = DEASSERT_AFTER_LAST)
```

Start begins a SPI transaction using the provided buffers.

Parameters

<i>transmitBufferPtr</i>	A pointer to the buffer containing data to be sent out. If the buffer is NULL, the bus will be held low instead.
<i>receiveBufferPtr</i>	A pointer to the buffer to write received data to. If the buffer is NULL, then receive data will silently be discarded.
<i>byteCount</i>	The number of bytes to send and receive in this transaction. The value must be a multiple of the number of bytes in each individual transfer.
<i>csReturnToInactive</i>	Configures when the ChipSelect that is used for the transaction should be deasserted and return to being inactive.

See also

[csReturnType](#)

Returns

Non-zero on error

16.83.3.13 Tx()

```
uint8_t DSPIModule::Tx (
    uint8_t * transmitBufferPtr,
    uint32_t byteCount,
    int csReturnToInactive = DEASSERT_AFTER_LAST) [inline]
```

Tx begins a transmit only SPI transaction using the provided buffer. Silently discards the received data.

Parameters

<i>transmitBufferPtr</i>	A pointer to the buffer containing data to be sent out. If the buffer is NULL, the bus will be held low instead.
<i>byteCount</i>	The number of bytes to send and receive in this transaction. The value must be a multiple of the number of bytes in each individual transfer.
<i>csReturnToInactive</i>	Configures when the ChipSelect that is used for the transaction should be deasserted and return to being inactive.

See also

[csReturnType](#)

Returns

Non-zero on error

16.83.4 Friends And Related Symbol Documentation

16.83.4.1 DSPIInit

```
uint8_t DSPIInit (
    uint8_t SPIModule = DEFAULT_DSPI_MODULE,
    uint32_t Baudrate = 2000000,
    uint8_t QueueBitSize = 8,
    uint8_t CS = 0x00,
    uint8_t CSPol = 0x0F,
    uint8_t ClkPolarity = 0,
    uint8_t ClkPhase = 1,
    BOOL DoutHiz = TRUE,
    uint8_t QCD = 0,
    uint8_t DTL = 0) [friend]
```

Initialize a DSPI module.

Notes:

- The maximum baud rate is CPU_CLOCK / 3
- Will initialize to the highest available baud rate that does not exceed the maximum
- If configured for 8 bits per transfer then the data must be uint8_t aligned
- If configured for greater than 8 bits per transfer then the data must be uint16_t aligned
- If configured for greater than 16 bits per transfer then the data must be uint32_t aligned
- Chip select is based on chipSelectPolarity
- 0 data captured leading edge of DSPI_CLK, changed following edge.
- 1 data changed leading edge of DSPI_CLK, captured following edge.
- 0 default is as close to 1/2 DSPI_CLK without going under, keeping with the interface to SPI
- 0 default is 17/(system clock / 2), in keeping with interface to QSPI

Parameters

<i>SPIModule</i>	SPI module number, 0 - 1
<i>Baudrate</i>	Maximum baud rate requested
<i>QueueBitSize</i>	Number of bits per transfer: 8, 16 or 32
<i>CS</i>	SPI chip selects to use for transfer
<i>CSPol</i>	0 = inactive logic level low, 1 = high
<i>ClkPolarity</i>	0 = inactive logic level low, 1 = high
<i>ClkPhase</i>	0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge.
<i>DoutHiz</i>	Data output high impedance between transfers
<i>QCD</i>	Delay from chip select to valid clock (default is 0). QCD is a value in the QDLYR register and will change the delay between the assertion of the chip select and the start of the DSPI clock. Default setting of one half DSPI clk will be used if parameter is specified as 0x0 or not included.
<i>DTL</i>	DTL is a value in the QDLYR register and will change the delay following a transfer of a single uint16_t in the DSPI queue. Default reset value of 17/(fsys/2) will be used if parameter is specified 0 or not included.

Returns

Current stat of DSPI bus [MOD5441xDspiStates](#)

Notes:

- The maximum baud rate is CPU_CLOCK / 3
- Will initialize to the highest available baud rate that does not exceed the maximum
- If configured for 8 bits per transfer then the data must be uint8_t aligned
- If configured for greater than 8 bits per transfer then the data must be uint16_t aligned
- If configured for greater than 16 bits per transfer then the data must be uint32_t aligned
- Chip select is based on chipSelectPolarity
- 0 data captured leading edge of DSPI_CLK, changed following edge.
- 1 data changed leading edge of DSPI_CLK, captured following edge.
- 0 default is as close to 1/2 DSPI_CLK without going under, keeping with the interface to SPI
- 0 default is 17/(system clock / 2), in keeping with interface to QSPI

Parameters

SPIModule	SPI module number, 0 - 1
<i>Baudrate</i>	Maximum baud rate requested
<i>QueueBitSize</i>	Number of bits per transfer: 8, 16 or 32
<i>CS</i>	SPI chip selects to use for transfer
<i>CSPol</i>	0 = inactive logic level low, 1 = high
<i>ClkPolarity</i>	0 = inactive logic level low, 1 = high
<i>ClkPhase</i>	0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge.
<i>DoutHiz</i>	Data output high impedance between transfers
<i>QCD</i>	Delay from chip select to valid clock (default is 0)
<i>DTL</i>	Chip select mode dspicChipSelectMode

Returns

Current stat of DSPI bus [dspicState](#)

The documentation for this class was generated from the following file:

- coldfire/cpu/MCF5441X/include/dspi.h

16.84 Dynamic_IPADDR Class Reference

[Dynamic_IPADDR](#).

```
#include <config_netobj.h>
```

Inherits [config_value](#).

Public Member Functions

- virtual void [GetTextValue](#) (NBString &s)
- [Dynamic_IPADDR](#) (const char *name, int item, [I6Record](#) *pl6, const char *desc=NULL)
- virtual ConfigTestResult [TestNewValue](#) ([ParsedJsonDataSet](#) &pjs)
- virtual void [CommitTestedValue](#) (uint32_t permission_mask)
- void [AddToJsonOutObject](#) ([ParsedJsonDataSet](#) &jo, const char *name)
- virtual void [ExtendedSchema](#) (int fd, int indent, bool pretty)
- virtual void [GetTypeValue](#) (NBString &s)

Additional Inherited Members**Protected Member Functions inherited from [config_value](#)**

- [config_value](#) ([config_obj](#) &owner, const char *name, const char *desc)
Object constructor with the parent/owner leaf parameter.
- [config_value](#) (const char *name, const char *desc)
Object constructor.

16.84.1 Detailed Description

[Dynamic_IPADDR](#).

16.84.2 Constructor & Destructor Documentation**16.84.2.1 Dynamic_IPADDR()**

```
Dynamic_IPADDR::Dynamic_IPADDR (
    const char * name,
    int item,
    I6Record * pI6,
    const char * desc = NULL) [inline]
```

16.84.3 Member Function Documentation**16.84.3.1 AddToJsonOutObject()**

```
void Dynamic_IPADDR::AddToJsonOutObject (
    ParsedJsonDataSet & jo,
    const char * name)
```

16.84.3.2 CommitTestedValue()

```
virtual void Dynamic_IPADDR::CommitTestedValue (
    uint32_t permission_mask) [inline], [virtual]
```

16.84.3.3 ExtendedSchema()

```
virtual void Dynamic_IPADDR::ExtendedSchema (
    int fd,
    int indent,
    bool pretty) [inline], [virtual]
```

16.84.3.4 GetTextValue()

```
virtual void Dynamic_IPADDR::GetTextValue (
    NBString & s) [virtual]
```

Implements [config_value](#).

16.84.3.5 GetTypeValue()

```
virtual void Dynamic_IPADDR::GetTypeValue (
    NBString & s) [inline], [virtual]
```

16.84.3.6 TestNewValue()

```
virtual ConfigTestResult Dynamic_IPADDR::TestNewValue (
    ParsedJsonDataSet & pjs) [inline], [virtual]
```

The documentation for this class was generated from the following file:

- [config_netobj.h](#)

16.85 EBI_CS_cfg_t Struct Reference

Configuration structure for an External Bus Interface (EBI) chip select.

```
#include <ebi.h>
```

Public Attributes

- [uint8_t ncs_rd_setup](#)
- [uint8_t nrd_setup](#)
- [uint8_t ncs_wr_setup](#)
- [uint8_t nwe_setup](#)
- [uint8_t ncs_rd_pulse](#)
- [uint8_t nrd_pulse](#)
- [uint8_t ncs_wr_pulse](#)
- [uint8_t nwe_pulse](#)
- [uint16_t nrd_cycles](#)
- [uint16_t nwe_cycles](#)
- [uint8_t tdf_cycles](#)
- [EBI_CS_BusWidth_t busWidth](#)
- [EBI_CS_ByteAccess_t byteAccess](#)
- [EBI_CS_NWait_t nWait](#)
- [EBI_CS_WrMode_t wrMode](#)
- [EBI_CS_RdMode_t rdMode](#)

16.85.1 Detailed Description

Configuration structure for an External Bus Interface (EBI) chip select.

This structure defines all timing and control parameters for a single chip select on the External Bus Interface. It controls setup times, pulse widths, cycle lengths, and operational modes for interfacing with external memory or peripheral devices.

The timing parameters are specified in clock cycles and use formulas to extend the range of values that can be represented in the limited bit fields.

Note

All timing values are expressed in EBI clock cycles. Calculate actual times by dividing by your EBI clock frequency.

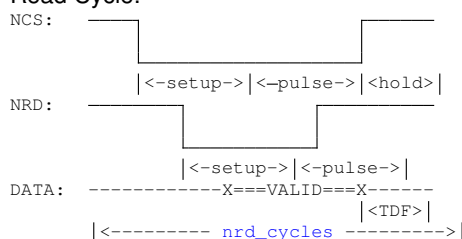
Setup + Pulse widths should not exceed the total cycle length. The hold time is implicitly: $\text{Hold} = \text{Total\_Cycles} - (\text{Setup} + \text{Pulse})$

Warning

Incorrect timing configuration can result in data corruption, device malfunction, or bus hangs. Always consult your external device datasheet for minimum timing requirements.

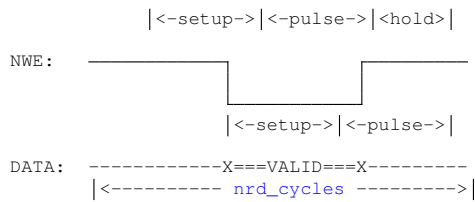
16.85.1.1 Timing Diagram Reference

Read Cycle:



Write Cycle:





See also

[EBI_CS_BusWidth_t](#)
[EBI_CS_ByteAccess_t](#)
[EBI_CS_NWait_t](#)
[EBI_CS_WrMode_t](#)
[EBI_CS_RdMode_t](#)

Expand for Example Usage

Examples

Example 1: Fast SRAM configuration (70ns access time)

```
// Assuming 150 MHz EBI clock (6.67 ns per cycle)
// SRAM requires: 10ns setup, 70ns pulse, 10ns hold

EBI_CS_cfg_t sramConfig = {
    // Read timing
    .ncs_rd_setup = 2,          // ~13ns (2 cycles)
    .nrd_setup = 2,            // ~13ns
    .ncs_rd_pulse = 11,        // ~73ns (11 cycles)
    .nrd_pulse = 11,           // ~73ns
    .nrd_cycles = 15,          // ~100ns total (15 cycles)

    // Write timing (same as read for this SRAM)
    .ncs_wr_setup = 2,         // ~13ns
    .nwe_setup = 2,            // ~13ns
    .ncs_wr_pulse = 11,        // ~73ns
    .nwe_pulse = 11,           // ~73ns
    .nwe_cycles = 15,          // ~100ns total

    // Additional parameters
    .tdf_cycles = 2,           // 2 cycles data float time
    .busWidth = EBI_BUS_WIDTH_16,
    .byteAccess = EBI_BYTE_ACCESS_WRITE,
    .nWait = EBI_NWAIT_DISABLED,
    .wrMode = EBI_WRITE_MODE_NWE,
    .rdMode = EBI_READ_MODE_NRD
};

iprintf("Fast SRAM configured: 70ns access time\r\n");
```

Example 2: Slow flash memory configuration (120ns access)

```
// Assuming 100 MHz EBI clock (10 ns per cycle)
// Flash requires: 20ns setup, 120ns pulse, 20ns hold

EBI_CS_cfg_t flashConfig = {
    // Read timing
    .ncs_rd_setup = 2,          // 20ns (2 cycles)
    .nrd_setup = 2,            // 20ns
    .ncs_rd_pulse = 12,        // 120ns (12 cycles)
    .nrd_pulse = 12,           // 120ns
    .nrd_cycles = 16,          // 160ns total (16 cycles)

    // Write timing (flash writes are slower)
    .ncs_wr_setup = 3,         // 30ns
    .nwe_setup = 3,            // 30ns
    .ncs_wr_pulse = 20,         // 200ns (longer for programming)
    .nwe_pulse = 20,           // 200ns
    .nwe_cycles = 26,          // 260ns total

    // Additional parameters
    .tdf_cycles = 3,           // 3 cycles data float
    .busWidth = EBI_BUS_WIDTH_8,
    .byteAccess = EBI_BYTE_ACCESS_SELECT,
    .nWait = EBI_NWAIT_FROZEN, // Flash can extend cycles
    .wrMode = EBI_WRITE_MODE_NWE,
    .rdMode = EBI_READ_MODE_NRD
};

iprintf("Flash memory configured with NWAIT support\r\n");
```

Example 3: Simple 8-bit I/O device (minimal timing)

```
// Assuming 120 MHz EBI clock (8.33 ns per cycle)
// Simple device with relaxed timing: 25ns setup, 50ns pulse

EBI_CS_cfg_t ioConfig = {
    // Read timing
    .ncs_rd_setup = 3,          // ~25ns (3 cycles)
    .nrd_setup = 3,            // ~25ns
    .ncs_rd_pulse = 6,         // ~50ns (6 cycles)
    .nrd_pulse = 6,            // ~50ns
    .nrd_cycles = 10,          // ~83ns total

    // Write timing (same as read)
    .ncs_wr_setup = 3,         // ~25ns
    .nwe_setup = 3,            // ~25ns
    .ncs_wr_pulse = 6,         // ~50ns
    .nwe_pulse = 6,            // ~50ns
    .nwe_cycles = 10,          // ~83ns total

    // Additional parameters
    .tdf_cycles = 1,           // Minimal data float
    .busWidth = EBI_BUS_WIDTH_8,
    .byteAccess = EBI_BYTE_ACCESS_SELECT,
    .nWait = EBI_NWAIT_DISABLED,
    .wrMode = EBI_WRITE_MODE_NCS, // Use CS for control
    .rdMode = EBI_READ_MODE_NCS
};

iprintf("Simple I/O device configured\r\n");
```

Example 4: Extended timing range using formula (slow device)

```
// Device requires very long access time: 1 microsecond
// Assuming 100 MHz EBI clock (10 ns per cycle)
// Need 100 cycles for 1us pulse

EBI_CS_cfg_t slowConfig = {
    // Using extended range formula for pulse:
    // nrd_pulse[6] = 0, nrd_pulse[5:0] = 100 won't work (max 63)
    // Need to use bit 6: pulse = 256*0 + 100 = 100 cycles
    // But 100 > 63, so we need bit 6 set
    // Actually: 256*0 + 100 still exceeds 6 bits
    // Correct approach: For 100 cycles, we can't fit in 6 bits alone
    // Let's use a realistic long pulse: 200 cycles = 2us

    .ncs_rd_setup = 10,        // 100ns
    .nrd_setup = 10,           // 100ns
    .ncs_rd_pulse = 200,        // 2us (bit 6 will be set automatically)
    .nrd_pulse = 200,          // 2us
    .nrd_cycles = 220,          // 2.2us total

    .ncs_wr_setup = 10,
    .nwe_setup = 10,
    .ncs_wr_pulse = 200,
    .nwe_pulse = 200,
    .nwe_cycles = 220,

    .tdf_cycles = 5,           // Longer data float time
    .busWidth = EBI_BUS_WIDTH_16,
    .byteAccess = EBI_BYTE_ACCESS_WRITE,
    .nWait = EBI_NWAIT_READY,
    .wrMode = EBI_WRITE_MODE_NWE,
    .rdMode = EBI_READ_MODE_NRD
};

iprintf("Very slow device configured: 2us access time\r\n");
```

Example 5: Helper function to calculate timing from nanoseconds

```
void CalculateEBITiming(uint32_t clockFreqMHz, uint32_t setupNs,
                        uint32_t pulseNs, uint32_t holdNs,
                        EBI_CS_cfg_t *config)
{
    // Calculate cycles needed (round up)
    float nsPerCycle = 1000.0f / clockFreqMHz;

    uint8_t setupCycles = (uint8_t)((setupNs + nsPerCycle - 1) / nsPerCycle);
    uint8_t pulseCycles = (uint8_t)((pulseNs + nsPerCycle - 1) / nsPerCycle);
    uint8_t holdCycles = (uint8_t)((holdNs + nsPerCycle - 1) / nsPerCycle);

    uint16_t totalCycles = setupCycles + pulseCycles + holdCycles;

    // Apply to both read and write (can be customized)
    config->ncs_rd_setup = setupCycles;
    config->nrd_setup = setupCycles;
```

```

    config->ncs_rd_pulse = pulseCycles;
    config->nrd_pulse = pulseCycles;
    config->nrd_cycles = totalCycles;

    config->ncs_wr_setup = setupCycles;
    config->nwe_setup = setupCycles;
    config->ncs_wr_pulse = pulseCycles;
    config->nwe_pulse = pulseCycles;
    config->nwe_cycles = totalCycles;

    iprintf("Calculated timing: %u setup, %u pulse, %u hold cycles\r\n",
            setupCycles, pulseCycles, holdCycles);
}

void UserMain(void *pd) {
    init();

    EBI_CS_cfg_t myConfig = {0};
    myConfig.busWidth = EBI_BUS_WIDTH_16;
    myConfig.wrMode = EBI_WRITE_MODE_NWE;
    myConfig.rdMode = EBI_READ_MODE_NRD;
    myConfig.nWait = EBI_NWAIT_DISABLED;
    myConfig.tdf_cycles = 2;

    // Calculate timing for 150 MHz clock, 20ns setup, 70ns pulse, 10ns hold
    CalculateEBI_Timing(150, 20, 70, 10, &myConfig);

    while(1) {
        OSTimeDly(TICKS_PER_SECOND);
    }
}

```

Example 6: Validating timing configuration

```

bool ValidateEBI_Timing(const EBI_CS_cfg_t *config)
{
    // Check TDF cycles range
    if (config->tdf_cycles > 15) {
        iprintf("Error: tdf_cycles must be 0-15\r\n");
        return false;
    }

    // Verify read timing makes sense
    uint16_t rdSetup = config->ncs_rd_setup;
    uint16_t rdPulse = config->ncs_rd_pulse;
    if ((rdSetup + rdPulse) > config->nrd_cycles) {
        iprintf("Error: Read setup+pulse exceeds total cycles\r\n");
        return false;
    }

    // Verify write timing makes sense
    uint16_t wrSetup = config->ncs_wr_setup;
    uint16_t wrPulse = config->ncs_wr_pulse;
    if ((wrSetup + wrPulse) > config->nwe_cycles) {
        iprintf("Error: Write setup+pulse exceeds total cycles\r\n");
        return false;
    }

    iprintf("EBI timing configuration valid\r\n");
    return true;
}

void ConfigureEBI_WithValidation() {
    EBI_CS_cfg_t config = {
        .ncs_rd_setup = 2, .nrd_setup = 2,
        .ncs_rd_pulse = 10, .nrd_pulse = 10,
        .nrd_cycles = 15,
        .ncs_wr_setup = 2, .nwe_setup = 2,
        .ncs_wr_pulse = 10, .nwe_pulse = 10,
        .nwe_cycles = 15,
        .tdf_cycles = 2,
        .busWidth = EBI_BUS_WIDTH_16,
        .wrMode = EBI_WRITE_MODE_NWE,
        .rdMode = EBI_READ_MODE_NRD,
        .nWait = EBI_NWAIT_DISABLED
    };

    if (ValidateEBI_Timing(&config)) {
        // Apply configuration to hardware
        iprintf("Configuration validated and applied\r\n");
    }
}

```

The documentation for this struct was generated from the following file:

- ebi.h

16.86 Effs_Cfg Struct Reference

Filesystem Configuration.

```
#include <effs_pt.h>
```

Public Attributes

- union {
 - struct {
 - uint16_t **maxblock**
Maximum number of block can be used.
 - uint8_t **sSizeExp**
Sector size wanted to use (less than block size.
 - uint8_t **blSizeExp**
Block size in bytes.
 - uint8_t **dSizeExp**
Max size of fat+directory+block index.
 - uint8_t **cacheSizeExp**
Write cache size.
 - uint8_t **separatedir**
If directory used separatelly from FAT (NAND)
 - uint8_t **csum**
Hamming code csum of struct.
 - uint8_t **pAsBytePtr** [1]
- ```
};
```

### 16.86.1 Detailed Description

Filesystem Configuration.

The documentation for this struct was generated from the following file:

- `effs_pt.h`

## 16.87 fifo\_buffer\_storage Class Reference

FIFO buffer storage using linked pool buffers.

```
#include <buffers.h>
```

### Classes

- class [PeekIterator](#)  
*Iterator for non-destructive examination of buffered data.*
- class [ReadBufPtr](#)  
*RAII accessor for reading from buffer storage.*
- class [WriteBufPtr](#)  
*RAII accessor for writing to buffer storage.*

### Public Member Functions

- uint16\_t [SpaceAvail](#) ()  
*Get number of bytes available for writing.*
- uint16\_t [SpaceUsed](#) ()

- Get number of bytes currently stored.*
- int [ReadData](#) (puint8\_t pCopyTo, int max\_bytes)  
*Read data from the buffer.*
- uint8\_t [PeekData](#) (int idx=-1)  
*Peek at data without removing it.*
- int [WriteData](#) (puint8\_t pCopyFrom, int num\_bytes)  
*Write data to the buffer.*
- [PeekIterator Peek\\_Begin](#) ()  
*Get iterator for beginning of buffered data.*
- [PeekIterator Peek\\_End](#) ()  
*Get iterator for end of buffered data.*
- BOOL [Empty](#) ()  
*Check if buffer is empty.*
- BOOL [Full](#) ()  
*Check if buffer is full.*
- [fifo\\_buffer\\_storage](#) (uint8\_t max\_buffers=0, uint8\_t use\_fromisr=1, uint8\_t use\_fast\_buffs=1)  
*Constructor.*
- [~fifo\\_buffer\\_storage](#) ()  
*Destructor - frees all allocated buffers.*
- void [Reset](#) (uint8\_t max\_buffers)  
*Reset buffer storage and free all data.*

### 16.87.1 Detailed Description

FIFO buffer storage using linked pool buffers.

Manages a FIFO queue of data using a chain of [pool\\_buffer](#) structures. Provides stream-like read/write operations with automatic buffer allocation and deallocation. Used for serial ports and network I/O buffering.

### 16.87.2 Constructor & Destructor Documentation

#### 16.87.2.1 fifo\_buffer\_storage()

```
fifo_buffer_storage::fifo_buffer_storage (
 uint8_t max_buffers = 0,
 uint8_t use_fromisr = 1,
 uint8_t use_fast_buffs = 1)
```

Constructor.

Parameters

|                       |                                  |
|-----------------------|----------------------------------|
| <i>max_buffers</i>    | Maximum number of buffers to use |
| <i>use_fromisr</i>    | Enable ISR-safe operation        |
| <i>use_fast_buffs</i> | Prefer fast buffer pool          |

### 16.87.3 Member Function Documentation

#### 16.87.3.1 Empty()

```
BOOL fifo_buffer_storage::Empty ()
```

Check if buffer is empty.

Returns

TRUE if no data is stored

### 16.87.3.2 Full()

```
BOOL fifo_buffer_storage::Full ()
```

Check if buffer is full.

Returns

TRUE if no more data can be stored

### 16.87.3.3 Peek\_Begin()

```
PeekIterator fifo_buffer_storage::Peek_Begin ()
```

Get iterator for beginning of buffered data.

Returns

Iterator positioned at start of data

### 16.87.3.4 Peek\_End()

```
PeekIterator fifo_buffer_storage::Peek_End ()
```

Get iterator for end of buffered data.

Returns

Iterator positioned after last byte

### 16.87.3.5 PeekData()

```
uint8_t fifo_buffer_storage::PeekData (
 int idx = -1)
```

Peek at data without removing it.

Parameters

|            |                                          |
|------------|------------------------------------------|
| <i>idx</i> | Byte index to peek at (-1 for next byte) |
|------------|------------------------------------------|

Returns

Byte value at specified position

### 16.87.3.6 ReadData()

```
int fifo_buffer_storage::ReadData (
 uint8_t pCopyTo,
 int max_bytes)
```

Read data from the buffer.

Parameters

|                  |                       |
|------------------|-----------------------|
| <i>pCopyTo</i>   | Destination buffer    |
| <i>max_bytes</i> | Maximum bytes to read |

Returns

Number of bytes actually read

### 16.87.3.7 Reset()

```
void fifo_buffer_storage::Reset (
 uint8_t max_buffers)
```

Reset buffer storage and free all data.



## Parameters

|                    |                          |
|--------------------|--------------------------|
| <i>max_buffers</i> | New maximum buffer count |
|--------------------|--------------------------|

**16.87.3.8 SpaceAvail()**

```
uint16_t fifo_buffer_storage::SpaceAvail ()
```

Get number of bytes available for writing.

## Returns

Available space in bytes

**16.87.3.9 SpaceUsed()**

```
uint16_t fifo_buffer_storage::SpaceUsed ()
```

Get number of bytes currently stored.

## Returns

Used space in bytes

**16.87.3.10 WriteData()**

```
int fifo_buffer_storage::WriteData (
 uint8_t *pCopyFrom,
 int num_bytes)
```

Write data to the buffer.

## Parameters

|                  |                          |
|------------------|--------------------------|
| <i>pCopyFrom</i> | Source data buffer       |
| <i>num_bytes</i> | Number of bytes to write |

## Returns

Number of bytes actually written

The documentation for this class was generated from the following file:

- buffers.h

**16.88 HtmlPageHandler Class Reference**

Base class for HTTP request handlers that process requests for specific URLs and HTTP methods.

```
#include <http.h>
```

Inherited by [AcmeAuthItem](#), [CallBackFunctionPageHandler](#), [CallBackWSEndPoint](#), [HtmlConfigExposer](#), and [HtmlPostHandler](#).

**Public Member Functions**

- [HtmlPageHandler](#) (const char \*url, [HTTP\\_RequestTypes](#) rt=[tGet](#), int accessGroup=0, bool Before\_[↩](#)Files=false)  
*Constructs and registers an HTTP request handler for a specific URL pattern and method.*
- [~HtmlPageHandler](#) ()  
*Destructor that automatically unregisters the handler from the global chain.*
- void [MoveHandlerToLast](#) ()

- *Moves this handler to the end of the handler chain, making it function as a 404 handler.*
- virtual int **ProcessRaw** (int sock, [HTTP\\_Request](#) &pd)=0  
*Pure virtual method that must be implemented to process HTTP requests.*
- int **GetGroup** ()  
*Returns the access control group identifier for this handler.*

### Static Public Member Functions

- static [HtmlPageHandler](#) \* **FindHandler** ([HTTP\\_Request](#) &req, bool bBeforeFiles)  
*Locates the appropriate handler for an incoming HTTP request.*

### Protected Member Functions

- void **InsertSort** ([HtmlPageHandler](#) \*&ph)  
*Inserts handler into sorted linked list based on priority.*
- int **SortValue** ([HtmlPageHandler](#) \*pv)  
*Compares handler priorities for sorting (-1, 0, 1)*
- void **Remove** ()  
*Removes handler from the linked list.*
- virtual bool **Match** ([HTTP\\_Request](#) &req)  
*Determines if this handler matches the incoming request.*

### Protected Attributes

- [HtmlPageHandler](#) \* **m\_pNextHandler**  
*Pointer to next handler in the linked list for efficient traversal.*
- const char \* **m\_pUrlName**  
*URL pattern to match. Empty string or NULL matches all requests.*
- int **m\_access\_group**  
*Access control group identifier for authorization checks.*
- [HTTP\\_RequestTypes](#) **m\_requestTypes**  
*HTTP method types this handler responds to (GET, POST, etc.)*

## 16.88.1 Detailed Description

Base class for HTTP request handlers that process requests for specific URLs and HTTP methods.

This abstract base class provides the foundation for handling HTTP requests in NetBurner web applications. Derived classes implement specific request processing logic for particular URLs and HTTP methods (GET, POST, etc.). The class manages handler registration, URL matching, access control, and provides a linked-list structure for efficient handler lookup during request processing. Handlers can be configured to process requests before or after built-in file serving, allowing for custom processing or overriding of static content.

The handler system supports:

- URL pattern matching with wildcard support (NULL/empty string matches all URLs)
- HTTP method filtering (GET, POST, PUT, DELETE, etc.)
- Access group-based authorization
- Priority ordering for handler evaluation
- Integration with NetBurner's built-in file serving system

### Note

Despite the class name "HtmlPageHandler", this class handles all HTTP request types, not just HTML pages. The name is maintained for backward compatibility with existing NetBurner applications. Handlers are automatically registered upon construction and unregistered upon destruction.

## 16.88.2 Constructor & Destructor Documentation

### 16.88.2.1 HtmlPageHandler()

```
HtmlPageHandler::HtmlPageHandler (
 const char * url,
 HTTP_RequestTypes rt = tGet,
 int accessGroup = 0,
 bool Before_Files = false)
```

Constructs and registers an HTTP request handler for a specific URL pattern and method.

Creates a new request handler and automatically registers it in the global handler chain. The handler will be invoked when incoming requests match the specified URL pattern and HTTP method type, subject to access control restrictions.

#### Parameters

|                     |                                                                                                                                                                            |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>url</i>          | URL pattern to handle. NULL or empty string creates a catch-all handler that matches any URL not handled by more specific handlers                                         |
| <i>rt</i>           | HTTP request type(s) to respond to from HTTP_RequestTypes enumeration (default: tGet for GET requests)                                                                     |
| <i>accessGroup</i>  | Access control group identifier for authorization (default: 0 for unrestricted access)                                                                                     |
| <i>Before_Files</i> | Handler priority relative to built-in file serving true: process before file serving (allows overriding static content) false: process after file serving (default: false) |

## 16.88.3 Member Function Documentation

### 16.88.3.1 FindHandler()

```
static HtmlPageHandler * HtmlPageHandler::FindHandler (
 HTTP_Request & req,
 bool bBeforeFiles) [static]
```

Locates the appropriate handler for an incoming HTTP request.

Searches the handler chain to find the first handler that matches the request's URL, HTTP method, and access requirements. Handlers are evaluated in priority order.

#### Parameters

|                     |                                                                                                                                        |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>req</i>          | HTTP request object containing URL, method, and other request details                                                                  |
| <i>bBeforeFiles</i> | Handler priority group to search true: search handlers that run before file serving false: search handlers that run after file serving |

#### Returns

Pointer to matching handler if found, or NULL if no handler matches the request

### 16.88.3.2 GetGroup()

```
int HtmlPageHandler::GetGroup () [inline]
```

Returns the access control group identifier for this handler.

#### Returns

Access group identifier used for authorization checks, [HTTP\\_ACCESS](#)

#### See also

[CheckHttpAccess](#) for access control implementation details

### 16.88.3.3 MoveHandlerToLast()

```
void HtmlPageHandler::MoveHandlerToLast ()
```

Moves this handler to the end of the handler chain, making it function as a 404 handler.

Repositions the handler to be evaluated last, effectively creating a catch-all handler that processes requests not matched by any other handlers. This is commonly used to implement custom 404 error pages or fallback request processing.

### 16.88.3.4 ProcessRaw()

```
virtual int HtmlPageHandler::ProcessRaw (
 int sock,
 HTTP_Request & pd) [pure virtual]
```

Pure virtual method that must be implemented to process HTTP requests.

This method is called when a matching request is received and must contain the actual request processing logic. Implementations should read request data, perform necessary processing, and send appropriate HTTP responses.

#### Parameters

|             |                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------|
| <i>sock</i> | Socket file descriptor for the client connection                                              |
| <i>pd</i>   | HTTP request object containing parsed request details (headers, URL, method, body data, etc.) |

#### Returns

Non-zero if the request was successfully processed and a response was sent, or 0 if the request was not processed (allows fallback to other handlers)

Implemented in [CallBackFunctionPageHandler](#), and [CallBackFunctionPostHandler](#).

The documentation for this class was generated from the following file:

- [http.h](#)

## 16.89 HtmlPostVariableListCallback Class Reference

Implements the HtmlPostVariableListHandler class as a function pointer callback for HTTP POST submissions.

```
#include <httppost.h>
```

Inherits [HtmlPostVariableListHandler](#).

### Public Member Functions

- [HtmlPostVariableListCallback](#) (const char \*pUrl, postvarhandler \*pCallback, int accessGroup=0)  
*Custom HTTP POST handler callback function constructor.*
- [HtmlPostVariableListCallback](#) (const char \*pUrl, postvarhandler\_int \*pCallback, int accessGroup=0)  
*Custom HTTP POST handler callback function constructor.*

### 16.89.1 Detailed Description

Implements the HtmlPostVariableListHandler class as a function pointer callback for HTTP POST submissions.

### 16.89.2 Constructor & Destructor Documentation

#### 16.89.2.1 HtmlPostVariableListCallback() [1/2]

```
HtmlPostVariableListCallback::HtmlPostVariableListCallback (
 const char * pUrl,
 postvarhandler * pCallback,
 int accessGroup = 0) [inline]
```

Custom HTTP POST handler callback function constructor.

## Parameters

|                    |                                                         |
|--------------------|---------------------------------------------------------|
| <i>pUrl</i>        | Pointer to URL character string                         |
| <i>pCallback</i>   | Pointer to callback function that will process the POST |
| <i>accessGroup</i> | Password group access                                   |

**16.89.2.2 HtmlPostVariableListCallback() [2/2]**

```
HtmlPostVariableListCallback::HtmlPostVariableListCallback (
 const char * pUrl,
 postvarhandler_int * pCallback,
 int accessGroup = 0) [inline]
```

Custom HTTP POST handler callback function constructor.

## Parameters

|                    |                                                         |
|--------------------|---------------------------------------------------------|
| <i>pUrl</i>        | Pointer to URL character string                         |
| <i>pCallback</i>   | Pointer to callback function that will process the POST |
| <i>accessGroup</i> | Password group access                                   |

The documentation for this class was generated from the following file:

- httppost.h

**16.90 HTTP\_Request Struct Reference**

HTTP Request Structure.

```
#include <http.h>
```

**Public Member Functions**

- void **ProcessLine** (PSTR startofline)  
*internal function*
- int **Respond** (int socket)  
*internal function*
- bool **ExtractAuthentication** (char \*\*pPassword, char \*\*pUser)  
*used to extract user name and password in authentication test*
- const char \* [GetBoundaryMarker](#) ()  
*Extracts the boundary marker string from multipart Content-Type headers.*
- int [ReadSimpleBody](#) (int socket, const char \*boundary=NULL)  
*Reads the HTTP request body from a socket into the internal buffer and processes termination conditions.*

**Public Attributes**

- PSTR **pURL**  
*Request URL.*
- PSTR **pAuthorization**  
*Authorization header if present, otherwise null.*
- PSTR **pFirstCookie**  
*First cookie if present, otherwise null. More may follow.*
- PSTR **pData**  
*Pointer to entire data set. Note: not null terminated.*
- PSTR **pSep**

- Separator for multipart forms, null if not present.*
- **PSTR pHost**  
*Pointer to host record null if not present.*
- **PSTR last\_datarx**  
*Pointer to last data read, internal use only.*
- **PSTR wsKey**  
*Web socket key, internal use only.*
- **PSTR wsProtocol**  
*Web socket protocol, internal use only.*
- **uint16\_t sep\_len**  
*Length of separator for multipart forms.*
- **uint32\_t content\_length**  
*Content length field from HTML header.*
- **uint32\_t rx\_length**  
*Total bytes receive from request, internal use only.*
- **IPADDR client\_IPAddr**  
*IP address of client.*
- **uint8\_t websocketFlags**  
*Web socket flags.*
- **HTTP\_RequestTypes req**  
*Type of request [HTTP\\_RequestTypes](#).*

### 16.90.1 Detailed Description

HTTP Request Structure.

This structure is populated before a HTTP request is sent out.

### 16.90.2 Member Function Documentation

#### 16.90.2.1 GetBoundaryMarker()

```
const char * HTTP_Request::GetBoundaryMarker ()
```

Extracts the boundary marker string from multipart Content-Type headers.

This function searches through the previously parsed HTTP headers to locate and extract the boundary parameter from the Content-Type header when multipart content is present. The boundary marker is used to delimit individual parts within multipart HTTP content such as multipart/form-data or multipart/mixed. The function parses the Content-Type header according to RFC specifications to isolate the boundary value from other Content-Type parameters.

#### Returns

Pointer to null-terminated string containing the boundary marker value if found, or NULL if no multipart Content-Type header exists or no boundary parameter is specified in the Content-Type header

#### Note

The returned pointer references internal header data and remains valid as long as the header information is preserved. This function should only be called after HTTP headers have been successfully parsed. The boundary marker does not include the leading "--" prefix used in actual multipart delimiters. Multipart content requires a boundary parameter to be properly processed according to RFC standards.

#### See also

[ReadSimpleBody\(\)](#) for using the boundary marker during body parsing

RFC 2046 for multipart media type specifications

### 16.90.2.2 ReadSimpleBody()

```
int HTTP_Request::ReadSimpleBody (
 int socket,
 const char * boundary = NULL)
```

Reads the HTTP request body from a socket into the internal buffer and processes termination conditions. This function reads the remaining HTTP request body data from an established socket connection into the buffer backing the request object, preparing the `pData` member for access. The function handles different HTTP body termination scenarios including content-length bounded bodies, chunked transfer encoding, and boundary-delimited content such as multipart data. Reading continues until one of several termination conditions is met, allowing proper handling of various HTTP body formats.

#### Parameters

|                 |                                                                                                                                                                                                                                                                   |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>socket</i>   | File descriptor of the established HTTP connection Must be a valid socket with available data to read                                                                                                                                                             |
| <i>boundary</i> | Pointer to null-terminated string containing the boundary sequence that indicates the end of the request body (default: NULL) Used for multipart content or custom termination sequences If NULL, termination is determined by Content-Length or chunked encoding |

#### Returns

HTTP\_RRR\_END\_FOUND if the function successfully found the trailing termination sequence and completed reading the body

HTTP\_RRR\_FINISHED if all content was read as indicated by HTTP headers (Content-Length reached or chunked encoding completed)

HTTP\_RRR\_BUFF\_FULL if the internal buffer became full before the body was completely read, indicating more data remains available on the socket

HTTP\_RRR SOCK\_ERR if a socket error occurred during reading, typically indicating the connection was closed or experienced a network error

#### Note

This function modifies the internal buffer state and primes the `pData` member for access. The socket must have HTTP headers already processed before calling this function. For multipart or boundary-terminated content, the boundary parameter must be specified. Buffer overflow conditions require additional reads to retrieve remaining data. The function respects HTTP protocol specifications for body termination.

#### See also

[HTTP\\_RRR\\_END\\_FOUND](#), [HTTP\\_RRR\\_FINISHED](#), [HTTP\\_RRR\\_BUFF\\_FULL](#), [HTTP\\_RRR SOCK\\_ERR](#) for return code definitions

The documentation for this struct was generated from the following file:

- `http.h`

## 16.91 I2C Class Reference

I2C Peripheral Class.

```
#include <i2c.h>
```

#### Public Types

- enum [Result\\_t](#) {  
[I2C\\_RES\\_ACK](#) , [I2C\\_RES\\_NACK](#) , [I2C\\_RES\\_ARB\\_LST](#) , [I2C\\_RES\\_BUSY](#) ,  
[I2C\\_RES\\_ARG](#) }  
*Return result types.*

## Public Member Functions

- [I2C](#) (int module)  
*Constructor for the [I2C](#) peripheral module.*
- void [setup](#) (uint32\_t busSpeed)  
*Setup the [I2C](#) peripheral interface.*
- void [resetBus](#) ()  
*Reset the [I2C](#) bus.*
- [Result\\_t DoTransaction](#) (I2CTxn\_t \*pTransaction, bool bRepeatedStart=false)  
*Start an [I2C](#) transaction.*
- void [setNumAddressBytes](#) (uint8\_t numAddressBytes=1)  
*Specify the register address size for a read or write transaction. A number of bytes, from 0 to 3, can be sent to specify an address up to 24-bit (3 bytes): 0 = none, 1 = 8-bits, 2 = 16-bits, 3 = 24-bits.*
- [Result\\_t writeReg8](#) (uint8\_t devAddr, uint32\_t reg, uint8\_t data)  
*Write an 8-bit value to a [I2C](#) slave device register.*
- [Result\\_t readReg8](#) (uint8\_t devAddr, uint32\_t reg, uint8\_t &data)  
*Read an 8-bit value form an [I2C](#) slave device register.*
- [Result\\_t writeRegN](#) (uint8\_t devAddr, uint32\_t reg, const uint8\_t \*buf, uint32\_t blen)  
*Write a number of 8-bit values to an [I2C](#) slave to the specified register address.*
- [Result\\_t readRegN](#) (uint8\_t devAddr, uint32\_t reg, uint8\_t \*buf, uint32\_t blen)  
*Read a number of 8-bit values from an [I2C](#) slave at the specified register address.*

## Friends

- class [WireIntf](#)
- class [TwoWire](#)

## 16.91.1 Detailed Description

[I2C](#) Peripheral Class.

For [I2C](#) communication, you can choose the [WireIntf](#) class, or the [I2C](#) class. The [WireIntf](#) class is simpler to use, while the [I2C](#) class provides more low level control.

Note that the system automatically instantiates [I2C](#) and [WireIntf](#) objects for each of the [I2C](#) peripheral modules. The [WireIntf](#) objects can be accessed with: [Wire](#), [Wire1](#) and [Wire2](#). The [I2C](#) objects can be accessed with [I2C\[0\]](#), [I2C\[1\]](#) and [I2C\[2\]](#).

## 16.91.2 Member Enumeration Documentation

### 16.91.2.1 Result\_t

enum [I2C::Result\\_t](#)

Return result types.

#### Enumerator

|                                 |                        |
|---------------------------------|------------------------|
| <a href="#">I2C_RES_ACK</a>     | Acknowledged.          |
| <a href="#">I2C_RES_NACK</a>    | Not acknowledged.      |
| <a href="#">I2C_RES_ARB_LST</a> | Arbitration listening. |
| <a href="#">I2C_RES_BUSY</a>    | Bus is busy.           |
| <a href="#">I2C_RES_ARG</a>     | Bad argument.          |



## 16.91.3 Constructor & Destructor Documentation

### 16.91.3.1 I2C()

```
I2C::I2C (
 int module)
```

Constructor for the [I2C](#) peripheral module.

#### Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>module</i> | The <a href="#">I2C</a> module to instantiate (0, 1 or 2) |
|---------------|-----------------------------------------------------------|

## 16.91.4 Member Function Documentation

### 16.91.4.1 DoTransaction()

```
Result_t I2C::DoTransaction (
 I2CTxn_t * pTransaction,
 bool bRepeatedStart = false)
```

Start an [I2C](#) transaction.

Execute the transaction pointed to by pTransaction

#### Parameters

|                       |                                               |
|-----------------------|-----------------------------------------------|
| <i>*pTransaction</i>  | Pointer to the transaction to execute         |
| <i>bRepeatedStart</i> | Repeat the <a href="#">I2C</a> start sequence |

#### Returns

[Result\\_t](#)

### 16.91.4.2 readReg8()

```
Result_t I2C::readReg8 (
 uint8_t devAddr,
 uint32_t reg,
 uint8_t & data)
```

Read an 8-bit value form an [I2C](#) slave device register.

#### Parameters

|                |                                                           |
|----------------|-----------------------------------------------------------|
| <i>devAddr</i> | Address of <a href="#">I2C</a> device                     |
| <i>reg</i>     | Register address to read                                  |
| <i>data</i>    | Reference to variable in which to store the register data |

#### Returns

[Result\\_t](#)

### 16.91.4.3 readRegN()

```
Result_t I2C::readRegN (
 uint8_t devAddr,
 uint32_t reg,
 uint8_t * buf,
 uint32_t blen)
```

Read a number of 8-bit values from an [I2C](#) slave at the specified register address.

## Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>devAddr</i> | Address of <a href="#">I2C</a> device    |
| <i>reg</i>     | Register address to read                 |
| <i>*buf</i>    | Pointer to buffer to store received data |
| <i>blen</i>    | Number of bytes to read                  |

## Returns

[Result\\_t](#)**16.91.4.4 setNumAddressBytes()**

```
void I2C::setNumAddressBytes (
 uint8_t numAddressBytes = 1) [inline]
```

Specify the register address size for a read or write transaction. A number of bytes, from 0 to 3, can be sent to specify an address up to 24-bit (3 bytes): 0 = none, 1 = 8-bits, 2 = 16-bits, 3 = 24-bits.

If you are using a single [I2C](#) device, or all devices use the same address size, this function only needs to be called once. If you have multiple [I2C](#) devices with different address/register bit sizes, then this function should be called before each read/write operation for each different device.

## Parameters

|                        |                                                                          |
|------------------------|--------------------------------------------------------------------------|
| <i>numAddressBytes</i> | The number of address bytes to send: 0 - 3. The default value is 1 byte. |
|------------------------|--------------------------------------------------------------------------|

**16.91.4.5 setup()**

```
void I2C::setup (
 uint32_t busSpeed)
```

Setup the [I2C](#) peripheral interface.

## Parameters

|                 |                                                                                                |
|-----------------|------------------------------------------------------------------------------------------------|
| <i>busSpeed</i> | Target bus speed. If the specified speed cannot be achieved, a lower bus speed may be enabled. |
|-----------------|------------------------------------------------------------------------------------------------|

**16.91.4.6 writeReg8()**

```
Result_t I2C::writeReg8 (
 uint8_t devAddr,
 uint32_t reg,
 uint8_t data)
```

Write an 8-bit value to a [I2C](#) slave device register.

## Parameters

|                |                                       |
|----------------|---------------------------------------|
| <i>devAddr</i> | Address of <a href="#">I2C</a> device |
| <i>reg</i>     | Register address to write             |
| <i>data</i>    | Data to write to register             |

## Returns

[Result\\_t](#)

### 16.91.4.7 writeRegN()

```
Result_t I2C::writeRegN (
 uint8_t devAddr,
 uint32_t reg,
 const uint8_t * buf,
 uint32_t blen)
```

Write a number of 8-bit values to an [I2C](#) slave to the specified register address.

Executing this function will send the following: a start bit, the slave device address and read/write bit, the 8-bit data bytes, and finally a stop bit. Note that the address is not incremented with each 8-bit data value.

#### Parameters

|                |                                            |
|----------------|--------------------------------------------|
| <i>devAddr</i> | Address of <a href="#">I2C</a> device      |
| <i>reg</i>     | Register address to write                  |
| <i>*buf</i>    | Pointer to buffer containing data to write |
| <i>blen</i>    | Number of bytes to write                   |

#### Returns

[Result\\_t](#)

The documentation for this class was generated from the following file:

- [i2c.h](#)

## 16.92 I2CDevice Class Reference

[I2C](#) Device Class (recommended)

```
#include <i2c.h>
```

### Public Member Functions

- [I2CDevice](#) ([I2C](#) &Interface, uint8\_t deviceAddress, uint8\_t numAddressBytes=1)  
*Initialize the [I2C](#) module.*
- uint8\_t [getI2CAddress](#) ()  
*Get device's [I2C](#) address.*
- void [setup](#) (uint32\_t busSpeed)  
*Setup the [I2C](#) peripheral module.*
- void [resetBus](#) ()  
*Reset the [I2C](#) bus.*
- [I2C::Result\\_t writeReg8](#) (uint32\_t reg, uint8\_t data)  
*Write an 8-bit value to an [I2C](#) slave device register.*
- [I2C::Result\\_t readReg8](#) (uint32\_t reg, uint8\_t &data)  
*Read an 8-bit value form an [I2C](#) slave device register.*
- [I2C::Result\\_t writeRegN](#) (uint32\_t reg, const uint8\_t \*buf, uint32\_t blen)  
*Write a number of 8-bit values to an [I2C](#) slave to the specified register address.*
- [I2C::Result\\_t readRegN](#) (uint32\_t reg, uint8\_t \*buf, uint32\_t blen)  
*Read a number of 8-bit values from an [I2C](#) slave at the specified register address.*

### 16.92.1 Detailed Description

[I2C](#) Device Class (recommended)

For [I2C](#) communication, you can choose the [I2CDevice](#) class, the [I2C](#) class, or the [WireIntf](#) class. The recommended interface is the [I2CDevice](#) class.

For example, to create an object for an analog to digital converter:

```
I2CDevice AdcDevice(i2c[I2C_MODULE_NUM], I2C_ADC_ADDRESS);
```

To read a 16 bit value:

```
uint8_t I2CStat = AdcDevice.readRegN(0x00, buf, 2);
```

## 16.92.2 Constructor & Destructor Documentation

### 16.92.2.1 I2CDevice()

```
I2CDevice::I2CDevice (
 I2C & pInterface,
 uint8_t deviceAddress,
 uint8_t numAddressBytes = 1) [inline]
```

Initialize the I2C module.

#### Parameters

|                        |                                                                                                                                                                                                                                                                                                              |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>pInterface</i>      | Pointer to the I2C interface to use. This will normally be one of the preallocated I2C modules in the i2c[] array: i2c[0] (default), i2c[1] or i2c[2].                                                                                                                                                       |
| <i>deviceAddress</i>   | I2C device address                                                                                                                                                                                                                                                                                           |
| <i>numAddressBytes</i> | Number of address bytes to send to specify the register or address to access for a read or write operation. Refer to your I2C device data sheet to determine the correct number. The I2C address range is 24-bits maximum (3 bytes). The default value is 1 byte, representing an address range of 0 to 255. |

## 16.92.3 Member Function Documentation

### 16.92.3.1 getI2CAddress()

```
uint8_t I2CDevice::getI2CAddress () [inline]
```

Get device's I2C address.

Note that the return address will be just the I2C address bits, not including the read/write bit. If your device data sheet specifies the address byte to include the read/write bit, that value would be this address left shifted by 1. For example, if a device data sheet specifies the address as 0xA2 and includes the read/write bit, the actual address would be 0x51.

#### Returns

The device's I2C address

### 16.92.3.2 readReg8()

```
I2C::Result_t I2CDevice::readReg8 (
 uint32_t reg,
 uint8_t & data) [inline]
```

Read an 8-bit value form an I2C slave device register.

#### Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>reg</i>  | Register address to read                                  |
| <i>data</i> | Reference to variable in which to store the register data |

#### Returns

I2C::Result\_t

### 16.92.3.3 readRegN()

```
I2C::Result_t I2CDevice::readRegN (
 uint32_t reg,
 uint8_t * buf,
 uint32_t blen) [inline]
```

Read a number of 8-bit values from an I2C slave at the specified register address.

## Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>reg</i>  | Register address to read                 |
| <i>*buf</i> | Pointer to buffer to store received data |
| <i>blen</i> | Number of bytes to read                  |

## Returns

[I2C::Result\\_t](#)

**16.92.3.4 setup()**

```
void I2CDevice::setup (
 uint32_t busSpeed) [inline]
```

Setup the [I2C](#) peripheral module.

## Parameters

|                 |                                                                                                |
|-----------------|------------------------------------------------------------------------------------------------|
| <i>busSpeed</i> | Target bus speed. If the specified speed cannot be achieved, a lower bus speed may be enabled. |
|-----------------|------------------------------------------------------------------------------------------------|

**16.92.3.5 writeReg8()**

```
I2C::Result_t I2CDevice::writeReg8 (
 uint32_t reg,
 uint8_t data) [inline]
```

Write an 8-bit value to an [I2C](#) slave device register.

## Parameters

|             |                           |
|-------------|---------------------------|
| <i>reg</i>  | Register address to write |
| <i>data</i> | Data to write to register |

## Returns

[I2C::Result\\_t](#)

**16.92.3.6 writeRegN()**

```
I2C::Result_t I2CDevice::writeRegN (
 uint32_t reg,
 const uint8_t * buf,
 uint32_t blen) [inline]
```

Write a number of 8-bit values to an [I2C](#) slave to the specified register address.

Executing this function will send the following: a start bit, the slave device address and read/write bit, the 8-bit data bytes, and finally a stop bit. Note that the address is not incremented with each 8-bit data value.

## Parameters

|             |                                            |
|-------------|--------------------------------------------|
| <i>reg</i>  | Register address to write                  |
| <i>*buf</i> | Pointer to buffer containing data to write |
| <i>blen</i> | Number of bytes to write                   |

## Returns

[I2C::Result\\_t](#)

The documentation for this class was generated from the following file:

- [i2c.h](#)

## 16.93 I4Record Class Reference

#include <config\_netobj.h>

Inherits [config\\_obj](#).

### Public Member Functions

- [I4Record](#) (const char \*name, const char \*desc=NULL)
- void [Attach](#) (void \*pb)
- void [GetReportItem](#) (NBString &s, int item)

### Public Member Functions inherited from [config\\_obj](#)

- [config\\_obj](#) ([config\\_obj](#) &owner, const char \*name, const char \*desc)  
*Object constructor with the parent/owner leaf parameter.*
- [config\\_obj](#) (const char \*name, const char \*desc)  
*Object constructor.*
- virtual void [GetTextValue](#) (NBString &s) override  
*Get the object value as a text string to the specified NBString object.*
- virtual void [GetTypeValue](#) (NBString &s) override  
*Assigns the object type value to the specified NBString object.*

### 16.93.1 Detailed Description

### 16.93.2 Constructor & Destructor Documentation

#### 16.93.2.1 I4Record()

```
I4Record::I4Record (
 const char * name,
 const char * desc = NULL) [inline]
```

### 16.93.3 Member Function Documentation

#### 16.93.3.1 Attach()

```
void I4Record::Attach (
 void * pb) [inline]
```

#### 16.93.3.2 GetReportItem()

```
void I4Record::GetReportItem (
 NBString & s,
 int item)
```

The documentation for this class was generated from the following file:

- config\_netobj.h

## 16.94 I6Record Class Reference

#include <config\_netobj.h>

Inherits [config\\_obj](#).

### Public Member Functions

- [I6Record](#) (const char \*name, const char \*desc=NULL)
- void [Attach](#) (int nif)
- void [ReportGetTextValue](#) (NBString &s, int item)
- int [GetNlf](#) ()

## Public Member Functions inherited from [config\\_obj](#)

- [config\\_obj](#) ([config\\_obj](#) &owner, const char \*name, const char \*desc)  
*Object constructor with the parent/owner leaf parameter.*
- [config\\_obj](#) (const char \*name, const char \*desc)  
*Object constructor.*
- virtual void [GetTextValue](#) ([NBString](#) &s) override  
*Get the object value as a text string to the specified [NBString](#) object.*
- virtual void [GetTypeValue](#) ([NBString](#) &s) override  
*Assigns the object type value to the specified [NBString](#) object.*

### 16.94.1 Detailed Description

### 16.94.2 Constructor & Destructor Documentation

#### 16.94.2.1 I6Record()

```
I6Record::I6Record (
 const char * name,
 const char * desc = NULL) [inline]
```

### 16.94.3 Member Function Documentation

#### 16.94.3.1 Attach()

```
void I6Record::Attach (
 int nif) [inline]
```

#### 16.94.3.2 GetnIf()

```
int I6Record::GetnIf () [inline]
```

#### 16.94.3.3 ReportGetTextValue()

```
void I6Record::ReportGetTextValue (
 NBString & s,
 int item)
```

The documentation for this class was generated from the following file:

- [config\\_netobj.h](#)

## 16.95 IEEE1588Timer Class Reference

```
#include <IEEE1588Timer.h>
```

### Public Member Functions

- bool [init](#) ()
- uint32\_t [getCurrentTime](#) (EthernetVars &ev)
- void [enablePPSCapture](#) ()
- void [markValidTime](#) (uint8\_t index, uint32\_t ntpTime)
- uint32\_t [getConsecutiveValidSeconds](#) () const
- void [updateConsecutiveSeconds](#) (uint32\_t currentTimeTick)
- void [resetConsecutiveSeconds](#) ()
- uint32\_t [getPPSLoop](#) () const
- uint32\_t [getPPSCapture](#) (uint8\_t index) const

## Static Public Member Functions

- static void [PPS\\_1588\\_TimeCaplSr](#) ()
- static uint32\_t [ring\\_diff\\_forward](#) (uint32\_t a, uint32\_t b)

### 16.95.1 Detailed Description

[IEEE1588Timer](#) class - Encapsulates 1588 timer functionality for GPS PPS synchronization

This class manages the hardware 1588 timer on the ethernet controller for precise time synchronization with GPS PPS signals. It provides methods to:

- Initialize the 1588 timer hardware
- Capture PPS edges with precise timestamps
- Track GPS time synchronization validity
- Calculate precise NTP timestamps

### 16.95.2 Member Function Documentation

#### 16.95.2.1 [enablePPSCapture\(\)](#)

```
void IEEE1588Timer::enablePPSCapture ()
```

Enable PPS capture on channel 2 Configures interrupt handling for PPS edge detection

#### 16.95.2.2 [getConsecutiveValidSeconds\(\)](#)

```
uint32_t IEEE1588Timer::getConsecutiveValidSeconds () const [inline]
```

Get count of consecutive valid GPS seconds

Returns

Number of consecutive seconds with valid GPS time

#### 16.95.2.3 [getCurrentTime\(\)](#)

```
uint32_t IEEE1588Timer::getCurrentTime (
 EthernetVars & ev)
```

Get current 1588 timer value

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <a href="#">ev</a> | Reference to ethernet variables structure |
|--------------------|-------------------------------------------|

Returns

Current 1588 timer count

#### 16.95.2.4 [getPPSCapture\(\)](#)

```
uint32_t IEEE1588Timer::getPPSCapture (
 uint8_t index) const [inline]
```

Access PPS capture arrays for timestamp calculation These methods provide read-only access to internal arrays for use by [NtpTimeStamp::SetFrom](#) method

#### 16.95.2.5 [getPPSLoop\(\)](#)

```
uint32_t IEEE1588Timer::getPPSLoop () const [inline]
```

Get current PPS loop index

Returns

Current ring buffer write index



**16.95.2.6 init()**

```
bool IEEE1588Timer::init ()
```

Initialize the 1588 timer hardware Sets up the timer with proper clock source and enables PPS capture

**Returns**

true if initialization successful, false on error

**16.95.2.7 markValidTime()**

```
void IEEE1588Timer::markValidTime (
 uint8_t index,
 uint32_t ntpTime)
```

Mark a GPS time as valid for a specific PPS capture Called by GPS processing when GPRMC message provides valid time

**Parameters**

|                |                                         |
|----------------|-----------------------------------------|
| <i>index</i>   | Ring buffer index for the PPS capture   |
| <i>ntpTime</i> | NTP timestamp corresponding to this PPS |

**16.95.2.8 PPS\_1588\_TimeCapIsr()**

```
void IEEE1588Timer::PPS_1588_TimeCapIsr () [static]
```

Static ISR handler for PPS capture interrupt This must be static to be used as C-style function pointer

**16.95.2.9 resetConsecutiveSeconds()**

```
void IEEE1588Timer::resetConsecutiveSeconds () [inline]
```

Reset consecutive seconds tracking Called when GPS synchronization is lost

**16.95.2.10 ring\_diff\_forward()**

```
static uint32_t IEEE1588Timer::ring_diff_forward (
 uint32_t a,
 uint32_t b) [inline], [static]
```

Utility functions for ring buffer arithmetic These are used for calculating time differences in wrap-around scenarios

**16.95.2.11 updateConsecutiveSeconds()**

```
void IEEE1588Timer::updateConsecutiveSeconds (
 uint32_t currentTimeTick)
```

Update consecutive seconds tracking Called when new valid GPS time is received

**Parameters**

|                        |                       |
|------------------------|-----------------------|
| <i>currentTimeTick</i> | Current OS tick count |
|------------------------|-----------------------|

The documentation for this class was generated from the following files:

- IEEE1588Timer.h
- IEEE1588Timer.cpp

**16.96 InterfaceBlock Class Reference**

Network interface configuration block class for interface control and configuration.

```
#include <netinterface.h>
```

Inherits [config\\_obj](#).

Inherited by EtherLikeInterface, MultiHomeInterface, and PPPInterface.

## Public Member Functions

- [InterfaceBlock](#) (const char \*name, const char \*desc=NULL)  
*Constructor with interface name and description.*
- [InterfaceBlock](#) ([config\\_obj](#) &owner, const char \*name, const char \*desc=NULL)  
*Constructor with [config\\_obj](#) owner, interface name and description.*
- void [RegisterInterface](#) ()  
*Register a network interface with system.*
- virtual void [EnableMulticast](#) ([MACADR](#) macAddress, BOOL addAddress)=0  
*Enable Multicast on this interface.*
- virtual bool [LinkActive](#) ()=0  
*Returns the link status of the network interface.*
- virtual int [LinkSpeed](#) ()=0  
*Returns the link speed of the network interface.*
- virtual bool [LinkDuplex](#) ()=0  
*Returns the full or half link duplex of the network interface.*
- void [InterfaceLinkChange](#) (bool link)  
*Change the link status of the network interface.*
- virtual const char \* [GetInterfaceName](#) ()  
*Returns the interface name network interface.*
- int [GetInterfaceNumber](#) ()  
*Returns the Interface Number of the network interface.*
- bool [IsRootInterface](#) ()  
*Returns true if this is the first interface in the interface list.*

## Public Member Functions inherited from [config\\_obj](#)

- [config\\_obj](#) ([config\\_obj](#) &owner, const char \*name, const char \*desc)  
*Object constructor with the parent/owner leaf parameter.*
- [config\\_obj](#) (const char \*name, const char \*desc)  
*Object constructor.*
- virtual void [GetTextValue](#) (NBString &s) override  
*Get the object value as a text string to the specified [NBString](#) object.*
- virtual void [GetTypeValue](#) (NBString &s) override  
*Assigns the object type value to the specified [NBString](#) object.*

## Public Attributes

- [I4Record](#) **ip4** {"IPv4"}  
*IPv4 configuration for "this" interface, see [config\\_netobj.h](#) for details.*
- [I6Record](#) **ip6** {"IPv6"}  
*IPv6 configuration for "this" interface, see [config\\_netobj.h](#) for details.*
- [config\\_MACADR](#) **MAC** {"00:00:00:00:00:00", "MAC"}  
*Interface MAC address. May be 0 or null for interfaces such as PPP.*
- [config\\_string](#) **device\_name** {"", "DeviceName"}  
*Name of interface. Used to register for DNS and NetBIOS.*
- [config\\_string](#) **net\_domain** {"", "NetworkDomain"}  
*Network Domain.*
- [config\\_int](#) **dhcp\_discover\_secs** {1, "DhcpDiscoverSec"}  
*Seconds to delay after boot before sending a [DHCP](#) Discover message.*
- [config\\_string](#) **discovery\_server** {"discover.netburner.com/DevicePost", "DiscoveryReportUrl"}  
*NetBurner discover server name.*

- `config_int discovery_interval` {(15 \* 60), "DiscoveryReportInterval"}  
*How often to report to the NetBurner discover server.*
- `config_bool obfuscate_discovery` {true, "DiscoveryObfuscate", "Should discovery obfuscate data"}  
*Obfuscate data instead of plain text.*
- `config_bool suppress_default_responses` {false, "SuppressDefault", "Suppress ping and udp Echo ports"}  
*Disable Ping and Echo (port 7)*
- `config_int vlan_tag_value` {0, "VlanTag"}  
*VLAN tag.*
- `int32_t root_if`  
*Root interface number. If multihomed, the interface would be a child of the root interface.*
- `int32_t my_ifnum`  
*This interface number.*
- `uint16_t netMss`  
*Network max segment size for this interface.*
- `uint8_t checksumOffload`  
*Non-zero if hardware interface support offloading checksum calculation.*
- `bool bHaveSeenLink`  
*True if a link has been present previously at least once.*
- `DhcpObject dhcpClient` {this}  
*DHCP client support.*
- `discover_servlet disco_servlet` {this}  
*The servlet that runs the NetBurner discovery server process.*
- `void(* MCastLinkNotify)(InterfaceBlock *pBlock, bool bLink)`  
*A function pointer called when link changes for multicast.*
- `IPADDR4 previous_addr`  
*Previous IPv4 address.*
- `IPADDR4 previous_mask`  
*Previous IPv4 mask.*
- `IPADDR4 previous_gate`  
*Previous IPv4 gateway.*
- `AutoIPClient AutoClient`  
*AutoIP client object for this interface.*

### 16.96.1 Detailed Description

Network interface configuration block class for interface control and configuration.

Network Interfaces are configured and controlled C++ [InterfaceBlock](#) objects. The [InterfaceBlock](#) contains variables and member functions for things such as:

- The Configuration Server interface
- IP settings for both IPv4 and IPv6 types
- Configuration to static or [DHCP](#), and [DHCP](#) status (both v4 and v6)
- Multihome
- Routing
- AutoIP

#### Note

While the recommended method for getting and setting network interface parameters is to obtain a pointer to an [InterfaceBlock](#), there are also non-member functions of the object that can be used for situations that are best served by a single function call, such as just getting a single IP address of an interface. When using those functions pay special attention to the return value type. While `IPADDR` should be used because it supports both `IPADDR4` and `IPADDR6` types automatically, some of these helper functions may specifically return an `IPADDR4` type.

## 16.96.2 Constructor & Destructor Documentation

### 16.96.2.1 InterfaceBlock() [1/2]

```
InterfaceBlock::InterfaceBlock (
 const char * name,
 const char * desc = NULL)
```

Constructor with interface name and description.

#### Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>name</i> | Interface name.                                      |
| <i>desc</i> | Interface description. If not used, default is null. |

### 16.96.2.2 InterfaceBlock() [2/2]

```
InterfaceBlock::InterfaceBlock (
 config_obj & owner,
 const char * name,
 const char * desc = NULL)
```

Constructor with [config\\_obj](#) owner, interface name and description.

#### Parameters

|              |                                                                |
|--------------|----------------------------------------------------------------|
| <i>owner</i> | Reference to <a href="#">config_obj</a> owner of the interface |
| <i>name</i>  | Interface name.                                                |
| <i>desc</i>  | Interface description. If not used, default is null.           |

## 16.96.3 Member Function Documentation

### 16.96.3.1 EnableMulticast()

```
virtual void InterfaceBlock::EnableMulticast (
 MACADR macAddress,
 BOOL addAddress) [pure virtual]
```

Enable Multicast on this interface.

Calls registered interface multicast routine.

#### Parameters

|                   |                                                   |
|-------------------|---------------------------------------------------|
| <i>macAddress</i> | Multicast MAC address.                            |
| <i>addAddress</i> | True to add address/interface, false to remove it |

#### See also

[EnableMulticast\(MACADR macAddress, int interface\)](#), [DisableMulticast\(\)](#)

### 16.96.3.2 GetInterfaceName()

```
virtual const char * InterfaceBlock::GetInterfaceName () [virtual]
```

Returns the interface name network interface.

#### Returns

Pointer to a character string containing the name of the network interface

#### See also

[InterfaceName\(\)](#)

### 16.96.3.3 GetInterfaceNumber()

```
int InterfaceBlock::GetInterfaceNumber () [inline]
```

Returns the Interface Number of the network interface.

#### Returns

Interface number of the [InterfaceBlock](#).

#### See also

[GetInterfaceNumber\(InterfaceBlock \\*pifb\)](#)

### 16.96.3.4 InterfaceLinkChange()

```
void InterfaceBlock::InterfaceLinkChange (
 bool link)
```

Change the link status of the network interface.

#### Parameters

|             |                                         |
|-------------|-----------------------------------------|
| <i>link</i> | Set link active status to true or false |
|-------------|-----------------------------------------|

### 16.96.3.5 IsRootInterface()

```
bool InterfaceBlock::IsRootInterface () [inline]
```

Returns true if this is the first interface in the interface list.

#### Returns

True if this is the first interface in the interface list, otherwise false

### 16.96.3.6 LinkActive()

```
virtual bool InterfaceBlock::LinkActive () [pure virtual]
```

Returns the link status of the network interface.

#### Returns

True if link is active, otherwise false

#### See also

[GetInterfaceLink\(\)](#), [InterfaceLinkActive\(\)](#)

### 16.96.3.7 LinkDuplex()

```
virtual bool InterfaceBlock::LinkDuplex () [pure virtual]
```

Returns the full or half link duplex of the network interface.

#### Returns

True if full duplex, false if half duplex

#### See also

[InterfaceLinkDuplex\(\)](#)

### 16.96.3.8 LinkSpeed()

```
virtual int InterfaceBlock::LinkSpeed () [pure virtual]
```

Returns the link speed of the network interface.

#### Returns

10, 100, or 1000, representing the Mbps speed

#### See also

[InterfaceLinkSpeed\(\)](#)

### 16.96.3.9 RegisterInterface()

```
void InterfaceBlock::RegisterInterface ()
```

Register a network interface with system.

[InterfaceBlock](#) member function to register a new network interface

#### See also

[Removeinterface\(\)](#), [EnableMulticast\(\)](#), [DisableMulticast\(\)](#)

The documentation for this class was generated from the following file:

- `netinterface.h`

## 16.97 IPADDR4 Class Reference

Used to store and manipulate IPv4 addresses in dual stack mode.

```
#include <nettypes.h>
```

### Public Member Functions

- **IPADDR4** ()=default  
*Constructor to create an IPv4 object initialized to null.*
- **IPADDR4** (const **IPADDR4** &v)=default  
*Constructor to create an IPv4 object initialized to to the specified [IPADDR4](#) IP address.*
- bool **IsNull** () const  
*Check if the IP address is null.*
- bool **NotNull** () const  
*Check if the IP address is not null.*
- void **SetNull** ()  
*Set the IP address to null.*
- bool **IsLoopBack** () const  
*Check if the IP address is the loopback address for the interface.*
- bool **IsMultiCast** () const  
*Check if the [IPADDR4](#) object contains a Multicast IP address the interface.*
- bool **IsmDns** ()  
*Check if the [IPADDR4](#) object contains a mDNS IP address the interface.*
- bool **IsGlobalBroadCast** () const  
*Check if the [IPADDR4](#) object contains a global broadcast address: 255.255.255.255.*
- bool **IsAutoIP** ()  
*Check if the [IPADDR4](#) object contains an AutoIP address.*
- void **print** () const  
*Print an IPv4 address to the stdout serial port.*
- void **fdprint** (int fd) const

*Print an IPv4 address to the specified file descriptor.*

- int [sprintf](#) (char \*cp, int maxlen) const  
*sprintf an IPv4 address to the specified buffer*
- void [SetFromAscii](#) (const char \*cp)  
*Set the IPv4 address from a character string.*

### Static Public Member Functions

- static [IPADDR4 NullIP](#) ()  
*C++ static function for a null IP address.*
- static [IPADDR4 GlobalBroadcast](#) ()  
*C++ static function for a global broadcast IP address.*

### Friends

- class [CUR\\_IPADDR4](#)

## 16.97.1 Detailed Description

Used to store and manipulate IPv4 addresses in dual stack mode.

It is recommended to use the IPADDR object, which can hold either an IPv4 or IPv6 address, rather than explicitly specifying IPv4.

## 16.97.2 Constructor & Destructor Documentation

### 16.97.2.1 IPADDR4()

```
IPADDR4::IPADDR4 (
 const IPADDR4 & v) [default]
```

Constructor to create an IPv4 object initialized to to the specified [IPADDR4](#) IP address.

#### Parameters

|          |                                                         |
|----------|---------------------------------------------------------|
| <i>v</i> | <a href="#">IPADDR4</a> object used for initialization. |
|----------|---------------------------------------------------------|

## 16.97.3 Member Function Documentation

### 16.97.3.1 fdprint()

```
void IPADDR4::fdprint (
 int fd) const
```

Print an IPv4 address to the specified file descriptor.

#### Parameters

|           |                                         |
|-----------|-----------------------------------------|
| <i>fd</i> | File descriptor used to send IP address |
|-----------|-----------------------------------------|

### 16.97.3.2 IsAutoIP()

```
bool IPADDR4::IsAutoIP () [inline]
```

Check if the [IPADDR4](#) object contains an AutoIP address.

#### Return values

|             |                                              |
|-------------|----------------------------------------------|
| <i>true</i> | if the IP address value is an AutoIP address |
|-------------|----------------------------------------------|

See also

[IsLoopBack\(\)](#), [IsLinkLocal\(\)](#)

### 16.97.3.3 IsGlobalBroadcast()

```
bool IPADDR4::IsGlobalBroadcast () const [inline]
```

Check if the [IPADDR4](#) object contains a global broadcast address: 255.255.255.255.

Return values

|             |                                               |
|-------------|-----------------------------------------------|
| <i>true</i> | if the IP address value is a global broadcast |
|-------------|-----------------------------------------------|

See also

[IsLoopBack\(\)](#), [IsLinkLocal\(\)](#)

### 16.97.3.4 IsLoopBack()

```
bool IPADDR4::IsLoopBack () const [inline]
```

Check if the IP address is the loopback address for the interface.

Return values

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>true</i> | if the IP address value is the loopback address |
|-------------|-------------------------------------------------|

See also

[IsMultiCast\(\)](#), [IsLinkLocal\(\)](#)

### 16.97.3.5 IsmDns()

```
bool IPADDR4::IsmDns () [inline]
```

Check if the [IPADDR4](#) object contains a mDNS IP address the interface.

Return values

|             |                                           |
|-------------|-------------------------------------------|
| <i>true</i> | if the IP address value is a mDNS address |
|-------------|-------------------------------------------|

See also

[IsLoopBack\(\)](#), [IsLinkLocal\(\)](#)

### 16.97.3.6 IsMultiCast()

```
bool IPADDR4::IsMultiCast () const [inline]
```

Check if the [IPADDR4](#) object contains a Multicast IP address the interface.

Return values

|             |                                                |
|-------------|------------------------------------------------|
| <i>true</i> | if the IP address value is a Multicast address |
|-------------|------------------------------------------------|

See also

[IsLoopBack\(\)](#), [IsLinkLocal\(\)](#)

### 16.97.3.7 IsNull()

```
bool IPADDR4::IsNull () const [inline]
```

Check if the IP address is null.



## Return values

|             |                                 |
|-------------|---------------------------------|
| <i>true</i> | if the IP address value is null |
|-------------|---------------------------------|

## See also

[NotNull\(\)](#), [SetNull\(\)](#)**16.97.3.8 NotNull()**

```
bool IPADDR4::NotNull () const [inline]
```

Check if the IP address is not null.

## Return values

|             |                                     |
|-------------|-------------------------------------|
| <i>true</i> | if the IP address value is not null |
|-------------|-------------------------------------|

## See also

[IsNull\(\)](#), [SetNull\(\)](#)**16.97.3.9 SetFromAscii()**

```
void IPADDR4::SetFromAscii (
 const char * cp)
```

Set the IPv4 address from a character string.

## Parameters

|           |                                               |
|-----------|-----------------------------------------------|
| <i>cp</i> | Character string. For example, "192.168.1.10" |
|-----------|-----------------------------------------------|

**16.97.3.10 SetNull()**

```
void IPADDR4::SetNull () [inline]
```

Set the IP address to null.

## See also

[IsNull\(\)](#), [NotNull\(\)](#)**16.97.3.11 sprintf()**

```
int IPADDR4::sprintf (
 char * cp,
 int maxlen) const
```

sprintf an IPv4 address to the specified buffer

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>cp</i>   | Destination character buffer |
| <i>maxl</i> | Maximum number of characters |

## Returns

Number of characters

The documentation for this class was generated from the following file:

- `nettypes.h`

## 16.98 IPADDR6 Class Reference

Used to hold and manipulate IPv4 and IPv6 addresses in dual stack mode.

```
#include <ipv6_addr.h>
```

### Public Member Functions

- bool [IsEmbeddedIPv4](#) () const  
*An [IPADDR6](#) object can store a IPv4 or IPv6 address. This function returns true if the instance contains an IPv4 address.*
- [IPADDR4 Extract4](#) () const  
*Extracts an IPv4 address from the object.*
- uint32\_t [GetEmbeddedIP4UInt](#) () const  
*Get the embedded IPv4 address as a 32-bit unsigned integer.*
- bool [IsNull](#) () const  
*Check if the IP address is null.*
- bool [NotNull](#) () const  
*Check if the IP address is not null.*
- bool [IsLoopBack](#) () const  
*Check if the IP address is the loopback address for the interface.*
- bool [IsMultiCast](#) () const  
*Check if the [IPADDR6](#) object contains a Multicast IP address the interface.*
- bool [IsLinkLocal](#) () const  
*Check if the IP address is the link-local address for the interface.*
- [MACADR McastMac](#) () const  
*Return the MAC address used for Multicasts for the interface.*
- void [print](#) (bool bCompact=true, bool bShowV4Raw=false) const  
*Print the IP address value to stdout.*
- void [fdprint](#) (int fd, bool bCompact=true, bool bShowV4Raw=false) const  
*Print the IP address to the specified file descriptor.*
- int [sprintf](#) (char \*cp, int maxl, bool bCompact=true, bool bShowV4Raw=false) const  
*Print the IP address to the specified buffer.*
- void [SetFromAscii](#) (const char \*cp, bool bembed\_v4addresses=true)  
*Set the IP address value of an [IPADDR6](#) object.*
- void [SetFromIP4](#) ([IPADDR4](#) ip)  
*Set the IP address value of an [IPADDR6](#) object from an [IPADDR4](#) object.*
- void [SetFromUInt32Shortcut](#) (uint32\_t w0, uint32\_t w1, uint32\_t w2, uint32\_t w3)  
*Set the IP address value of an [IPADDR6](#) object from 4 discrete uint32\_t values.*
- void [SetNull](#) ()  
*Set the IP address value of an [IPADDR6](#) object to null.*

### Static Public Member Functions

- static [IPADDR6 AsciiToIp6](#) (const char \*cp, bool bembed\_v4addresses=true)  
*Static function to return an [IPADDR6](#) object created from an ASCII value IPv4 or IPv6 address.*
- static [IPADDR6 NullIP](#) ()  
*Static function to return a null [IPADDR6](#) object.*

### 16.98.1 Detailed Description

Used to hold and manipulate IPv4 and IPv6 addresses in dual stack mode.

#### Note

It is recommended to use the [IPADDR](#) object, which can hold either an IPv4 or IPv6 address, rather than explicitly specifying IPv6.

## 16.98.2 Member Function Documentation

### 16.98.2.1 AsciiToIp6()

```
static IPADDR6 IPADDR6::AsciiToIp6 (
 const char * cp,
 bool bembed_v4addresses = true) [static]
```

Static function to return an [IPADDR6](#) object created from an ASCII value IPv4 or IPv6 address.

#### Parameters

|                           |                                                                  |
|---------------------------|------------------------------------------------------------------|
| <i>*cp</i>                | Pointer to the ASCII string representing an IPv4 or IPv6 address |
| <i>bembed_v4addresses</i> | If false function will only process an IPv6 address              |

#### Return values

|                         |        |
|-------------------------|--------|
| <a href="#">IPADDR6</a> | object |
|-------------------------|--------|

#### See also

[NullIP\(\)](#)

### 16.98.2.2 Extract4()

```
IPADDR4 IPADDR6::Extract4 () const [inline]
```

Extracts an IPv4 address from the object.

This function extracts an IPv4 address from an [IPADDR6](#) object that contains an embedded IPv4 address. If the object contains an IPv6 address instead of an embedded IPv4, it returns a null IPv4 address (0.0.0.0).

#### Returns

[IPADDR4](#) object containing the extracted IPv4 address, or null (0.0.0.0) if the [IPADDR6](#) object does not contain an embedded IPv4 address

#### See also

[IsEmbeddedIPV4\(\)](#), [GetEmbeddedIP4UInt\(\)](#)

### 16.98.2.3 fdprint()

```
void IPADDR6::fdprint (
 int fd,
 bool bCompact = true,
 bool bShowV4Raw = false) const
```

Print the IP address to the specified file descriptor.

#### Parameters

|                   |                                                                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>fd</i>         | Valid file descriptor to send the data                                                                                                        |
| <i>bCompact</i>   | Display IPv6 address in compact notation                                                                                                      |
| <i>bShowV4Raw</i> | Normally set to false. If set to true, will display the IPv4 address 128-bits of the IP address object in the raw format "::FFFF:xx.xx.xx.xx" |

#### See also

[print\(\)](#), [sprintf\(\)](#)

#### 16.98.2.4 GetEmbeddedIP4Uint()

```
uint32_t IPADDR6::GetEmbeddedIP4Uint () const [inline]
```

Get the embedded IPv4 address as a 32-bit unsigned integer.

This function returns the embedded IPv4 address as a raw 32-bit unsigned integer value. This is useful when you need the IPv4 address in its numeric form for calculations or comparisons.

##### Returns

32-bit unsigned integer representation of the IPv4 address, or 0 if the [IPADDR6](#) object does not contain an embedded IPv4 address

##### Note

The returned value is in natural order as the `beuint32_t` cast converts.

##### See also

[Extract4\(\)](#), [IsEmbeddedIPV4\(\)](#)

#### 16.98.2.5 IsEmbeddedIPV4()

```
bool IPADDR6::IsEmbeddedIPV4 () const [inline]
```

An [IPADDR6](#) object can store a IPv4 or IPv6 address. This function returns true if the instance contains an IPv4 address.

Internally, an IPv4 address is stored in the format: "::FFFF:xx.xx.xx.xx"

##### Return values

|              |                                                                 |
|--------------|-----------------------------------------------------------------|
| <i>true</i>  | If the <a href="#">IPADDR6</a> object contains and IPv4 address |
| <i>false</i> | Otherwise                                                       |

#### 16.98.2.6 IsLinkLocal()

```
bool IPADDR6::IsLinkLocal () const [inline]
```

Check if the IP address is the link-local address for the interface.

Link local is

`FE80::/10`

. Masks the top 10 bits and check.

##### Return values

|             |                                                   |
|-------------|---------------------------------------------------|
| <i>true</i> | if the IP address value is the link-local address |
|-------------|---------------------------------------------------|

##### See also

[IsLoopBack\(\)](#), [IsMultiCast\(\)](#)

#### 16.98.2.7 IsLoopBack()

```
bool IPADDR6::IsLoopBack () const [inline]
```

Check if the IP address is the loopback address for the interface.

##### Return values

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>true</i> | if the IP address value is the loopback address |
|-------------|-------------------------------------------------|

##### See also

[IsMultiCast\(\)](#), [IsLinkLocal\(\)](#)

**16.98.2.8 IsMultiCast()**

```
bool IPADDR6::IsMultiCast () const [inline]
```

Check if the [IPADDR6](#) object contains a Multicast IP address the interface.

Return values

|             |                                                |
|-------------|------------------------------------------------|
| <i>true</i> | if the IP address value is a Multicast address |
|-------------|------------------------------------------------|

See also

[IsLoopBack\(\)](#), [IsLinkLocal\(\)](#)

**16.98.2.9 IsNull()**

```
bool IPADDR6::IsNull () const [inline]
```

Check if the IP address is null.

Return values

|             |                                 |
|-------------|---------------------------------|
| <i>true</i> | if the IP address value is null |
|-------------|---------------------------------|

See also

[NotNull\(\)](#)

**16.98.2.10 McastMac()**

```
MACADDR IPADDR6::McastMac () const
```

Return the MAC address used for Multicasts for the interface.

Return values

|                         |                               |
|-------------------------|-------------------------------|
| <a href="#">MACADDR</a> | if a Multicast address exists |
| <i>null</i>             | otherwise                     |

**16.98.2.11 NotNull()**

```
bool IPADDR6::NotNull () const [inline]
```

Check if the IP address is not null.

Return values

|             |                                     |
|-------------|-------------------------------------|
| <i>true</i> | if the IP address value is not null |
|-------------|-------------------------------------|

See also

[IsNull\(\)](#)

**16.98.2.12 NullIP()**

```
static IPADDR6 IPADDR6::NullIP () [static]
```

Static function to return a null [IPADDR6](#) object.

Static functions can be called without a specific C++ object. For example,

```
if (myIPAddress == IPADDR6::NullIP())
 iprintf("myIPAddress is null\r\n");
else
 iprintf("myIPAddress = %I\r\n", myIPAddress);
```

[IPADDR6::NullIP\(\)](#) can be used on [IPADDR](#), [IPADDR4](#) and [IPADDR6](#) objects.

## Return values

|             |                                |
|-------------|--------------------------------|
| <i>null</i> | <a href="#">IPADDR6</a> object |
|-------------|--------------------------------|

## See also

[AsciiToIp6\(\)](#)**16.98.2.13 print()**

```
void IPADDR6::print (
 bool bCompact = true,
 bool bShowV4Raw = false) const
```

Print the IP address value to stdout.

## Parameters

|                   |                                                                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>bCompact</i>   | Display IPv6 address in compact notation                                                                                                      |
| <i>bShowV4Raw</i> | Normally set to false. if set to true, will display the IPv4 address 128-bits of the IP address object in the raw format "::FFFF:xx.xx.xx.xx" |

## See also

[fdprint\(\)](#), [sprintf\(\)](#)**16.98.2.14 SetFromAscii()**

```
void IPADDR6::SetFromAscii (
 const char * cp,
 bool bembed_v4addresses = true)
```

Set the IP address value of an [IPADDR6](#) object.

## Parameters

|                           |                                                                  |
|---------------------------|------------------------------------------------------------------|
| <i>*cp</i>                | Pointer to the ASCII string representing an IPv4 or IPv6 address |
| <i>bembed_v4addresses</i> | If false function will only process an IPv6 address              |

## See also

[SetFromIP4\(\)](#), [SetNull\(\)](#)**16.98.2.15 SetFromIP4()**

```
void IPADDR6::SetFromIP4 (
 IPADDR4 ip)
```

Set the IP address value of an [IPADDR6](#) object from an [IPADDR4](#) object.

## Parameters

|           |                                |
|-----------|--------------------------------|
| <i>ip</i> | <a href="#">IPADDR4</a> object |
|-----------|--------------------------------|

## See also

[SetFromAscii\(\)](#), [SetNull\(\)](#)

**16.98.2.16 SetFromUint32Shortcut()**

```
void IPADDR6::SetFromUint32Shortcut (
 uint32_t w0,
 uint32_t w1,
 uint32_t w2,
 uint32_t w3) [inline]
```

Set the IP address value of an [IPADDR6](#) object from 4 discrete uint32\_t values.

**Parameters**

|           |                         |
|-----------|-------------------------|
| <i>w0</i> | Unsigned 32-bit integer |
| <i>w1</i> | Unsigned 32-bit integer |
| <i>w2</i> | Unsigned 32-bit integer |
| <i>w3</i> | Unsigned 32-bit integer |

**See also**

[SetFromAscii\(\)](#), [SetFromIP4\(\)](#), [SetNull\(\)](#)

**16.98.2.17 SetNull()**

```
void IPADDR6::SetNull () [inline]
```

Set the IP address value of an [IPADDR6](#) object to null.

**See also**

[SetFromAscii\(\)](#), [SetFromIP4\(\)](#)

**16.98.2.18 sprintf()**

```
int IPADDR6::sprintf (
 char * cp,
 int maxl,
 bool bCompact = true,
 bool bShowV4Raw = false) const
```

Print the IP address to the specified buffer.

**Parameters**

|                   |                                                                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>*cp</i>        | Pointer to the destination buffer to store the data                                                                                           |
| <i>maxl</i>       | Maximum number of bytes to write to buffer                                                                                                    |
| <i>bCompact</i>   | Display IPv6 address in compact notation                                                                                                      |
| <i>bShowV4Raw</i> | Normally set to false. if set to true, will display the IPv4 address 128-bits of the IP address object in the raw format "::FFFF:xx.xx.xx.xx" |

**Returns**

Number of bytes copied to buffer

**See also**

[print\(\)](#), [fdprint\(\)](#)

The documentation for this class was generated from the following file:

- `ipv6_addr.h`

## 16.99 IPPKT Struct Reference

Internal IPv4 Header Representation.

```
#include <ip.h>
```

### Public Attributes

- `uint8_t bVerHdrLen`  
*Version and header length (same as versionNLength in IpHeaderIPv4)*
- `uint8_t bTos`  
*Type of Service / Differentiated Services field (same as diffServNEcn)*
- `beuint16_t wLength`  
*Total packet length in bytes including header.*
- `beuint16_t wpktId`  
*Packet identification for fragment reassembly.*
- `beuint16_t wFlags_Frag`  
*Flags and fragment offset combined.*
- `uint8_t bTTL`  
*Time To Live / hop count.*
- `uint8_t proto`  
*Protocol number (TCP, UDP, ICMP, etc.)*
- `uint16_t hCSum`  
*Header checksum.*
- `IPADDR4 ipSrc`  
*Source IP address.*
- `IPADDR4 ipDst`  
*Destination IP address.*
- `uint8_t DATA []`  
*Flexible array member - payload data follows header (actual length varies by packet)*

### 16.99.1 Detailed Description

Internal IPv4 Header Representation.

This is the internal structure used throughout the NetBurner IP stack for manipulating IPv4 packets. It provides a slightly different layout than the standard `IpHeaderIPv4` structure for implementation convenience.

The `DATA[]` flexible array member allows direct access to the packet payload that follows the IP header.

#### Note

This structure is packed to ensure proper alignment with network packet data.

The `DATA[]` field extends beyond the structure to the actual packet length.

### 16.99.2 Member Data Documentation

#### 16.99.2.1 hCSum

```
uint16_t IPPKT::hCSum
```

Header checksum.

#### Warning

Do not make a big endian variable - checksumming is treated as native endian

The documentation for this struct was generated from the following file:

- `ip.h`



## 16.100 IPV6\_DEST Struct Reference

IPv6 destination cache element.

#include <ipv6\_interface.h>

Inherits [IPV6\\_ROOT\\_EL](#).

### Public Member Functions

- [IPV6\\_DEST](#) \* [GetNext](#) ()  
*Get next destination in list.*
- void [free\\_element](#) ()  
*Free resources for this destination.*
- virtual bool [AgeStillValidTest](#) ()  
*Check if destination is still valid.*
- virtual void [ShowItem](#) ()  
*Display destination information.*

### Public Member Functions inherited from [IPV6\\_ROOT\\_EL](#)

- virtual ~[IPV6\\_ROOT\\_EL](#) ()  
*Virtual destructor.*

### Public Attributes

- [IPV6\\_NEIGHBOR](#) \* [m\\_pNeighbor](#)  
*Associated neighbor entry.*
- uint16\_t [m\\_PathMTU](#)  
*Path MTU to this destination.*
- uint32\_t [m\\_nSecsLastUsed](#)  
*Last time this destination was used.*
- int [m\\_nLockCount](#)  
*Reference count to prevent destruction.*
- bool [m\\_bStaticRoute](#)  
*True if this is a static route.*

### Public Attributes inherited from [IPV6\\_ROOT\\_EL](#)

- [IPV6\\_ROOT\\_EL](#) \* [m\\_pNext](#)  
*Next element in list.*
- [IPV6\\_ROOT\\_EL](#) \* [m\\_pPrev](#)  
*Previous element in list.*
- [IPV6\\_ROOT\\_EL\\_CONTAINER](#) \* [m\\_pContainer](#)  
*Container that owns this element.*
- [IPADDR6](#) [m\\_IPAddress](#)  
*IPv6 address associated with this element.*

#### 16.100.1 Detailed Description

IPv6 destination cache element.

Caches information about IPv6 destinations including path MTU and associated neighbor.

## 16.100.2 Member Function Documentation

### 16.100.2.1 AgeStillValidTest()

```
virtual bool IPV6_DEST::AgeStillValidTest () [virtual]
```

Check if destination is still valid.

Returns

true if destination is still valid

Reimplemented from [IPV6\\_ROOT\\_EL](#).

### 16.100.2.2 GetNext()

```
IPV6_DEST * IPV6_DEST::GetNext () [inline]
```

Get next destination in list.

Returns

Pointer to next destination or NULL

The documentation for this struct was generated from the following file:

- [ipv6\\_interface.h](#)

## 16.101 IPV6\_DHCPD Struct Reference

DHCPv6 server information element.

```
#include <ipv6_interface.h>
```

Inherits [IPV6\\_ROOT\\_EL](#).

### Public Member Functions

- void **free\_element** ()  
*Free resources for this [DHCP](#) server element.*
- virtual bool **AgeStillValidTest** ()  
*Check if [DHCP](#) lease is still valid.*
- virtual void **ShowItem** ()  
*Display [DHCP](#) server information.*
- [IPV6\\_PREFIX](#) \* **FindIA\_Addr** ([IPV6\\_PREFIX](#) \*start)  
*Find associated IA address starting from given prefix.*
- uint32\_t **GetDhcpRenewTime** ()  
*Get [DHCP](#) renew time.*
- uint32\_t **GetDhcpRebindTime** ()  
*Get [DHCP](#) rebind time.*

### Public Member Functions inherited from [IPV6\\_ROOT\\_EL](#)

- virtual **~IPV6\_ROOT\_EL** ()  
*Virtual destructor.*

### Public Attributes

- uint8\_t **m\_ServerID** [CLIENT\_LONG\_SERVID]  
*Space to contain the ServerID of the leasing server.*
- uint32\_t **m\_IAID**  
*Identity Association Identifier.*
- uint32\_t **m\_renewTick**  
*Tick when lease should be renewed.*
- uint32\_t **m\_rebindTick**  
*Tick when lease should be rebound.*

## Public Attributes inherited from [IPV6\\_ROOT\\_EL](#)

- [IPV6\\_ROOT\\_EL](#) \* **m\_pNext**  
*Next element in list.*
- [IPV6\\_ROOT\\_EL](#) \* **m\_pPrev**  
*Previous element in list.*
- [IPV6\\_ROOT\\_EL\\_CONTAINER](#) \* **m\_pContainer**  
*Container that owns this element.*
- [IPADDR6](#) **m\_IPAddress**  
*IPv6 address associated with this element.*

### 16.101.1 Detailed Description

DHCPv6 server information element.

Stores information about DHCPv6 servers including lease timing and server identification.

### 16.101.2 Member Function Documentation

#### 16.101.2.1 AgeStillValidTest()

```
virtual bool IPV6_DHCPD::AgeStillValidTest () [virtual]
```

Check if [DHCP](#) lease is still valid.

Returns

true if lease is still valid

Reimplemented from [IPV6\\_ROOT\\_EL](#).

#### 16.101.2.2 FindIA\_Addr()

```
IPV6_PREFIX * IPV6_DHCPD::FindIA_Addr (
 IPV6_PREFIX * start)
```

Find associated IA address starting from given prefix.

Parameters

|              |                            |
|--------------|----------------------------|
| <i>start</i> | Starting prefix for search |
|--------------|----------------------------|

Returns

Pointer to found prefix or NULL

#### 16.101.2.3 GetDhcpRebindTime()

```
uint32_t IPV6_DHCPD::GetDhcpRebindTime () [inline]
```

Get [DHCP](#) rebind time.

Returns

Rebind time in ticks

#### 16.101.2.4 GetDhcpRenewTime()

```
uint32_t IPV6_DHCPD::GetDhcpRenewTime () [inline]
```

Get [DHCP](#) renew time.

Returns

Renew time in ticks

The documentation for this struct was generated from the following file:

- [ipv6\\_interface.h](#)

## 16.102 IPV6\_DNS Struct Reference

IPv6 DNS server information element.

#include <ipv6\_interface.h>

Inherits [IPV6\\_ROOT\\_EL](#).

### Public Member Functions

- [IPV6\\_DNS](#) \* **GetNext** ()  
*Get next DNS server in list.*
- virtual bool **AgeStillValidTest** ()  
*Check if DNS server is still valid.*
- virtual void **ShowItem** ()  
*Display DNS server information.*
- void **CleanUpAndRemove** ()  
*Clean up and remove DNS server from lists.*
- void **free\_element** ()  
*Free resources for this DNS server.*

### Public Member Functions inherited from [IPV6\\_ROOT\\_EL](#)

- virtual ~**IPV6\_ROOT\_EL** ()  
*Virtual destructor.*

### Public Attributes

- uint32\_t **m\_nSecsToBeValid**  
*Seconds until this DNS entry expires.*
- [IPV6\\_ROUTER](#) \* **pRouter**  
*Router that provided this DNS server.*
- [IPV6\\_DHCPD](#) \* **pDHCPD**  
*DHCP server that provided this DNS server.*

### Public Attributes inherited from [IPV6\\_ROOT\\_EL](#)

- [IPV6\\_ROOT\\_EL](#) \* **m\_pNext**  
*Next element in list.*
- [IPV6\\_ROOT\\_EL](#) \* **m\_pPrev**  
*Previous element in list.*
- [IPV6\\_ROOT\\_EL\\_CONTAINER](#) \* **m\_pContainer**  
*Container that owns this element.*
- [IPADDR6](#) **m\_IPAddress**  
*IPv6 address associated with this element.*

#### 16.102.1 Detailed Description

IPv6 DNS server information element.

Stores information about configured DNS servers including validity timing.

## 16.102.2 Member Function Documentation

### 16.102.2.1 AgeStillValidTest()

```
virtual bool IPV6_DNS::AgeStillValidTest () [virtual]
```

Check if DNS server is still valid.

#### Returns

true if DNS server is still valid

Reimplemented from [IPV6\\_ROOT\\_EL](#).

### 16.102.2.2 GetNext()

```
IPV6_DNS * IPV6_DNS::GetNext () [inline]
```

Get next DNS server in list.

#### Returns

Pointer to next DNS server

The documentation for this struct was generated from the following file:

- [ipv6\\_interface.h](#)

## 16.103 IPV6\_IA\_ADDR Struct Reference

DHCPv6 Identity Association address element.

```
#include <ipv6_interface.h>
```

Inherits [IPV6\\_ROOT\\_EL](#).

### Public Member Functions

- void **free\_element** ()  
*Free resources for this IA address.*
- virtual bool [AgeStillValidTest](#) ()  
*Check if IA address is still valid.*
- virtual void **ShowItem** ()  
*Display IA address information.*

### Public Member Functions inherited from [IPV6\\_ROOT\\_EL](#)

- virtual  $\sim$ [IPV6\\_ROOT\\_EL](#) ()  
*Virtual destructor.*

### Additional Inherited Members

### Public Attributes inherited from [IPV6\\_ROOT\\_EL](#)

- [IPV6\\_ROOT\\_EL](#) \* **m\_pNext**  
*Next element in list.*
- [IPV6\\_ROOT\\_EL](#) \* **m\_pPrev**  
*Previous element in list.*
- [IPV6\\_ROOT\\_EL\\_CONTAINER](#) \* **m\_pContainer**  
*Container that owns this element.*
- [IPADDR6](#) **m\_IPAddress**  
*IPv6 address associated with this element.*

### 16.103.1 Detailed Description

DHCPv6 Identity Association address element.

### 16.103.2 Member Function Documentation

#### 16.103.2.1 AgeStillValidTest()

```
virtual bool IPV6_IA_ADDR::AgeStillValidTest () [virtual]
```

Check if IA address is still valid.

Returns

true if address is still valid

Reimplemented from [IPV6\\_ROOT\\_EL](#).

The documentation for this struct was generated from the following file:

- [ipv6\\_interface.h](#)

## 16.104 IPV6\_NEIGHBOR Struct Reference

IPv6 neighbor cache element.

```
#include <ipv6_interface.h>
```

Inherits [IPV6\\_ROOT\\_EL](#).

### Public Member Functions

- **IPV6\_NEIGHBOR** ()  
*Default constructor.*
- [IPV6\\_NEIGHBOR](#) \* **GetNext** ()  
*Get next neighbor in list.*
- void **free\_element** ()  
*Free resources for this neighbor.*
- void **Discard** ()  
*Discard pending packets and clean up.*
- bool **ProcessAdvert** ([IPV6FrameProcessingStruct](#) &p6proc, ICMP6\_ND\_ADVERT \*pRsp)  
*Process neighbor advertisement message.*
- void **ProcessTick** ()  
*Process timer tick for neighbor discovery.*
- void **Send\_ND\_Solicit** (bool multicast)  
*Send neighbor solicitation message.*
- **eRouteOutResult** **Send** ([IPV6FrameProcessingStruct](#) &p6proc, uint16\_t mtu)  
*Attempt to send packet via this neighbor.*
- virtual bool **AgeStillValidTest** ()  
*Check if neighbor is still valid.*
- bool **StillValidToSend** ()  
*Check if neighbor is still valid for sending.*
- void **SetState** ([eMY\\_NEIGHBOR\\_STATE](#) s)  
*Set neighbor discovery state.*
- [eMY\\_NEIGHBOR\\_STATE](#) **GetState** ()  
*Get current neighbor state.*
- void **CoreSendPend** ([PoolIPtr](#) pp)  
*Queue packet for transmission when neighbor becomes reachable.*
- bool **SendPending** ()  
*Send any pending packets.*
- virtual void **ShowItem** ()  
*Display neighbor information.*

## Public Member Functions inherited from [IPV6\\_ROOT\\_EL](#)

- `virtual ~IPV6_ROOT_EL ()`  
*Virtual destructor.*

## Public Attributes

- `MACADR m_Macaddr`  
*MAC address of this neighbor.*
- `eMY_NEIGHBOR_STATE m_NeighborState`  
*Current neighbor discovery state.*
- `uint8_t m_SentNDCount`  
*Number of neighbor solicitations sent.*
- `uint32_t m_TickTimeOfNextAction`  
*When next action should be taken.*
- `bool m_bActiveTimer`  
*Whether timer processing is active.*
- `IPV6_ROUTER * m_pRouter`  
*Associated router entry.*
- `PoolPtr m_pOutBound1`  
*First pending outbound packet.*
- `PoolPtr m_pOutBound2`  
*Second pending outbound packet.*

## Public Attributes inherited from [IPV6\\_ROOT\\_EL](#)

- `IPV6_ROOT_EL * m_pNext`  
*Next element in list.*
- `IPV6_ROOT_EL * m_pPrev`  
*Previous element in list.*
- `IPV6_ROOT_EL_CONTAINER * m_pContainer`  
*Container that owns this element.*
- `IPADDR6 m_IPAddress`  
*IPv6 address associated with this element.*

### 16.104.1 Detailed Description

IPv6 neighbor cache element.

Manages neighbor discovery state and MAC address resolution for IPv6 addresses.

### 16.104.2 Member Function Documentation

#### 16.104.2.1 AgeStillValidTest()

```
virtual bool IPV6_NEIGHBOR::AgeStillValidTest () [virtual]
```

Check if neighbor is still valid.

Returns

true if neighbor is still valid

Reimplemented from [IPV6\\_ROOT\\_EL](#).

#### 16.104.2.2 CoreSendPend()

```
void IPV6_NEIGHBOR::CoreSendPend (
 PoolPtr pp)
```

Queue packet for transmission when neighbor becomes reachable.

## Parameters

|           |                 |
|-----------|-----------------|
| <i>pp</i> | Packet to queue |
|-----------|-----------------|

**16.104.2.3 GetNext()**

```
IPV6_NEIGHBOR * IPV6_NEIGHBOR::GetNext () [inline]
```

Get next neighbor in list.

## Returns

Pointer to next neighbor or NULL

**16.104.2.4 GetState()**

```
eMY_NEIGHBOR_STATE IPV6_NEIGHBOR::GetState () [inline]
```

Get current neighbor state.

## Returns

Current neighbor state

**16.104.2.5 ProcessAdvert()**

```
bool IPV6_NEIGHBOR::ProcessAdvert (
 IPv6FrameProcessingStruct & p6proc,
 ICMP6_ND_ADVERT * pRsp)
```

Process neighbor advertisement message.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>p6proc</i> | IPv6 frame processing structure   |
| <i>pRsp</i>   | Pointer to neighbor advertisement |

## Returns

true if advertisement was processed successfully

**16.104.2.6 Send()**

```
eRouteOutResult IPV6_NEIGHBOR::Send (
 IPv6FrameProcessingStruct & p6proc,
 uint16_t mtu)
```

Attempt to send packet via this neighbor.

## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>p6proc</i> | IPv6 frame processing structure |
| <i>mtu</i>    | Maximum transmission unit       |

## Returns

Route result status

**16.104.2.7 Send\_ND\_Solicit()**

```
void IPV6_NEIGHBOR::Send_ND_Solicit (
 bool multicast)
```

Send neighbor solicitation message.



## Parameters

|                  |                                                        |
|------------------|--------------------------------------------------------|
| <i>multicast</i> | true to send multicast solicitation, false for unicast |
|------------------|--------------------------------------------------------|

**16.104.2.8 SendPending()**

`bool IPV6_NEIGHBOR::SendPending ()`  
 Send any pending packets.

## Returns

true if packets were sent

**16.104.2.9 SetState()**

`void IPV6_NEIGHBOR::SetState (`  
     `eMY_NEIGHBOR_STATE s)`  
 Set neighbor discovery state.

## Parameters

|          |                    |
|----------|--------------------|
| <i>s</i> | New neighbor state |
|----------|--------------------|

**16.104.2.10 StillValidToSend()**

`bool IPV6_NEIGHBOR::StillValidToSend ()`  
 Check if neighbor is still valid for sending.

## Returns

true if neighbor can be used for sending

The documentation for this struct was generated from the following file:

- `ipv6_interface.h`

**16.105 IPV6\_PREFIX Struct Reference**

IPv6 prefix and address management element.

`#include <ipv6_interface.h>`

Inherits [IPV6\\_ROOT\\_EL](#).

**Public Member Functions**

- [IPV6\\_PREFIX](#) \* `GetNext ()`  
*Get next prefix in list.*
- [IPV6\\_PREFIX](#) ()  
*Default constructor.*
- void **CleanUpAndRemove** ()  
*Clean up and remove prefix from lists.*
- void **free\_element** ()  
*Free resources for this prefix.*
- bool [OnLink](#) (const [IPADDR6](#) &ip)  
*Check if given IP address is on-link for this prefix.*
- bool [Preferred](#) ()  
*Check if this prefix is preferred.*

- [eMY\\_ADDR\\_STATE](#) **GetState** ()  
*Get current state of this address.*
- virtual void **ShowItem** ()  
*Display prefix information.*
- [ePrefixSource](#) **Source** ()  
*Get the source of this prefix.*
- uint32\_t **RemainingValidTime** ()  
*Get remaining valid time for this prefix.*
- uint32\_t **GetRemainingDhcpLeaseTime** ()  
*Get remaining [DHCP](#) lease time.*
- uint32\_t **GetRemainingDhcpLeasePreferredTime** ()  
*Get remaining [DHCP](#) preferred time.*
- virtual bool **AgeStillValidTest** ()  
*Check if prefix is still valid based on aging.*
- bool **AgeStillPreferred** ()  
*Check if prefix is still preferred based on aging.*
- void **CheckTenative** ()  
*Check tentative state and perform duplicate address detection.*

### Public Member Functions inherited from [IPV6\\_ROOT\\_EL](#)

- virtual **~IPV6\_ROOT\_EL** ()  
*Virtual destructor.*

### Public Attributes

- [eMY\\_ADDR\\_STATE](#) **m\_state**  
*Current state of this address.*
- uint32\_t **time\_to\_lose\_prefered\_in\_secs\_from\_establish**  
*Preferred lifetime from establishment.*
- uint32\_t **time\_to\_lose\_valid\_in\_secs\_from\_establish**  
*Valid lifetime from establishment.*
- uint32\_t **time\_established\_in\_secs**  
*When this prefix was established.*
- uint32\_t **max\_valid\_time\_seen**  
*Maximum valid time seen (for IOL cleanup)*
- [IPV6\\_ROUTER](#) \* **pRouter**  
*Router that established this prefix.*
- [IPV6\\_DHCPD](#) \* **pDHCPD**  
*[DHCP](#) server that established this prefix.*
- uint8\_t **PrefixLen**  
*Prefix length in bits.*
- bool **bValidForInterface**  
*Valid for this interface.*
- bool **bOnLink**  
*On-link flag.*

## Public Attributes inherited from [IPV6\\_ROOT\\_EL](#)

- [IPV6\\_ROOT\\_EL](#) \* **m\_pNext**  
*Next element in list.*
- [IPV6\\_ROOT\\_EL](#) \* **m\_pPrev**  
*Previous element in list.*
- [IPV6\\_ROOT\\_EL\\_CONTAINER](#) \* **m\_pContainer**  
*Container that owns this element.*
- [IPADDR6](#) **m\_IPAddress**  
*IPv6 address associated with this element.*

### 16.105.1 Detailed Description

IPv6 prefix and address management element.

Manages IPv6 prefixes and addresses including their validity, timing, and source information.

### 16.105.2 Member Function Documentation

#### 16.105.2.1 AgeStillPreferred()

```
bool IPV6_PREFIX::AgeStillPreferred ()
```

Check if prefix is still preferred based on aging.

Returns

true if prefix is still preferred

#### 16.105.2.2 AgeStillValidTest()

```
virtual bool IPV6_PREFIX::AgeStillValidTest () [virtual]
```

Check if prefix is still valid based on aging.

Returns

true if prefix is still valid

Reimplemented from [IPV6\\_ROOT\\_EL](#).

#### 16.105.2.3 GetNext()

```
IPV6_PREFIX * IPV6_PREFIX::GetNext () [inline]
```

Get next prefix in list.

Returns

Pointer to next prefix

#### 16.105.2.4 GetRemainingDhcpLeasePreferredTime()

```
uint32_t IPV6_PREFIX::GetRemainingDhcpLeasePreferredTime () [inline]
```

Get remaining [DHCP](#) preferred time.

Returns

Remaining preferred time in seconds

#### 16.105.2.5 GetRemainingDhcpLeaseTime()

```
uint32_t IPV6_PREFIX::GetRemainingDhcpLeaseTime () [inline]
```

Get remaining [DHCP](#) lease time.

Returns

Remaining lease time in seconds

**16.105.2.6 GetState()**

```
eMY_ADDR_STATE IPV6_PREFIX::GetState () [inline]
```

Get current state of this address.

**Returns**

Current address state

**16.105.2.7 OnLink()**

```
bool IPV6_PREFIX::OnLink (
 const IPADDR6 & ip)
```

Check if given IP address is on-link for this prefix.

**Parameters**

|           |                       |
|-----------|-----------------------|
| <i>ip</i> | IPv6 address to check |
|-----------|-----------------------|

**Returns**

true if address is on-link

**16.105.2.8 Preferred()**

```
bool IPV6_PREFIX::Preferred ()
```

Check if this prefix is preferred.

**Returns**

true if prefix is preferred

**16.105.2.9 RemainingValidTime()**

```
uint32_t IPV6_PREFIX::RemainingValidTime () [inline]
```

Get remaining valid time for this prefix.

**Returns**

Remaining valid time in seconds

**16.105.2.10 Source()**

```
ePrefixSource IPV6_PREFIX::Source () [inline]
```

Get the source of this prefix.

**Returns**

Prefix source type

The documentation for this struct was generated from the following file:

- `ipv6_interface.h`

**16.106 IPV6\_ROOT\_EL Struct Reference**

Base class for all IPv6 interface elements.

```
#include <ipv6_interface.h>
```

Inherited by [IPV6\\_DEST](#), [IPV6\\_DHCPD](#), [IPV6\\_DNS](#), [IPV6\\_IA\\_ADDR](#), [IPV6\\_NEIGHBOR](#), [IPV6\\_PREFIX](#), and [IPV6\\_ROUTER](#).

### Public Member Functions

- virtual void **ShowItem** ()  
*Display information about this element.*
- virtual bool **AgeStillValidTest** ()  
*Test if this element is still valid based on aging.*
- virtual void **free\_element** ()  
*Free resources associated with this element.*
- virtual **~IPV6\_ROOT\_EL** ()  
*Virtual destructor.*

### Public Attributes

- **IPV6\_ROOT\_EL** \* **m\_pNext**  
*Next element in list.*
- **IPV6\_ROOT\_EL** \* **m\_pPrev**  
*Previous element in list.*
- **IPV6\_ROOT\_EL\_CONTAINER** \* **m\_pContainer**  
*Container that owns this element.*
- **IPADDR6** **m\_IPAddress**  
*IPv6 address associated with this element.*

## 16.106.1 Detailed Description

Base class for all IPv6 interface elements.

Provides common functionality for all IPv6 interface management elements including linked list management and aging.

## 16.106.2 Member Function Documentation

### 16.106.2.1 AgeStillValidTest()

```
virtual bool IPV6_ROOT_EL::AgeStillValidTest () [virtual]
```

Test if this element is still valid based on aging.

#### Returns

true if element is still valid, false if expired

Reimplemented in [IPV6\\_DEST](#), [IPV6\\_DHCPD](#), [IPV6\\_DNS](#), [IPV6\\_IA\\_ADDR](#), [IPV6\\_NEIGHBOR](#), [IPV6\\_PREFIX](#), and [IPV6\\_ROUTER](#).

The documentation for this struct was generated from the following file:

- [ipv6\\_interface.h](#)

## 16.107 IPV6\_ROOT\_EL\_CONTAINER Struct Reference

Container for managing collections of IPv6 root elements.

```
#include <ipv6_interface.h>
```

### Public Member Functions

- void **Insert** ([IPV6\\_ROOT\\_EL](#) \*pEl)  
*Insert element at the head of the list.*
- void **InsertBack** ([IPV6\\_ROOT\\_EL](#) \*pEl)  
*Insert element at the tail of the list.*
- [IPV6\\_ROOT\\_EL](#) \* **Find** (const [IPADDR6](#) &ip, bool age)

- Find element by IPv6 address.*
- `IPV6_ROOT_EL * First ()`  
*Get first element in container.*
- `IPV6_ROOT_EL_CONTAINER ()`  
*Default constructor.*
- `void Attach (IPv6Interface &Interface, const char *name)`  
*Attach container to an interface.*
- `void MoveToTop (IPV6_ROOT_EL *pEl)`  
*Move element to top of list.*
- `void Remove (IPV6_ROOT_EL *pEl)`  
*Remove element from container.*
- `void Show ()`  
*Display all elements in container.*

### Public Attributes

- `const char * m_name`  
*Name of this container.*
- `IPV6_ROOT_EL * m_pHead`  
*Head of element list.*
- `IPV6_ROOT_EL * m_pTail`  
*Tail of element list.*
- `IPv6Interface * m_pInterface`  
*Interface that owns this container.*
- `int m_nElements`  
*Number of elements in container.*

## 16.107.1 Detailed Description

Container for managing collections of IPv6 root elements.

Provides organized storage and retrieval of IPv6 interface elements with automatic aging and management functionality.

## 16.107.2 Member Function Documentation

### 16.107.2.1 Attach()

```
void IPV6_ROOT_EL_CONTAINER::Attach (
 IPv6Interface & Interface,
 const char * name)
```

Attach container to an interface.

#### Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>Interface</i> | IPv6 interface to attach to |
| <i>name</i>      | Name for this container     |

### 16.107.2.2 Find()

```
IPV6_ROOT_EL * IPV6_ROOT_EL_CONTAINER::Find (
 const IPADDR6 & ip,
 bool age)
```

Find element by IPv6 address.

## Parameters

|            |                                              |
|------------|----------------------------------------------|
| <i>ip</i>  | IPv6 address to search for                   |
| <i>age</i> | Whether to perform aging check during search |

## Returns

Pointer to found element or NULL if not found

**16.107.2.3 First()**

```
IPV6_ROOT_EL * IPV6_ROOT_EL_CONTAINER::First ()
```

Get first element in container.

## Returns

Pointer to first element or NULL if empty

**16.107.2.4 Insert()**

```
void IPV6_ROOT_EL_CONTAINER::Insert (
 IPV6_ROOT_EL * pEl)
```

Insert element at the head of the list.

## Parameters

|                                  |                   |
|----------------------------------|-------------------|
| $p \leftrightarrow$<br><i>El</i> | Element to insert |
|----------------------------------|-------------------|

**16.107.2.5 InsertBack()**

```
void IPV6_ROOT_EL_CONTAINER::InsertBack (
 IPV6_ROOT_EL * pEl)
```

Insert element at the tail of the list.

## Parameters

|                                  |                   |
|----------------------------------|-------------------|
| $p \leftrightarrow$<br><i>El</i> | Element to insert |
|----------------------------------|-------------------|

**16.107.2.6 MoveToTop()**

```
void IPV6_ROOT_EL_CONTAINER::MoveToTop (
 IPV6_ROOT_EL * pEl)
```

Move element to top of list.

## Parameters

|                                  |                 |
|----------------------------------|-----------------|
| $p \leftrightarrow$<br><i>El</i> | Element to move |
|----------------------------------|-----------------|

**16.107.2.7 Remove()**

```
void IPV6_ROOT_EL_CONTAINER::Remove (
 IPV6_ROOT_EL * pEl)
```

Remove element from container.

## Parameters

|                                  |                   |
|----------------------------------|-------------------|
| $p \leftrightarrow$<br><i>El</i> | Element to remove |
|----------------------------------|-------------------|

The documentation for this struct was generated from the following file:

- `ipv6_interface.h`

## 16.108 IPV6\_ROUTER Struct Reference

IPv6 router information element.

```
#include <ipv6_interface.h>
```

Inherits [IPV6\\_ROOT\\_EL](#).

### Public Member Functions

- [IPV6\\_ROUTER \\* GetNextValid](#) ()  
*Get next valid router in list.*
- [IPV6\\_ROUTER \\* GetNext](#) ()  
*Get next router in list.*
- void **CleanUpAndRemove** ()  
*Clean up and remove router from lists.*
- void **RemoveFromDefault** ()  
*Remove router from default router list.*
- void **free\_element** ()  
*Free resources for this router.*
- virtual bool [AgeStillValidTest](#) ()  
*Check if router is still valid.*
- virtual void **ShowItem** ()  
*Display router information.*

### Public Member Functions inherited from [IPV6\\_ROOT\\_EL](#)

- virtual  $\sim$ [IPV6\\_ROOT\\_EL](#) ()  
*Virtual destructor.*

### Public Attributes

- `uint16_t m_CheckSumCache`  
*Cached checksum value.*
- `uint32_t m_SecsLastAdvertise`  
*Seconds since last router advertisement.*
- [IPV6\\_NEIGHBOR](#) \* `m_pNeighbor`  
*Associated neighbor entry.*
- `uint32_t m_Life_In_Secs`  
*Router lifetime in seconds.*
- `uint16_t m_PathMTU`  
*Path MTU for this router.*



## Public Attributes inherited from [IPV6\\_ROOT\\_EL](#)

- [IPV6\\_ROOT\\_EL](#) \* **m\_pNext**  
*Next element in list.*
- [IPV6\\_ROOT\\_EL](#) \* **m\_pPrev**  
*Previous element in list.*
- [IPV6\\_ROOT\\_EL\\_CONTAINER](#) \* **m\_pContainer**  
*Container that owns this element.*
- [IPADDR6](#) **m\_IPAddress**  
*IPv6 address associated with this element.*

### 16.108.1 Detailed Description

IPv6 router information element.

Stores information about IPv6 routers including advertisement timing and path MTU.

### 16.108.2 Member Function Documentation

#### 16.108.2.1 AgeStillValidTest()

```
virtual bool IPV6_ROUTER::AgeStillValidTest () [virtual]
```

Check if router is still valid.

##### Returns

true if router is still valid

Reimplemented from [IPV6\\_ROOT\\_EL](#).

#### 16.108.2.2 GetNext()

```
IPV6_ROUTER * IPV6_ROUTER::GetNext ()
```

Get next router in list.

##### Returns

Pointer to next router or NULL

#### 16.108.2.3 GetNextValid()

```
IPV6_ROUTER * IPV6_ROUTER::GetNextValid ()
```

Get next valid router in list.

##### Returns

Pointer to next valid router or NULL

The documentation for this struct was generated from the following file:

- [ipv6\\_interface.h](#)

## 16.109 IPv6FrameProcessingStruct Struct Reference

IPv6 frame processing structure for packet analysis and header walking.

```
#include <ipv6_interface.h>
```

## Public Member Functions

- void **free\_proc\_struct** ()  
*Free the processing structure and associated resources.*
- void **release** ()  
*Release buffer pool resources.*
- uint8\_t **WalkNextHeader** (int \*pError\_offset=NULL)  
*Walk through IPv6 extension headers to the next header.*
- bool **DestOptHasError** (puint8\_t pData, int \*pError\_offset)  
*Check destination options header for errors.*
- bool **HopByHopParseHasError** (puint8\_t pData, int \*pError\_offset)  
*Check hop-by-hop options header for errors.*
- bool **RouterHeaderParseHasError** (puint8\_t pData, int \*pError\_offset)  
*Check routing header for errors.*
- **IPv6FrameProcessingStruct** (PoolPtr pp)  
*Constructor with buffer pool.*
- **IPv6FrameProcessingStruct** ()  
*Default constructor.*
- puint8\_t **GetDataPointer** ()  
*Get pointer to frame data.*
- EFRAME \* **GetEframe** ()  
*Get pointer to Ethernet frame.*

## Public Attributes

- **PoolPtr** root\_pp  
*Root buffer pool pointer.*
- IP6FRAME \* **pIPf**  
*Pointer to IPv6 frame header.*
- uint8\_t \* **pData**  
*Pointer to current data location.*
- uint16\_t **nDataLen**  
*Length of current data segment.*
- uint8\_t \* **pLargeData**  
*Used for fragments larger than 1500 bytes.*

## 16.109.1 Detailed Description

IPv6 frame processing structure for packet analysis and header walking.

This structure contains all the necessary components for processing IPv6 packets, including buffer management and header parsing capabilities.

## 16.109.2 Constructor & Destructor Documentation

### 16.109.2.1 IPv6FrameProcessingStruct()

```
IPv6FrameProcessingStruct::IPv6FrameProcessingStruct (
 PoolPtr pp)
```

Constructor with buffer pool.

#### Parameters

|           |                     |
|-----------|---------------------|
| <i>pp</i> | Buffer pool pointer |
|-----------|---------------------|

### 16.109.3 Member Function Documentation

#### 16.109.3.1 DestOptHasError()

```
bool IPv6FrameProcessingStruct::DestOptHasError (
 uint8_t pData,
 int * pError_offset)
```

Check destination options header for errors.

##### Parameters

|                      |                                                 |
|----------------------|-------------------------------------------------|
| <i>pData</i>         | Pointer to header data                          |
| <i>pError_offset</i> | Optional pointer to store error offset location |

##### Returns

true if errors found, false otherwise

#### 16.109.3.2 GetDataPointer()

```
uint8_t IPv6FrameProcessingStruct::GetDataPointer () [inline]
```

Get pointer to frame data.

##### Returns

Pointer to frame data or NULL if not available

#### 16.109.3.3 GetEframe()

```
EFRAME * IPv6FrameProcessingStruct::GetEframe () [inline]
```

Get pointer to Ethernet frame.

##### Returns

Pointer to EFRAME or NULL if not available

#### 16.109.3.4 HopByHopParseHasError()

```
bool IPv6FrameProcessingStruct::HopByHopParseHasError (
 uint8_t pData,
 int * pError_offset)
```

Check hop-by-hop options header for errors.

##### Parameters

|                      |                                                 |
|----------------------|-------------------------------------------------|
| <i>pData</i>         | Pointer to header data                          |
| <i>pError_offset</i> | Optional pointer to store error offset location |

##### Returns

true if errors found, false otherwise

#### 16.109.3.5 RouterHeaderParseHasError()

```
bool IPv6FrameProcessingStruct::RouterHeaderParseHasError (
 uint8_t pData,
 int * pError_offset)
```

Check routing header for errors.

## Parameters

|                      |                                                 |
|----------------------|-------------------------------------------------|
| <i>pData</i>         | Pointer to header data                          |
| <i>pError_offset</i> | Optional pointer to store error offset location |

## Returns

true if errors found, false otherwise

**16.109.3.6 WalkNextHeader()**

```
uint8_t IPv6FrameProcessingStruct::WalkNextHeader (
 int * pError_offset = NULL)
```

Walk through IPv6 extension headers to the next header.

## Parameters

|                      |                                                 |
|----------------------|-------------------------------------------------|
| <i>pError_offset</i> | Optional pointer to store error offset location |
|----------------------|-------------------------------------------------|

## Returns

Next header type value

The documentation for this struct was generated from the following file:

- `ipv6_interface.h`

**16.110 IPv6Interface Class Reference**

Main IPv6 interface management class.

```
#include <ipv6_interface.h>
```

Inherits `TimeoutElement`.

**Public Member Functions**

- `uint16_t GetDestinationMTU` (const `IPADDR` &ip)  
*Get destination MTU for specified address.*
- `IPv6Interface * GetNext_IP6_Interface` ()  
*Get next IPv6 interface in the list.*
- `IPADDR6 MySourceAddress` (const `IPADDR6` &ip)  
*Determine best source address for communicating with destination.*
- `int GetInterfaceNumber` ()  
*Get interface number.*
- `IPADDR6 MyDNSAddress` ()  
*Get DNS server address for this interface.*
- `bool HadLink` ()  
*Check if interface has had link connectivity.*
- `IPV6_PREFIX * AddStaticAddress` (const `IPADDR6` &ip, int PrefixLen)  
*Add static IPv6 address to interface.*
- `IPV6_ROUTER * AddDefaultGateway` (const `IPADDR6` &ip)  
*Add default gateway to interface.*
- `IPV6_DNS * AddStaticDNS` (const `IPADDR6` &ip)  
*Add static DNS server to interface.*
- `bool RemoveStaticAddress` (const `IPADDR6` &ip)

- Remove static IPv6 address from interface.*

  - bool [RemoveDefaultGateway](#) (const [IPADDR6](#) &ip)
- Remove default gateway from interface.*

  - bool [RemoveStaticDNS](#) (const [IPADDR6](#) &ip)
- Remove static DNS server from interface.*

  - void [SetStaticDNS](#) ([IPADDR6](#) dns)
- Set static DNS server address.*

  - void [StartDHCP\\_Solicit](#) ()
- Start DHCPv6 solicitation process.*

  - void [StartDHCP\\_InfoReq](#) ()
- Start DHCPv6 information request process.*

  - bool [IsMyAddress](#) (const [IPADDR6](#) &ip6, bool bMustBePreferred)
- Check if IPv6 address belongs to this interface.*

  - bool [OnLink](#) (const [IPADDR6](#) &ip6)
- Check if IPv6 address is on-link for this interface.*

  - bool [HasRoute](#) (const [IPADDR6](#) &ip6)
- Check if there is a route to the specified IPv6 address.*

  - void [NotifyReachable](#) (const [IPADDR6](#) &ip6)
- Notify that neighbor is reachable.*

  - void [NotifyUnreachable](#) (const [IPADDR6](#) &ip)
- Notify that neighbor is unreachable.*

  - [IPADDR6](#) [GetMyFirstAddress](#) ()
- Get first configured IPv6 address.*

  - [IPADDR6](#) [GetMyNextAddress](#) (const [IPADDR6](#) &ip)
- Get next IPv6 address after specified address.*

  - [IPv6Interface](#) (int ifnum)
- Constructor for IPv6 interface.*

  - int [ping](#) (const [IPADDR6](#) &ip, uint16\_t id, uint16\_t seq, uint16\_t wait, int siz)
- Send IPv6 ping to specified address.*

  - int [SendMLDRegistration](#) (const [IPADDR](#) &regAddr, bool joinNotLeave=true)
- Send MLD registration message.*

  - void [ShowInfo](#) ()
- Display IPv6 information for this interface.*

### Static Public Member Functions

- static [IPv6Interface](#) \* [GetInterfaceForDestination](#) (const [IPADDR6](#) &ip)

*Find interface that can reach the specified destination.*
- static [IPv6Interface](#) \* [GetInterfaceForSource](#) (const [IPADDR6](#) &ip)

*Find interface that should be used as source for specified destination.*
- static [IPv6Interface](#) \* [GetInterfaceN](#) (int n)

*Get IPv6 interface by interface number.*
- static [IPv6Interface](#) \* [GetFirst\\_IP6\\_Interface](#) ()

*Get first IPv6 interface in the system.*
- static void [ShowAllInfo](#) ()

*Display IPv6 information for all interfaces.*

### Public Attributes

- [IPV6\\_PREFIX](#) \* [m\\_pMyLinkLocal](#)
- Link-local address for this interface.*
- [IPADDR6](#) [m\\_StaticDNS](#)
- Static DNS server address.*

**Friends**

- struct **IPV6\_PREFIX**
- struct **IPV6\_DEST**
- struct **IPV6\_ROUTER**
- struct **IPV6\_NEIGHBOR**
- struct **IPV6\_DNS**
- struct **IPV6\_DHCPD**
- class **UDPPacket**
- void [InitIPv6](#) (int ifnum=0)
- int [Ping6](#) (const [IPADDR6](#) &to, uint16\_t id, uint16\_t seq, uint16\_t maxwaitticks, int size=32)

*Send an IPv6 "ping" packet and wait the specified timeout for a response.*

**16.110.1 Detailed Description**

Main IPv6 interface management class.

Provides comprehensive IPv6 interface functionality including address management, routing, neighbor discovery, and protocol processing.

**16.110.2 Constructor & Destructor Documentation****16.110.2.1 IPv6Interface()**

```
IPv6Interface::IPv6Interface (
 int ifnum)
```

Constructor for IPv6 interface.

**Parameters**

|              |                  |
|--------------|------------------|
| <i>ifnum</i> | Interface number |
|--------------|------------------|

**16.110.3 Member Function Documentation****16.110.3.1 AddDefaultGateway()**

```
IPV6_ROUTER * IPv6Interface::AddDefaultGateway (
 const IPADDR6 & ip)
```

Add default gateway to interface.

**Parameters**

|           |                                 |
|-----------|---------------------------------|
| <i>ip</i> | IPv6 address of default gateway |
|-----------|---------------------------------|

**Returns**

Pointer to created router entry or NULL on failure

**See also**

[RemoveDefaultGateway\(\)](#), [AddStaticAddress\(\)](#), [AddStaticDNS\(\)](#)

**16.110.3.2 AddStaticAddress()**

```
IPV6_PREFIX * IPv6Interface::AddStaticAddress (
 const IPADDR6 & ip,
 int PrefixLen)
```

Add static IPv6 address to interface.

**Parameters**

|                  |                       |
|------------------|-----------------------|
| <i>ip</i>        | IPv6 address to add   |
| <i>PrefixLen</i> | Prefix length in bits |

**Returns**

Pointer to created prefix entry or NULL on failure

**See also**

[RemoveStaticAddress\(\)](#), [AddDefaultGateway\(\)](#), [AddStaticDNS\(\)](#)

**16.110.3.3 AddStaticDNS()**

```
IPV6_DNS * IPv6Interface::AddStaticDNS (
 const IPADDR6 & ip)
```

Add static DNS server to interface.

**Parameters**

|           |                            |
|-----------|----------------------------|
| <i>ip</i> | IPv6 address of DNS server |
|-----------|----------------------------|

**Returns**

Pointer to created DNS entry or NULL on failure

**See also**

[RemoveStaticDNS\(\)](#), [AddStaticAddress\(\)](#), [AddDefaultGateway\(\)](#)

**16.110.3.4 GetDestinationMTU()**

```
uint16_t IPv6Interface::GetDestinationMTU (
 const IPADDR & ip)
```

Get destination MTU for specified address.

**Parameters**

|           |                        |
|-----------|------------------------|
| <i>ip</i> | Destination IP address |
|-----------|------------------------|

**Returns**

MTU value for the destination

**16.110.3.5 GetFirst\_IP6\_Interface()**

```
static IPv6Interface * IPv6Interface::GetFirst_IP6_Interface () [static]
```

Get first IPv6 interface in the system.

**Returns**

Pointer to first interface or NULL if none exist

**See also**

[GetInterfaceN\(\)](#), [GetNext\\_IP6\\_Interface\(\)](#)

**16.110.3.6 GetInterfaceForDestination()**

```
static IPv6Interface * IPv6Interface::GetInterfaceForDestination (
 const IPADDR6 & ip) [static]
```

Find interface that can reach the specified destination.

**Parameters**

|           |                          |
|-----------|--------------------------|
| <i>ip</i> | Destination IPv6 address |
|-----------|--------------------------|

**Returns**

Pointer to appropriate interface or NULL if none found

**See also**

[GetInterfaceForSource\(\)](#), [GetInterfaceN\(\)](#)

**16.110.3.7 GetInterfaceForSource()**

```
static IPv6Interface * IPv6Interface::GetInterfaceForSource (
 const IPADDR6 & ip) [static]
```

Find interface that should be used as source for specified destination.

**Parameters**

|           |                          |
|-----------|--------------------------|
| <i>ip</i> | Destination IPv6 address |
|-----------|--------------------------|

**Returns**

Pointer to appropriate interface or NULL if none found

**See also**

[GetInterfaceForDestination\(\)](#), [GetInterfaceN\(\)](#)

**16.110.3.8 GetInterfaceN()**

```
static IPv6Interface * IPv6Interface::GetInterfaceN (
 int n) [static]
```

Get IPv6 interface by interface number.

**Parameters**

|          |                  |
|----------|------------------|
| <i>n</i> | Interface number |
|----------|------------------|

**Returns**

Pointer to interface or NULL if not found

**See also**

[GetFirst\\_IP6\\_Interface\(\)](#), [GetNext\\_IP6\\_Interface\(\)](#)

**16.110.3.9 GetInterfaceNumber()**

```
int IPv6Interface::GetInterfaceNumber () [inline]
```

Get interface number.

**Returns**

Interface number for this interface



**16.110.3.10 GetMyFirstAddress()**

```
IPADDR6 IPv6Interface::GetMyFirstAddress () [inline]
```

Get first configured IPv6 address.

**Returns**

First IPv6 address or null address if none configured

**16.110.3.11 GetMyNextAddress()**

```
IPADDR6 IPv6Interface::GetMyNextAddress (
 const IPADDR6 & ip) [inline]
```

Get next IPv6 address after specified address.

**Parameters**

|           |                      |
|-----------|----------------------|
| <i>ip</i> | Current IPv6 address |
|-----------|----------------------|

**Returns**

Next IPv6 address or null address if at end

**16.110.3.12 GetNext\_IP6\_Interface()**

```
IPv6Interface * IPv6Interface::GetNext_IP6_Interface ()
```

Get next IPv6 interface in the list.

**Returns**

Pointer to next interface or NULL if at end of list

**See also**

[GetFirst\\_IP6\\_Interface\(\)](#), [GetInterfaceN\(\)](#)

**16.110.3.13 HadLink()**

```
bool IPv6Interface::HadLink () [inline]
```

Check if interface has had link connectivity.

**Returns**

true if interface has had link

**16.110.3.14 HasRoute()**

```
bool IPv6Interface::HasRoute (
 const IPADDR6 & ip6)
```

Check if there is a route to the specified IPv6 address.

**Parameters**

|            |                          |
|------------|--------------------------|
| <i>ip6</i> | Destination IPv6 address |
|------------|--------------------------|

**Returns**

true if route exists

**See also**

[IsMyAddress\(\)](#), [OnLink\(\)](#)

### 16.110.3.15 IsMyAddress()

```
bool IPv6Interface::IsMyAddress (
 const IPADDR6 & ip6,
 bool bMustBePreferred)
```

Check if IPv6 address belongs to this interface.

#### Parameters

|                         |                                            |
|-------------------------|--------------------------------------------|
| <i>ip6</i>              | IPv6 address to check                      |
| <i>bMustBePreferred</i> | true if address must be in preferred state |

#### Returns

true if address belongs to this interface

#### See also

[OnLink\(\)](#), [HasRoute\(\)](#)

### 16.110.3.16 MyDNSAddress()

```
IPADDR6 IPv6Interface::MyDNSAddress ()
```

Get DNS server address for this interface.

#### Returns

IPv6 address of DNS server

### 16.110.3.17 MySourceAddress()

```
IPADDR6 IPv6Interface::MySourceAddress (
 const IPADDR6 & ip)
```

Determine best source address for communicating with destination.

#### Parameters

|           |                          |
|-----------|--------------------------|
| <i>ip</i> | Destination IPv6 address |
|-----------|--------------------------|

#### Returns

Best source address for this interface

### 16.110.3.18 NotifyReachable()

```
void IPv6Interface::NotifyReachable (
 const IPADDR6 & ip6)
```

Notify that neighbor is reachable.

#### Parameters

|            |                                    |
|------------|------------------------------------|
| <i>ip6</i> | IPv6 address of reachable neighbor |
|------------|------------------------------------|

### 16.110.3.19 NotifyUnreachable()

```
void IPv6Interface::NotifyUnreachable (
 const IPADDR6 & ip)
```

Notify that neighbor is unreachable.

## Parameters

|           |                                      |
|-----------|--------------------------------------|
| <i>ip</i> | IPv6 address of unreachable neighbor |
|-----------|--------------------------------------|

**16.110.3.20 OnLink()**

```
bool IPv6Interface::OnLink (
 const IPADDR6 & ip6)
```

Check if IPv6 address is on-link for this interface.

## Parameters

|            |                       |
|------------|-----------------------|
| <i>ip6</i> | IPv6 address to check |
|------------|-----------------------|

## Returns

true if address is on-link

## See also

[IsMyAddress\(\)](#), [HasRoute\(\)](#)

**16.110.3.21 ping()**

```
int IPv6Interface::ping (
 const IPADDR6 & ip,
 uint16_t id,
 uint16_t seq,
 uint16_t wait,
 int siz)
```

Send IPv6 ping to specified address.

## Parameters

|             |                            |
|-------------|----------------------------|
| <i>ip</i>   | Destination IPv6 address   |
| <i>id</i>   | Ping identifier            |
| <i>seq</i>  | Ping sequence number       |
| <i>wait</i> | Maximum wait time in ticks |
| <i>siz</i>  | Ping packet size           |

## Returns

Ping result status

**16.110.3.22 RemoveDefaultGateway()**

```
bool IPv6Interface::RemoveDefaultGateway (
 const IPADDR6 & ip)
```

Remove default gateway from interface.

## Parameters

|           |                                   |
|-----------|-----------------------------------|
| <i>ip</i> | IPv6 address of gateway to remove |
|-----------|-----------------------------------|

**Returns**

true if gateway was found and removed

**See also**

[AddDefaultGateway\(\)](#), [RemoveStaticAddress\(\)](#), [RemoveStaticDNS\(\)](#)

**16.110.3.23 RemoveStaticAddress()**

```
bool IPv6Interface::RemoveStaticAddress (
 const IPADDR6 & ip)
```

Remove static IPv6 address from interface.

**Parameters**

|           |                        |
|-----------|------------------------|
| <i>ip</i> | IPv6 address to remove |
|-----------|------------------------|

**Returns**

true if address was found and removed

**See also**

[AddStaticAddress\(\)](#), [RemoveDefaultGateway\(\)](#), [RemoveStaticDNS\(\)](#)

**16.110.3.24 RemoveStaticDNS()**

```
bool IPv6Interface::RemoveStaticDNS (
 const IPADDR6 & ip)
```

Remove static DNS server from interface.

**Parameters**

|           |                                      |
|-----------|--------------------------------------|
| <i>ip</i> | IPv6 address of DNS server to remove |
|-----------|--------------------------------------|

**Returns**

true if DNS server was found and removed

**See also**

[AddStaticDNS\(\)](#), [RemoveStaticAddress\(\)](#), [RemoveDefaultGateway\(\)](#)

**16.110.3.25 SendMLDRegistration()**

```
int IPv6Interface::SendMLDRegistration (
 const IPADDR & regAddr,
 bool joinNotLeave = true)
```

Send MLD registration message.

**Parameters**

|                     |                               |
|---------------------|-------------------------------|
| <i>regAddr</i>      | Multicast address to register |
| <i>joinNotLeave</i> | true to join, false to leave  |

**Returns**

Result status

**16.110.3.26 SetStaticDNS()**

```
void IPv6Interface::SetStaticDNS (
 IPADDR6 dns) [inline]
```

Set static DNS server address.

**Parameters**

|            |                            |
|------------|----------------------------|
| <i>dns</i> | IPv6 address of DNS server |
|------------|----------------------------|

**16.110.3.27 ShowAllInfo()**

```
static void IPv6Interface::ShowAllInfo () [static]
```

Display IPv6 information for all interfaces.

**See also**

[ShowInfo\(\)](#)

**16.110.3.28 ShowInfo()**

```
void IPv6Interface::ShowInfo ()
```

Display IPv6 information for this interface.

**See also**

[ShowAllInfo\(\)](#)

**16.110.4 Friends And Related Symbol Documentation****16.110.4.1 InitIPv6**

```
void InitIPv6 (
 int ifnum = 0) [friend]
```

**16.110.4.2 Ping6**

```
int Ping6 (
 const IPADDR6 & to,
 uint16_t id,
 uint16_t seq,
 uint16_t maxwaitticks,
 int size = 32) [friend]
```

Send an IPv6 "ping" packet and wait the specified timeout for a response.

Sends an ICMPv6 echo request packet to the specified IPv6 destination address and waits for an echo reply. The ping uses the primary network interface. This follows the ICMPv6 echo request/reply protocol defined in RFC 4443.

**Parameters**

|                     |                                                            |
|---------------------|------------------------------------------------------------|
| <i>to</i>           | Destination IPv6 address to ping                           |
| <i>id</i>           | ICMP identifier field (typically process ID)               |
| <i>seq</i>          | ICMP sequence number field (typically increments per ping) |
| <i>maxwaitticks</i> | Maximum wait time in system time ticks                     |
| <i>size</i>         | Size of ping data payload in bytes (default: 32)           |

**Returns**

>=0 Number of ticks elapsed waiting for the response (successful ping) -1 Timeout occurred - no response received within maxwaitticks -2 Error other than timeout occurred

**Note**

IPv6 ping packets may traverse different paths than IPv4 pings to the same host.  
Link-local addresses require the appropriate zone ID/interface specification.

**See also**

[Ping6VialInterface\(\)](#), [Ping4\(\)](#), [Ping\(\)](#)

The documentation for this class was generated from the following file:

- [ipv6\\_interface.h](#)

## 16.111 JsonAllocString Struct Reference

A list of large strings that are created with malloc.

```
#include <json_lexer.h>
```

**Public Attributes**

- [JsonAllocString](#) \* **pNext**  
*A pointer to the next string.*
- char **data** []  
*The current string data.*

### 16.111.1 Detailed Description

A list of large strings that are created with malloc.

The documentation for this struct was generated from the following file:

- [json\\_lexer.h](#)

## 16.112 JsonLexerFDAdapter Class Reference

Get the type name of the JSON primitive type passed in.

```
#include <json_lexer.h>
```

Inherits [ParsedJsonDataSet](#), and [fd\\_adapter](#).

**Additional Inherited Members****Public Member Functions inherited from [ParsedJsonDataSet](#)**

- void **EnableLargeStrings** (bool b)  
*Call to allow allocation of strings larger than 1500 bytes.*
- virtual int [WriteData](#) (const unsigned char \*pCopyFrom, int numBytes)  
*Writes the passed in data to the JSON data set.*
- virtual int [ReadFrom](#) (int fd)  
*Reads in data from the specified file descriptor and parses it into the JSON data set.*
- bool [CopyObject](#) ([ParsedJsonDataSet](#) &src\_set)  
*Copies the provided JSON data set into the current one.*
- void **ClearObject** ()  
*Clears all data from the object and resets it to its initial state.*
- [JsonRef](#) [operator\[\]](#) (int i)  
*Get a [JsonRef](#) from an array by numerical index.*
- [JsonRef](#) [operator\(\)](#) (const char \*name)  
*Get a [JsonRef](#) representing any entity from a parent object by key name.*

- [JsonRef name](#) (const char \*name)  
*Get a [JsonRef](#) representing any entity from a parent object by key name.*
- [JsonRef object](#) (const char \*name)  
*Get a [JsonRef](#) representing an object from a parent object by key name.*
- [JsonRef start](#) ()  
*Begin traversal: check for ref validity and reset the position.*
- [JsonRef next](#) ()  
*Traverse: check for ref validity and increment the position.*
- [json\\_primitive\\_type GetFirst](#) ()  
*Get the first element of the JSON data set.*
- [json\\_primitive\\_type GetNext](#) ()  
*Get the element at the next position of the JSON data set.*
- [json\\_primitive\\_type GetCurrent](#) ()  
*Get the element at the current position of the JSON data set.*
- [json\\_primitive\\_type GetRawCurrent](#) ()  
*Get the element at the current position of the JSON data set, including non-public types.*
- [json\\_primitive\\_type GetNextName](#) ()  
*Iterates the current element of the JSON data set until it finds one of type NAME.*
- [json\\_primitive\\_type GetNextObject](#) ()  
*Get the next element of type name that exists in the current OBJECT after the current position.*
- [json\\_primitive\\_type GetNextArray](#) ()  
*Iterates the current element of the JSON data set until it finds one of type BEGIN\_ARRAY.*
- [json\\_primitive\\_type GetNextNameInCurrentObject](#) ()  
*Get the next element of type NAME that exists in the current OBJECT after the current position.*
- [json\\_primitive\\_type GetNextNameInCurrentArray](#) ()  
*Get the next element of type NAME that exists in the current ARRAY after the current position.*
- [json\\_primitive\\_type GetNextNumberInCurrentArray](#) ()  
*Get the next element of type NUMBER that exists in the current ARRAY after the current position.*
- [json\\_primitive\\_type GetNextStringInCurrentArray](#) ()  
*Get the next element of type STRING that exists in the current ARRAY after the current position.*
- [json\\_primitive\\_type GetNextBoolInCurrentArray](#) ()  
*Get the next element of type TRUE\_EL that exists in the current ARRAY after the current position.*
- [json\\_primitive\\_type GetNextObjectInCurrentArray](#) ()  
*Get the next element of type BEGIN\_OBJECT that exists in the current ARRAY after the current position.*
- [json\\_primitive\\_type SkipCurrentValue](#) ()  
*Skips over the current value, and get the next element. If called inside an ARRAY or OBJECT, it will walk to the end of the ARRAY or OBJECT and return the next element it finds. If it reaches the end of the ARRAY or OBJECT before it finds a value, it will return NOTFOUND.*
- void [ResetPosition](#) ()  
*Resets the parser position to the beginning of the JSON data set.*
- [JsonRef GetParsePosition](#) ()  
*Gets the current parse position object for the JSON data set.*
- [JsonRef SetParsePosition](#) (JsonRef pos)  
*Sets the current parse position object for the JSON data set.*
- [json\\_primitive\\_type FindFullName](#) (const char \*name)  
*Find the element in the data set with the given name and move the parser to the next element. This searches a full name path, such as ob1.ob2.ob3.ob4.*
- [json\\_primitive\\_type FindFullAtName](#) (const char \*name)  
*Find the element in the data set with the given name and move the parser to that element. This searches a full name path, such as ob1.ob2.ob3.ob4.*

- [json\\_primitive\\_type FindElementAfterName](#) (const char \*name)  
*Finds name in current object points at element after name. This only supports simple, single element names. This searches from the current position to the end.*
- [json\\_primitive\\_type FindGlobalElementAfterName](#) (const char \*name)  
*Finds name in current object points at element after name. This only supports simple, single element names. It starts the search from the parser's current position, and if it doesn't find it, it will start the search over from the beginning.*
- [json\\_primitive\\_type FindElementAfterNameInCurrentObject](#) (const char \*name)  
*Looks for elements with the current name in the current OBJECT only. Does not search sub objects.*
- [json\\_primitive\\_type FindElementAfterNameInCurrentArray](#) (const char \*name)  
*Looks for elements with the current name in the current ARRAY only. Does not search sub arrays.*
- bool [CurrentBool](#) ()  
*Returns true if the current element a TRUE\_EL primitive type.*
- bool [PermissiveCurrentBool](#) ()  
*Returns true if the current element a TRUE\_EL primitive type, is "True", "true", or is a non-zero number.*
- double [CurrentNumber](#) ()  
*Get the number value of the current element.*
- const char \* [CurrentString](#) ()  
*Get the string value of the current element.*
- int [CopyCurrentString](#) (char \*buf, int buflen)  
*Copy the string value of the current element.*
- int [CopyCurrentString](#) (NBString &nbs)  
*Copy the string value of the current element.*
- const char \* [CurrentName](#) ()  
*Get the name of the current element.*
- bool [FindFullNameBoolean](#) (const char \*name)  
*Find the boolean value of a given element.*
- bool [FindGlobalBoolean](#) (const char \*name)  
*Find the boolean value of a given element. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.*
- bool [FindBoolean](#) (const char \*name)  
*Find the boolean value of the element after the element with the given name.*
- bool [FindBooleanInCurentObject](#) (const char \*name)  
*Find the boolean value of the element within the current object. Does not search sub-objects.*
- bool [FindFullNamePermissiveBoolean](#) (const char \*name)  
*Find the permissive boolean value of a given element.*
- bool [FindGlobalPermissiveBoolean](#) (const char \*name)  
*Find the permissive boolean value of a given element. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.*
- bool [FindPermissiveBoolean](#) (const char \*name)  
*Find the permissive boolean value of the element after the element with the provided name.*
- bool [FindPermissiveBooleanInCurentObject](#) (const char \*name)  
*Find the permissive boolean value of the element with the given name in the current object. Does not search sub-objects.*
- const char \* [FindFullNameString](#) (const char \*name)  
*Find the string value of the element with the given name in the current object.*
- int [FindAndCopyFullNameString](#) (const char \*name, char \*buf, int buflen)  
*Find and copy the string value of the element with the given name in the current object. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.*
- int [FindAndCopyFullNameString](#) (const char \*name, NBString &nbs)  
*Find and copy the string value of the element with the given name in the current object. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.*
- const char \* [FindGlobalString](#) (const char \*name)



- Find the string value of a given element. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.*
- int [FindAndCopyGlobalString](#) (const char \*name, char \*buf, int buflen)  
*Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names. It then copies the found string to the provided buffer. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.*
  - int [FindAndCopyGlobalString](#) (const char \*name, [NBString](#) &nbs)  
*Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names. It then copies the found string to the provided buffer. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.*
  - const char \* [FindString](#) (const char \*name)  
*Find the string value of the element after the element with the given name.*
  - int [FindAndCopyString](#) (const char \*name, char \*buf, int buflen)  
*Find and copy the string value of the element with the given name. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.*
  - int [FindAndCopyString](#) (const char \*name, [NBString](#) &nbs)  
*Find and copy the string value of the element with the given name. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.*
  - const char \* [FindStringInCurentObject](#) (const char \*name)  
*Find the string value of the element with the given name in the current object. Does not search sub-objects.*
  - double [FindFullNameNumber](#) (const char \*name)  
*Find the number value of the element with the given name in the current object.*
  - double [FindGlobalNumber](#) (const char \*name)  
*Find the number value of a given element. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.*
  - double [FindNumber](#) (const char \*name)  
*Find the number value of the element after the element with the given name.*
  - double [FindNumberInCurentObject](#) (const char \*name)  
*Find the number value of the element with the given name in the current object. Does not search sub-objects.*
  - bool [FindGlobalObject](#) (const char \*name)  
*Find the OBJECT with the given name. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.*
  - bool [FindObject](#) (const char \*name)  
*Find the OBJECT of the element after the element with the given name.*
  - bool [FindObjectInCurentObject](#) (const char \*name)  
*Find the OBJECT of the element with the given name in the current OBJECT. Does not search sub-objects.*
  - void **DumpState** ()  
*Outputs what's in the parse tree to stdout.*
  - int [PrintObject](#) (bool pretty=false)  
*Prints the JSON data set to stdout.*
  - int [PrintObjectToBuffer](#) (char \*buffer, int maxlen, bool pretty=false)  
*Prints the JSON data set to a provided buffer.*
  - int [PrintObjectToFd](#) (int fd, bool pretty=false)  
*Prints the JSON data set to a specified file descriptor.*
  - int [PrintObjectToString](#) ([NBString](#) &s, bool pretty=false)  
*Prints the JSON data set to a [NBString](#) object.*
  - int **PrintChildren** (bool pretty=false)  
*Output child objects of this [JsonRef](#) as JSON text to stdout.*
  - int **PrintChildrenToFd** (int fd, bool pretty=false)  
*Output child objects of this [JsonRef](#) as JSON text to a file descriptor.*
  - int **PrintChildrenToBuffer** (char \*buffer, int maxlen, bool pretty=false)  
*Output child objects of this [JsonRef](#) as JSON text to a buffer pointer.*

- `int GetPrintSize (bool pretty=false)`  
*Calculates how many characters the JSON data set would take to print.*
- `ParsedJsonDataSet * StartBuilding ()`  
*Use to start building the JSON data set.*
- `ParsedJsonDataSet * AddObjectStart (const char *name)`  
*Use to start an object in the JSON data set.*
- `ParsedJsonDataSet * AddMyMac (const char *name)`  
*Add the device MAC address to the JSON data set.*
- `ParsedJsonDataSet * Add (const char *name, int i)`  
*Add a name/value pair to the JSON data set where the value is an int.*
- `ParsedJsonDataSet * Add (const char *name, short i)`  
*Add a name/value pair to the JSON data set where the value is a short.*
- `ParsedJsonDataSet * Add (const char *name, long i)`  
*Add a name/value pair to the JSON data set where the value is a long.*
- `ParsedJsonDataSet * Add (const char *name, unsigned int i)`  
*Add a name/value pair to the JSON data set where the value is an unsigned int.*
- `ParsedJsonDataSet * Add (const char *name, unsigned short i)`  
*Add a name/value pair to the JSON data set where the value is a unsigned short.*
- `ParsedJsonDataSet * Add (const char *name, unsigned long i)`  
*Add a name/value pair to the JSON data set where the value is a unsigned long.*
- `ParsedJsonDataSet * Add (const char *name, double d)`  
*Add a name/value pair to the JSON data set where the value is a double.*
- `ParsedJsonDataSet * Add (const char *name, const char *str)`  
*Add a name/value pair to the JSON data set where the value is a string.*
- `ParsedJsonDataSet * Add (const char *name, bool b)`  
*Add a name/value pair to the JSON data set where the value is a bool.*
- `ParsedJsonDataSet * Add (const char *name, IPADDR4 i4)`  
*Add a name/value pair to the JSON data set where the value is an IPv4 address.*
- `ParsedJsonDataSet * Add (const char *name, const IPADDR &i)`  
*Add a name/value pair to the JSON data set where the value is an IP address.*
- `ParsedJsonDataSet * Add (const char *name, const MACADR &ma)`  
*Add a name/value pair to the JSON data set where the value is a MAC address.*
- `ParsedJsonDataSet * AddNull (const char *name)`  
*Add a name/value pair to the JSON data set where the value is null.*
- `ParsedJsonDataSet * AddArrayStart (const char *name)`  
*Add an ARRAY start to the JSON data set.*
- `ParsedJsonDataSet * EndArray ()`  
*Add an ARRAY end to the JSON data set.*
- `ParsedJsonDataSet * AddArrayElement (int i)`  
*Add an integer value to the current array.*
- `ParsedJsonDataSet * AddArrayElement (short i)`  
*Add a short value to the current array.*
- `ParsedJsonDataSet * AddArrayElement (long i)`  
*Add a long value to the current array.*
- `ParsedJsonDataSet * AddArrayElement (unsigned int i)`  
*Add an unsigned int value to the current array.*
- `ParsedJsonDataSet * AddArrayElement (unsigned short i)`  
*Add an unsigned short value to the current array.*
- `ParsedJsonDataSet * AddArrayElement (unsigned long i)`  
*Add an unsigned long value to the current array.*

- [ParsedJsonDataSet](#) \* [AddArrayElement](#) (double d)  
*Add a double to the current array.*
- [ParsedJsonDataSet](#) \* [AddArrayElement](#) (const char \*str)  
*Add a string to the current array.*
- [ParsedJsonDataSet](#) \* [AddArrayElement](#) (bool b)  
*Add a bool to the current array.*
- [ParsedJsonDataSet](#) \* [AddArrayElement](#) (const IPADDR &i)  
*Add an IP address to the current array.*
- [ParsedJsonDataSet](#) \* [AddArrayElementArray](#) ()  
*Add the start of an array element to the current array.*
- [ParsedJsonDataSet](#) \* [AddArrayObjectStart](#) ()  
*Add the start of an object element to the current array.*
- [ParsedJsonDataSet](#) \* [AddNullArrayElement](#) ()  
*Add a null element to the current array.*
- [ParsedJsonDataSet](#) \* [EndObject](#) ()  
*Add an end to the current object.*
- [ParsedJsonDataSet](#) \* [DoneBuilding](#) ()  
*Add an end JSON data set and finish building.*

### 16.112.1 Detailed Description

Get the type name of the JSON primitive type passed in.

#### Parameters

|          |                                             |
|----------|---------------------------------------------|
| <i>t</i> | The JSON primitive type to get the name of. |
|----------|---------------------------------------------|

#### Returns

The name of the JSON primitive type passed in.

The documentation for this class was generated from the following file:

- json\_lexer.h

## 16.113 JsonRef Class Reference

Represents a positional reference (pointer) of a location inside a [ParsedJsonDataSet](#) object.

```
#include <json_lexer.h>
```

### Public Member Functions

#### Querying the Data Structure

*Allows easy retrieval of entities by array index or object key name*

#### Example

```
printf("User %d name %s",
 (int)myJsonRef("users")[0]("id"),
 (const char *)myJsonRef("users")[0]("name")
);
```

- [JsonRef operator\[\]](#) (int i)  
*Get a [JsonRef](#) from an array by numerical index.*
- [JsonRef object](#) (const char \*name)  
*Get a [JsonRef](#) representing an object from a parent object by key name.*
- [JsonRef name](#) (const char \*name)

Get a [JsonRef](#) representing any entity from a parent object by key name.

- [JsonRef operator\(\)](#) (const char \*namev)

Get a [JsonRef](#) representing any entity from a parent object by key name.

## Types and Operators

Allows casting of a [JsonRef](#) to the desired type

### Example

```
bool b = (bool)myJsonRef("user")("isEnabled");
int i = (int)myJsonRef("user")("id");
const char *str = (const char *)myJsonRef("user")("name");
```

- [operator bool](#) () const  
Operator to return a JSON element of type boolean.
- [operator float](#) () const  
Operator to return a JSON element of type float.
- [operator double](#) () const  
Operator to return a JSON element of type double.
- [operator uint8\\_t](#) () const  
Operator to return a JSON element of type uint8\_t.
- [operator int](#) () const  
Operator to return a JSON element of type int.
- [operator uint16\\_t](#) () const  
Operator to return a JSON element of type uint16\_t.
- [operator uint32\\_t](#) () const  
Operator to return a JSON element of type uint32\_t.
- [operator int8\\_t](#) () const  
Operator to return a JSON element of type int8\_t.
- [operator int16\\_t](#) () const  
Operator to return a JSON element of type int16\_t.
- [operator int32\\_t](#) () const  
Operator to return a JSON element of type int32\_t.
- [operator time\\_t](#) () const  
Operator to return a JSON element of type time\_t.
- [operator const char \\*](#) () const  
Operator to return a JSON element of type const char \*.
- [operator NBString](#) () const  
Operator to return a JSON element of type [NBString](#).

## Checking Validity

- bool [IsNumber](#) ()  
Check if the current [JsonRef](#) is a valid JSON number element.
- bool [IsObject](#) ()  
Check if the current [JsonRef](#) is a valid JSON object element.
- bool [IsString](#) ()  
Check if the current [JsonRef](#) is a valid JSON string element.
- bool [IsBool](#) ()  
Check if the current [JsonRef](#) is a valid JSON boolean element.
- bool [IsNull](#) ()  
Check if the current [JsonRef](#) is a valid JSON null element.
- bool [IsArray](#) ()  
Check if the current [JsonRef](#) is a valid JSON array element.
- bool [Valid](#) () const  
Check if the current [JsonRef](#) is a valid JSON position.

## Traversing the Data Structure

- [JsonRef start](#) ()  
Begin traversal: check for ref validity and reset the position.

- **JsonRef next ()** const  
*Traverse: check for ref validity and increment the position.*
- **bool IncPosition ()**  
*Increment the position index of the [ParsedJsonDataSet](#).*
- **bool JumpPosition (int siz)**  
*Jump to a specific position index in the [ParsedJsonDataSet](#).*
- **void ResetPosition ()**  
*Reset the position index of the [ParsedJsonDataSet](#) to the beginning.*
- **void SkipElement ()**  
*Increment the position index of the [ParsedJsonDataSet](#) beyond this element.*
- **void SkipArray ()**  
*Increment the position index of the [ParsedJsonDataSet](#) beyond this array.*
- **void SkipObject ()**  
*Increment the position index of the [ParsedJsonDataSet](#) beyond this object.*

### Utility

- **int PrintChildren (bool pretty=false)**  
*Output child objects of this [JsonRef](#) as JSON text to stdout.*
- **int PrintChildrenToFd (int fd, bool pretty=false)**  
*Output child objects of this [JsonRef](#) as JSON text to a file descriptor.*
- **int PrintChildrenToBuffer (char \*buffer, int maxlen, bool pretty=false)**  
*Output child objects of this [JsonRef](#) as JSON text to a buffer pointer.*
- **int PrintChildrenToString (NBString &s, bool pretty=false)**  
*Output child objects of this [JsonRef](#) as JSON text to a [NBString](#).*
- **int GetChildPrintSize (bool pretty=false)**  
*Return the length of the child objects' JSON text as written with the [PrintChildren\\*](#) functions.*
- **void DiagDump (const char \*lab)**  
*Output a verbose diagnostic report of the [ParsedJsonDatSet](#) to stdout.*
- **void MakeInvalid ()**

### Friends

- class **ParsedJsonDataSet**

## 16.113.1 Detailed Description

Represents a positional reference (pointer) of a location inside a [ParsedJsonDataSet](#) object.

A [JsonRef](#) can be used as a pointer to a variable or sub object inside a [ParsedJsonDataSet](#) object to make decoding the hierarchy much simpler and easier to understand. A [JsonRef](#) is not limited to just pointing at internal objects; it can point to any JSON parsed token/type.

## 16.113.2 Member Function Documentation

### 16.113.2.1 GetChildPrintSize()

```
int JsonRef::GetChildPrintSize (
 bool pretty = false)
```

Return the length of the child objects' JSON text as written with the [PrintChildren\\*](#) functions.

#### Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>pretty</i> | Whether indentation and new lines should be provided between elements. |
|---------------|------------------------------------------------------------------------|

#### Returns

The number of characters needed to print the JSON data set.

#### 16.113.2.2 IsArray()

```
bool JsonRef::IsArray () [inline]
```

Check if the current [JsonRef](#) is a valid JSON array element.

##### Returns

True if parsed JSON element is an array, otherwise false.

#### 16.113.2.3 IsBool()

```
bool JsonRef::IsBool () [inline]
```

Check if the current [JsonRef](#) is a valid JSON boolean element.

##### Returns

True if parsed JSON element is a boolean, otherwise false.

#### 16.113.2.4 IsNull()

```
bool JsonRef::IsNull () [inline]
```

Check if the current [JsonRef](#) is a valid JSON null element.

##### Returns

True if parsed JSON element is a null, otherwise false.

#### 16.113.2.5 IsNumber()

```
bool JsonRef::IsNumber () [inline]
```

Check if the current [JsonRef](#) is a valid JSON number element.

##### Returns

True if parsed JSON element is a number, otherwise false.

#### 16.113.2.6 IsObject()

```
bool JsonRef::IsObject () [inline]
```

Check if the current [JsonRef](#) is a valid JSON object element.

##### Returns

True if parsed JSON element is an object, otherwise false.

#### 16.113.2.7 IsString()

```
bool JsonRef::IsString () [inline]
```

Check if the current [JsonRef](#) is a valid JSON string element.

##### Returns

True if parsed JSON element is a string, otherwise false. A string can be of type `char *`, or a [NBString](#) object.

#### 16.113.2.8 MakeInvalid()

```
void JsonRef::MakeInvalid () [inline]
```

### 16.113.2.9 name()

```
JsonRef JsonRef::name (
 const char * name)
```

Get a [JsonRef](#) representing any entity from a parent object by key name. Useful if more than one JSON object/element has the same name.

#### Example

```
JsonRef userThree = myJsonRef.name("users")[3];
```

#### Parameters

|              |                                       |
|--------------|---------------------------------------|
| <i>*name</i> | Pointer to name of the entity to find |
|--------------|---------------------------------------|

#### Returns

The reference to the entity of type [JsonRef](#)

#### See also

[JsonRef::operator\(\)\(\)](#), [JsonRef::object\(\)](#)

### 16.113.2.10 object()

```
JsonRef JsonRef::object (
 const char * name) [inline]
```

Get a [JsonRef](#) representing an object from a parent object by key name.

#### Example

```
JsonRef john = myJsonRef.object("users").object("john");
```

#### Parameters

|              |                                       |
|--------------|---------------------------------------|
| <i>*name</i> | Pointer to name of the object to find |
|--------------|---------------------------------------|

#### Returns

The reference to the object of type [JsonRef](#)

### 16.113.2.11 operator bool()

```
JsonRef::operator bool () const [inline]
```

Operator to return a JSON element of type boolean.

#### Returns

Boolean value

### 16.113.2.12 operator const char \*()

```
JsonRef::operator const char * () const [inline]
```

Operator to return a JSON element of type const char \*.

#### Returns

const char \* value

**16.113.2.13 operator double()**

```
JsonRef::operator double () const [inline]
```

Operator to return a JSON element of type double.

**Returns**

double value

**16.113.2.14 operator float()**

```
JsonRef::operator float () const [inline]
```

Operator to return a JSON element of type float.

**Returns**

float value

**16.113.2.15 operator int()**

```
JsonRef::operator int () const [inline]
```

Operator to return a JSON element of type int.

**Returns**

integer value

**16.113.2.16 operator int16\_t()**

```
JsonRef::operator int16_t () const [inline]
```

Operator to return a JSON element of type int16\_t.

**Returns**

int16\_t value

**16.113.2.17 operator int32\_t()**

```
JsonRef::operator int32_t () const [inline]
```

Operator to return a JSON element of type int32\_t.

**Returns**

int32\_t value

**16.113.2.18 operator int8\_t()**

```
JsonRef::operator int8_t () const [inline]
```

Operator to return a JSON element of type int8\_t.

**Returns**

int8\_t value

**16.113.2.19 operator NSString()**

```
JsonRef::operator NSString () const [inline]
```

Operator to return a JSON element of type [NSString](#).

**Returns**

[NSString](#) object



**16.113.2.20 operator time\_t()**

```
JsonRef::operator time_t () const [inline]
```

Operator to return a JSON element of type `time_t`.

**Returns**

`time_t` value

**16.113.2.21 operator uint16\_t()**

```
JsonRef::operator uint16_t () const [inline]
```

Operator to return a JSON element of type `uint16_t`.

**Returns**

`uint16_t` value

**16.113.2.22 operator uint32\_t()**

```
JsonRef::operator uint32_t () const [inline]
```

Operator to return a JSON element of type `uint32_t`.

**Returns**

`uint32_t` value

**16.113.2.23 operator uint8\_t()**

```
JsonRef::operator uint8_t () const [inline]
```

Operator to return a JSON element of type `uint8_t`.

**Returns**

`uint8_t` value

**16.113.2.24 operator>()()**

```
JsonRef JsonRef::operator() (
 const char * namev) [inline]
```

Get a [JsonRef](#) representing any entity from a parent object by key name.  
Shortcut for [JsonRef::name\(\)](#)

**Example**

```
JsonRef allUsers = myJsonRef("users");
```

**Parameters**

|               |                                       |
|---------------|---------------------------------------|
| <i>*namev</i> | Pointer to name of the entity to find |
|---------------|---------------------------------------|

**Returns**

The reference to the entity of type [JsonRef](#)

**See also**

[JsonRef::name\(\)](#), [JsonRef::object\(\)](#)

**16.113.2.25 operator[]()**

```
JsonRef JsonRef::operator[] (
 int i)
```

Get a [JsonRef](#) from an array by numerical index.

**Example**

```
JsonRef userThree = myJsonRef("users")[3];
```

**Returns**

A [JsonRef](#) if array element is found

**16.113.2.26 Valid()**

```
bool JsonRef::Valid () const [inline]
```

Check if the current [JsonRef](#) is a valid JSON position.

**Returns**

True if a JSON element exists, otherwise false.

The documentation for this class was generated from the following file:

- [json\\_lexer.h](#)

**16.114 DHCP::LeaseAllocator Class Reference**

Base class/interface for lease allocators for the [DHCP](#) server.

```
#include <dhcpd.h>
```

Inherited by [DHCP::BlockAllocator](#), [DHCP::SingleAllocator](#), and [SpecialAllocator](#).

**16.114.1 Detailed Description**

Base class/interface for lease allocators for the [DHCP](#) server.

The documentation for this class was generated from the following file:

- [dhcpd.h](#)

**16.115 LetsEncryptAcmeServletObject Class Reference**

ACME Client for Let's Encrypt.

```
#include <acmeRFC8555Servlet.h>
```

Inherits [AcmeServletObject](#).

**Public Member Functions**

- [LetsEncryptAcmeServletObject](#) (const bool useStaging=false)  
*Enable the Let's Encrypt ACME client.*

**Public Member Functions inherited from [AcmeServletObject](#)**

- void [Delete\\_Everything\\_Restart](#) ()  
*Re-request the ACME certificate.*
- void [SetDiag](#) (bool v)  
*Turn on diagnostic printf messages to the console.*
- [NBString](#) [GetGlobalStateString](#) ()  
*Returns the detailed state of the ACME process.*

- [NSString](#) **GetStateString** ()  
*Returns the specific state condition of the acmeservlet.*
- const char \* [GetStateCC](#) ()  
*Returns the specific state condition of the acmeservlet.*
- [AcmeServletObject](#) (const char \*pUrlDir)  
*ACME Client Constructor.*

### 16.115.1 Detailed Description

ACME Client for Let's Encrypt.

### 16.115.2 Constructor & Destructor Documentation

#### 16.115.2.1 LetsEncryptAcmeServletObject()

```
LetsEncryptAcmeServletObject::LetsEncryptAcmeServletObject (
 const bool useStaging = false) [inline]
```

Enable the Let's Encrypt ACME client.

#### Parameters

|                   |                                                                                      |
|-------------------|--------------------------------------------------------------------------------------|
| <i>useStaging</i> | Optional boolean to use the Let's Encrypt staging server for testing (default false) |
|-------------------|--------------------------------------------------------------------------------------|

The documentation for this class was generated from the following file:

- acmeRFC8555Servlet.h

## 16.116 listen\_list\_item Struct Reference

Structure to use to create custom lists of listentiing ports and interfaces.

```
#include <init.h>
```

### 16.116.1 Detailed Description

Structure to use to create custom lists of listentiing ports and interfaces.

If you want to control what sockets and or interfaces are used for the http(s) server Then make a global or static array of these structures. One of each combination of port and interface you want. open the listening socket (with whatever port, interface optiosn you want) and assign it to the fd\_listen member. Also set if this is supposed to use TLS.

Lasttly make one extra entry in the array with a zero fd\_listen to terminate the list.

#### Parameters

|                  |                                                              |
|------------------|--------------------------------------------------------------|
| <i>fd_listen</i> | The actiaon FD opened by listen or listen via for this entry |
| <i>bUseTls</i>   | true if you wnat this socket to use tls                      |

The documentation for this struct was generated from the following file:

- init.h

## 16.117 MACADR Class Reference

Used to store and manipulate MAC addresses.

```
#include <nettypes.h>
```

## Public Member Functions

- bool **IsNull** ()  
*Checks if MAC is null.*
- bool **IsMultiCast** ()  
*Checks if MAC is a Multicast MAC.*
- bool **IsBroadCast** ()  
*Checks if MAC is a Broadcast MAC.*
- uint8\_t **GetByte** (int n) const  
*Get the nth byte of the MAC.*
- void **SetFromBytes** (const uint8\_t \*pb)  
*Set the MAC Address from a pointer to an array of 6 bytes.*
- void **fdprint** (int fd)  
*print the MAC to a file descriptor*
- void **print** ()  
*print the MAC to file descriptor 1*
- [MACADR operator+](#) (uint32\_t rhs)

### 16.117.1 Detailed Description

Used to store and manipulate MAC addresses.

### 16.117.2 Member Function Documentation

#### 16.117.2.1 operator+()

```
MACADR MACADR::operator+ (
 uint32_t rhs) [inline]
```

The documentation for this class was generated from the following file:

- nettypes.h

## 16.118 DHCP::MacPrefixAllocator Class Reference

Derived from [BlockAllocator](#) Has the ability to whitelist/blacklist certain mac address ranges based on a mask. Useful for allocating addresses based on device manufacturer.

```
#include <dhcpcd.h>
```

Inherits [DHCP::BlockAllocator](#).

### 16.118.1 Detailed Description

Derived from [BlockAllocator](#) Has the ability to whitelist/blacklist certain mac address ranges based on a mask. Useful for allocating addresses based on device manufacturer.

The documentation for this class was generated from the following file:

- dhcpcd.h

## 16.119 mcanMODM7AE70::mcan\_config Class Reference

MCAN configuration structure.

```
#include <mcan.h>
```

## Public Member Functions

- void [set\\_config\\_defaults](#) ()  
*Initializes an MCAN configuration structure to defaults.*

**Public Attributes**

- bool [run\\_in\\_standby](#)
- uint8\_t [watchdog\\_configuration](#)
- bool [transmit\\_pause](#)
- bool [edge\\_filtering](#)
- bool [protocol\\_exception\\_handling](#)
- bool [automatic\\_retransmission](#)
- bool [clock\\_stop\\_request](#)
- bool [clock\\_stop\\_acknowledge](#)
- uint8\_t [timestamp\\_prescaler](#)
- uint16\_t [timeout\\_period](#)
- enum [mcan\\_timeout\\_mode](#) [timeout\\_mode](#)
- bool [timeout\\_enable](#)
- bool [tdc\\_enable](#)
- uint8\_t [delay\\_compensation\\_offset](#)
- uint8\_t [delay\\_compensation\\_filter\\_window\\_length](#)
- enum [mcan\\_nonmatching\\_frames\\_action](#) [nonmatching\\_frames\\_action\\_standard](#)
- enum [mcan\\_nonmatching\\_frames\\_action](#) [nonmatching\\_frames\\_action\\_extended](#)
- bool [remote\\_frames\\_standard\\_reject](#)
- bool [remote\\_frames\\_extended\\_reject](#)
- uint32\_t [extended\\_id\\_mask](#)
- bool [rx\\_fifo\\_0\\_overwrite](#)
- uint8\_t [rx\\_fifo\\_0\\_watermark](#)
- bool [rx\\_fifo\\_1\\_overwrite](#)
- uint8\_t [rx\\_fifo\\_1\\_watermark](#)
- bool [tx\\_queue\\_mode](#)
- uint8\_t [tx\\_event\\_fifo\\_watermark](#)

**16.119.1 Detailed Description**

MCAN configuration structure.

Configuration structure for an MCAN instance. This structure should be initialized by the `mcan_get_config_defaults()` function before being modified by the user application.

**16.119.2 Member Function Documentation****16.119.2.1 set\_config\_defaults()**

```
void mcanMODM7AE70::mcan_config::set_config_defaults () [inline]
```

Initializes an MCAN configuration structure to defaults.

Initializes a given MCAN configuration struct to a set of known default values. This function should be called on any new instance of the configuration struct before being modified by the user application.

The default configuration is as follows:

- Not run in standby mode
- Disable Watchdog
- Transmit pause enabled
- Edge filtering during bus integration enabled
- Protocol exception handling enabled
- Automatic retransmission enabled
- Clock stop request disabled
- Clock stop acknowledge disabled

- Timestamp Counter Prescaler 1
- Timeout Period with 0xFFFF
- Timeout Mode: Continuous operation
- Disable Timeout
- Transmitter Delay Compensation Offset is 0
- Transmitter Delay Compensation Filter Window Length is 0
- Reject nonmatching standard frames
- Reject nonmatching extended frames
- Reject remote standard frames
- Reject remote extended frames
- Extended ID Mask is 0x1FFFFFFF
- Rx FIFO 0 Operation Mode: overwrite
- Disable Rx FIFO 0 Watermark
- Rx FIFO 1 Operation Mode: overwrite
- Disable Rx FIFO 1 Watermark
- Tx FIFO/Queue Mode: FIFO
- Disable Tx Event FIFO Watermark

### 16.119.3 Member Data Documentation

#### 16.119.3.1 automatic\_retransmission

`bool mcanMODM7AE70::mcan_config::automatic_retransmission`  
Automatic Retransmission.

#### 16.119.3.2 clock\_stop\_acknowledge

`bool mcanMODM7AE70::mcan_config::clock_stop_acknowledge`  
Clock Stop Acknowledge.

#### 16.119.3.3 clock\_stop\_request

`bool mcanMODM7AE70::mcan_config::clock_stop_request`  
Clock Stop Request.

#### 16.119.3.4 delay\_compensation\_filter\_window\_length

`uint8_t mcanMODM7AE70::mcan_config::delay_compensation_filter_window_length`  
Transmitter Delay Compensation Filter Window Length : 0x0-0x7F

#### 16.119.3.5 delay\_compensation\_offset

`uint8_t mcanMODM7AE70::mcan_config::delay_compensation_offset`  
Transmitter Delay Compensation Offset : 0x0-0x7F

#### 16.119.3.6 edge\_filtering

`bool mcanMODM7AE70::mcan_config::edge_filtering`  
Edge Filtering during Bus Integration.

### 16.119.3.7 extended\_id\_mask

uint32\_t mcanMODM7AE70::mcan\_config::extended\_id\_mask  
Extended ID Mask: 0x0-0x1FFFFFFF.

### 16.119.3.8 nonmatching\_frames\_action\_extended

enum [mcan\\_nonmatching\\_frames\\_action](#) mcanMODM7AE70::mcan\_config::nonmatching\_frames\_action\_↵  
extended  
Nonmatching frames action for extended frames.

### 16.119.3.9 nonmatching\_frames\_action\_standard

enum [mcan\\_nonmatching\\_frames\\_action](#) mcanMODM7AE70::mcan\_config::nonmatching\_frames\_action\_↵  
standard  
Nonmatching frames action for standard frames.

### 16.119.3.10 protocol\_exception\_handling

bool mcanMODM7AE70::mcan\_config::protocol\_exception\_handling  
Protocol Exception Handling.

### 16.119.3.11 remote\_frames\_extended\_reject

bool mcanMODM7AE70::mcan\_config::remote\_frames\_extended\_reject  
Reject Remote Extended Frames.

### 16.119.3.12 remote\_frames\_standard\_reject

bool mcanMODM7AE70::mcan\_config::remote\_frames\_standard\_reject  
Reject Remote Standard Frames.

### 16.119.3.13 run\_in\_standby

bool mcanMODM7AE70::mcan\_config::run\_in\_standby  
MCAN run in standby control.

### 16.119.3.14 rx\_fifo\_0\_overwrite

bool mcanMODM7AE70::mcan\_config::rx\_fifo\_0\_overwrite  
Rx FIFO 0 Operation Mode.

### 16.119.3.15 rx\_fifo\_0\_watermark

uint8\_t mcanMODM7AE70::mcan\_config::rx\_fifo\_0\_watermark  
Rx FIFO 0 Watermark: 1-64, other value disable it.

### 16.119.3.16 rx\_fifo\_1\_overwrite

bool mcanMODM7AE70::mcan\_config::rx\_fifo\_1\_overwrite  
Rx FIFO 1 Operation Mode.

### 16.119.3.17 rx\_fifo\_1\_watermark

uint8\_t mcanMODM7AE70::mcan\_config::rx\_fifo\_1\_watermark  
Rx FIFO 1 Watermark: 1-64, other value disable it.

### 16.119.3.18 tdc\_enable

bool mcanMODM7AE70::mcan\_config::tdc\_enable  
Transceiver Delay Compensation enable.

**16.119.3.19 timeout\_enable**

bool mcanMODM7AE70::mcan\_config::timeout\_enable  
Timeout enable.

**16.119.3.20 timeout\_mode**

enum mcan\_timeout\_mode mcanMODM7AE70::mcan\_config::timeout\_mode  
Timeout Mode.

**16.119.3.21 timeout\_period**

uint16\_t mcanMODM7AE70::mcan\_config::timeout\_period  
Timeout Period.

**16.119.3.22 timestamp\_prescaler**

uint8\_t mcanMODM7AE70::mcan\_config::timestamp\_prescaler  
Timestamp Counter Prescaler: 0x0-0xF

**16.119.3.23 transmit\_pause**

bool mcanMODM7AE70::mcan\_config::transmit\_pause  
Transmit Pause.

**16.119.3.24 tx\_event\_fifo\_watermark**

uint8\_t mcanMODM7AE70::mcan\_config::tx\_event\_fifo\_watermark  
Tx Event FIFO Watermark: 1-32, other value disable it.

**16.119.3.25 tx\_queue\_mode**

bool mcanMODM7AE70::mcan\_config::tx\_queue\_mode  
Tx FIFO/Queue Mode, 0 for FIFO and 1 for Queue.

**16.119.3.26 watchdog\_configuration**

uint8\_t mcanMODM7AE70::mcan\_config::watchdog\_configuration  
Start value of the Message RAM Watchdog Counter  
The documentation for this class was generated from the following file:

- mcan.h

**16.120 mcan\_extended\_message\_filter\_element Class Reference**

MCAN extended message ID filter element structure.

```
#include <mcan_internal.h>
```

**16.120.1 Detailed Description**

MCAN extended message ID filter element structure.

Common element structure for extended message ID filter element.

The documentation for this class was generated from the following file:

- mcan\_internal.h

**16.121 mcanMODM7AE70::mcan\_module Class Reference**

MCAN Module Class.

```
#include <mcan.h>
```



## Public Member Functions

- [mcan\\_module](#) (Mcan \*phw, uint32\_t baud)  
*Create a MCAN module instance.*
- void [init](#) (Mcan \*phw, struct [mcan\\_config](#) \*config, uint32\_t baud)  
*Initialize a MCAN module.*
- void [send\\_message](#) (uint32\_t id\_value, uint8\_t \*data, uint32\_t data\_length, OS\_SEM \*pSem=0)  
*Send a MCAN message.*
- bool [blocking\\_send\\_message](#) (uint32\_t id\_value, uint8\_t \*data, uint32\_t data\_length, uint32\_t TimeOut)  
*Send a MCAN message and block until sent.*
- int [RegisterRxFifo](#) (uint32\_t composite\_id, OS\_FIFO \*pFifo, int channel=-1)  
*Register a FIFO to receive CAN messages.*
- int [RegisterRxFifoMask](#) (uint32\_t composite\_id, uint32\_t mask, OS\_FIFO \*pFifo, int channel=-1)  
*Register a FIFO to receive messages as specified by the address bit mask. A value of 1 = match, a value of 0 = does not match.*
- int [RegisterRxFifoRange](#) (uint32\_t composite\_id\_low, uint32\_t composite\_id\_hi, OS\_FIFO \*pFifo, int channel=-1)  
*Register a FIFO to receive messages in the specified numeric range of addresses from low to high.*
- int [UnRegisterFifo](#) (int channel)  
*Unregister the FIFO for the specified CAN channel.*
- int [MultiCanSetRTRMessage](#) (uint32\_t id, uint8\_t \*data, uint8\_t len, int channel=-1)  
*Enable an auto-reply to a CAN RTR message for the specified CAN ID. The reply message is specified by the data parameter.*
- int [MultiCanReplaceRTRMessage](#) (int channel, uint8\_t \*data, uint8\_t len)  
*Modifies/updates the response for the RTR message registered with [MultiCanSetRTRMessage\(\)](#).*
- int [MultiCanStopRTRMessage](#) (int channel)  
*Unregister the RTR message for the specified channel.*

### 16.121.1 Detailed Description

MCAN Module Class.

Class to control a MCAN modules (peripherals) on the MODM7AE70

### 16.121.2 Constructor & Destructor Documentation

#### 16.121.2.1 mcan\_module()

```
mcanMODM7AE70::mcan_module::mcan_module (
 Mcan * phw,
 uint32_t baud)
```

Create a MCAN module instance.

#### Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>phw</i>  | MCAN hardware peripheral, MCAN0 or MCAN 1 |
| <i>baud</i> | MCAN baud rate. For example, 500000       |

### 16.121.3 Member Function Documentation

#### 16.121.3.1 blocking\_send\_message()

```
bool mcanMODM7AE70::mcan_module::blocking_send_message (
 uint32_t id_value,
 uint8_t * data,
```

```
uint32_t data_length,
uint32_t TimeOut)
```

Send a MCAN message and block until sent.

## Parameters

|                    |                                                                        |
|--------------------|------------------------------------------------------------------------|
| <i>id_value</i>    | CAN ID, 11-bit standard or 29-bit extended                             |
| <i>data</i>        | Pointer to message data                                                |
| <i>data_length</i> | Length of data in bytes                                                |
| <i>TimeOut</i>     | Time out value in system time ticks. For example, TICKS_PER_SECOND * 1 |

## 16.121.3.2 init()

```
void mcanMODM7AE70::mcan_module::init (
 Mcan * phw,
 struct mcan_config * config,
 uint32_t baud)
```

Initialize a MCAN module.

## Parameters

|               |                                                                                       |
|---------------|---------------------------------------------------------------------------------------|
| <i>phw</i>    | MCAN hardware peripheral, MCAN0 or MCAN 1                                             |
| <i>config</i> | Pointer to a <a href="#">mcan_config</a> object, initialized with appropriate values. |
| <i>baud</i>   | MCAN baud rate. For example, 500000                                                   |

## 16.121.3.3 MultiCanReplaceRTRMessage()

```
int mcanMODM7AE70::mcan_module::MultiCanReplaceRTRMessage (
 int channel,
 uint8_t * data,
 uint8_t len)
```

Modifies/updates the response for the RTR message registered with [MultiCanSetRTRMessage\(\)](#).

## Returns

The channel number registered for the RTR message.

## See also

[MultiCanSetRTRMessage\(\)](#), [MultiCanStopRTRMessage\(\)](#)

## 16.121.3.4 MultiCanSetRTRMessage()

```
int mcanMODM7AE70::mcan_module::MultiCanSetRTRMessage (
 uint32_t id,
 uint8_t * data,
 uint8_t len,
 int channel = -1)
```

Enable an auto-reply to a CAN RTR message for the specified CAN ID. The reply message is specified by the data parameter.

The response message can be modified with [MultiCanReplaceRTRMessage\(\)](#).

## Parameters

|                |                                            |
|----------------|--------------------------------------------|
| <i>id</i>      | CAN ID, 11-bit standard or 29-bit extended |
| <i>data</i>    | Pointer to message data                    |
| <i>len</i>     | Length of message                          |
| <i>channel</i> | Optional CAN channel                       |

**Returns**

The channel number corresponding to the ID and RTR message.

**See also**

[MultiCanReplaceRTRMessage\(\)](#), [MultiCanStopRTRMessage\(\)](#)

**16.121.3.5 MultiCanStopRTRMessage()**

```
int mcanMODM7AE70::mcan_module::MultiCanStopRTRMessage (
 int channel)
```

Unregister the RTR message for the specified channel.

**Parameters**

|                |                    |
|----------------|--------------------|
| <i>channel</i> | CAN channel number |
|----------------|--------------------|

**Returns**

OS\_NO\_ERR on success

**See also**

[MultiCanSetRTRMessage\(\)](#), [MultiCanReplaceRTRMessage\(\)](#)

**16.121.3.6 RegisterRxFifo()**

```
int mcanMODM7AE70::mcan_module::RegisterRxFifo (
 uint32_t composite_id,
 OS_FIFO * pFifo,
 int channel = -1)
```

Register a FIFO to receive CAN messages.

**Parameters**

|                     |                                            |
|---------------------|--------------------------------------------|
| <i>composite_id</i> | CAN ID, 11-bit standard or 29-bit extended |
| <i>pFifo</i>        | Pointer to OS_FIFO for received data       |
| <i>channel</i>      | Optional CAN channel                       |

**Returns**

OS\_NO\_ERR on success

**16.121.3.7 RegisterRxFifoMask()**

```
int mcanMODM7AE70::mcan_module::RegisterRxFifoMask (
 uint32_t composite_id,
 uint32_t mask,
 OS_FIFO * pFifo,
 int channel = -1)
```

Register a FIFO to receive messages as specified by the address bit mask. A value of 1 = match, a value of 0 = does not match.

## Parameters

|                     |                                            |
|---------------------|--------------------------------------------|
| <i>composite_id</i> | CAN ID, 11-bit standard or 29-bit extended |
| <i>mask</i>         | Id mask                                    |
| <i>pFifo</i>        | Pointer to OS_FIFO for received data       |
| <i>channel</i>      | Optional CAN channel                       |

## Returns

OS\_NO\_ERR on success

**16.121.3.8 RegisterRxFifoRange()**

```
int mcanMODM7AE70::mcan_module::RegisterRxFifoRange (
 uint32_t composite_id_low,
 uint32_t composite_id_hi,
 OS_FIFO * pFifo,
 int channel = -1)
```

Register a FIFO to receive messages in the specified numeric range of addresses from low to high.

## Parameters

|                         |                                      |
|-------------------------|--------------------------------------|
| <i>composite_id_low</i> | Composite ID low value               |
| <i>composite_id_hi</i>  | Composite ID high value              |
| <i>pFifo</i>            | Pointer to OS_FIFO for received data |
| <i>channel</i>          | Optional CAN channel                 |

## Returns

OS\_NO\_ERR on success

**16.121.3.9 send\_message()**

```
void mcanMODM7AE70::mcan_module::send_message (
 uint32_t id_value,
 uint8_t * data,
 uint32_t data_length,
 OS_SEM * pSem = 0)
```

Send a MCAN message.

## Parameters

|                    |                                                              |
|--------------------|--------------------------------------------------------------|
| <i>id_value</i>    | CAN ID, 11-bit standard or 29-bit extended                   |
| <i>data</i>        | Pointer to message data                                      |
| <i>data_length</i> | Length of data in bytes                                      |
| <i>pSem</i>        | Optional pointer to semaphore to post to, default is no post |

**16.121.3.10 UnRegisterFifo()**

```
int mcanMODM7AE70::mcan_module::UnRegisterFifo (
 int channel)
```

Unregister the FIFO for the specified CAN channel.

**Parameters**

|                |                    |
|----------------|--------------------|
| <i>channel</i> | CAN channel number |
|----------------|--------------------|

**Returns**

OS\_NO\_ERR on success

The documentation for this class was generated from the following file:

- mcan.h

## 16.122 mcan\_rx\_element\_buffer Struct Reference

MCAN receive element structure for buffer.

```
#include <mcan_internal.h>
```

### 16.122.1 Detailed Description

MCAN receive element structure for buffer.

The documentation for this struct was generated from the following file:

- mcan\_internal.h

## 16.123 mcan\_tx\_element Struct Reference

MCAN transfer element structure.

```
#include <mcan_internal.h>
```

### 16.123.1 Detailed Description

MCAN transfer element structure.

Common element structure for transfer buffer and FIFO/QUEUE.

The documentation for this struct was generated from the following file:

- mcan\_internal.h

## 16.124 mcan\_tx\_event\_element Struct Reference

MCAN transfer event FIFO element structure.

```
#include <mcan_internal.h>
```

### 16.124.1 Detailed Description

MCAN transfer event FIFO element structure.

Common element structure for transfer event FIFO.

The documentation for this struct was generated from the following file:

- mcan\_internal.h

## 16.125 MQTT::mqtt\_bool Class Reference

Easy MqttObj class for automatically publishing and subscribing boolean values to/from an [MQTT](#) topic. Use like a normal bool.

```
#include <mqtt.h>
```

Inherits [MQTT::mqtt\\_leaf](#).

## Public Member Functions

- [mqtt\\_bool](#) (bool val, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Su)  
*Constructor for a standalone [mqtt\\_bool](#) object.*
- [mqtt\\_bool](#) (const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Su)  
*Constructor for a standalone [mqtt\\_bool](#) object.*
- [mqtt\\_bool](#) ([mqtt\\_leaf](#) &parent, bool val, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Su)  
*Constructor for a [mqtt\\_bool](#) object that has a parent.*
- [mqtt\\_bool](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Su)  
*Constructor for a [mqtt\\_bool](#) object that has a parent.*
- [mqtt\\_bool](#) & operator= (bool i)  
*Assignment operator for [mqtt\\_bool](#). Assigns the value to the object, then performs any post assignment operations.*
- [operator bool](#) () const  
*Boolean conversion operator for [mqtt\\_bool](#). Returns the value of the object as an boolean.*
- [bool operator!](#) () const  
*Boolean Not operator for [mqtt\\_bool](#). Returns the negated value of the object as an boolean.*
- virtual [ValueTestResult TestOrSetValue](#) ([ParsedJsonDataSet](#) &pjs, bool setVal) override  
*whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.*
- virtual [int SubscribeEvent](#) ([int](#) mqttFd, [PacketInfo::Msg](#) \*msg) override  
*Handle a message sent by the Broker triggered by this object's Subscription.*
- virtual [int RenderToBuffer](#) ([eObj\\_Serialize\\_t](#) form, [uint8\\_t](#) \*buf, [int](#) buflen) override  
*Render this object to an output buffer.*
- virtual [int RenderToFd](#) ([eObj\\_Serialize\\_t](#) form, [int](#) fd) override  
*Render this object to an output socket.*

Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- [~mqtt\\_leaf](#) ()  
*~mqtt\_leaf Destructor for base [mqtt\\_leaf](#)*
- void [RemoveFromTree](#) ()  
*Remove this object from it's parent tree.*
- [bool NameMatch](#) (const char \*cp)  
*Determine whether a given name matches this object.*
- virtual [int GetPubTopicName](#) ([NBString](#) &s)  
*Get the Topic Name this object Publishes to.*
- virtual [int GetSubTopicFilter](#) ([NBString](#) &s)  
*Get the Subscription Topic Filter this object Subscribes to.*
- virtual [int GetRenderLength](#) ([eObj\\_Serialize\\_t](#) form=[eObj\\_Serialize\\_CURRENT\\_FORM](#))  
*Get the number of bytes used to serialize this object in a particular format.*
- [int Publish](#) ([int](#) waitForCompletion=-1)  
*Publishes the current [MQTT](#) object, optionally waiting for completion.*
- void [SetPublishInterval](#) ([uint32\\_t](#) intervalPeriodTicks)  
*Sets or clears an automatic publishing interval for this [MQTT](#) object.*
- virtual void [SetSerializationForm](#) ([eObj\\_Serialize\\_t](#) form)  
*Set the default serialization format for this object.*
- [eObj\\_Serialize\\_t GetSerializationForm](#) ()  
*Get the default serialization format for this object.*
- virtual void [SetFlags](#) ([uint16\\_t](#) newFlags)  
*Set additional behavioral flags for this object.*
- virtual void [ClearFlags](#) ([uint16\\_t](#) delFlags)  
*Clear behavioral flags from this object.*

- uint16\_t [GetFlags](#) ()  
*Get the behavioral flags set for this object.*
- int [getPriority](#) ()  
*Get the message priority for this object.*

### Protected Member Functions

- virtual int [ParseFd](#) (int mqttFd) override  
*the data from a file descriptor according to this object's current default serialization method and assign the value.*

### Protected Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- [mqtt\\_leaf](#) (const char \*topic, [Client](#) \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_N  
*Constructor for [mqtt\\_leaf](#) objects.*
- [mqtt\\_leaf](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|  
*Constructor for [mqtt\\_leaf](#) objects.*
- virtual [mqtt\\_leaf](#) \* [GetChildren](#) ()  
*Get the first child of this object.*
- virtual [mqtt\\_leaf](#) \* [AddChild](#) ([mqtt\\_leaf](#) \*child)  
*Add an MqttObj as a child to this object.*
- virtual [mqtt\\_leaf](#) \* [RemoveFirstChild](#) ()  
*Remove the first child from this object.*
- void [ForEachLeaf](#) (LeafCallBack \*p, void \*pd)  
*Call a function on each element of the MqttObj tree with this object as the root.*
- void [RootParse](#) ([ParsedJsonDataSet](#) &pjs)  
*Assign to this object tree from a JSON dataset.*
- bool [MatchId](#) ([ParsedJsonDataSet](#) &pjs)  
*Determine whether the JSON dataset's current index matches this object's name.*
- int [QueuePublish](#) ()  
*Queue this object to be Published via the registered [Client](#).*
- int [QueueSubscribe](#) ()  
*Queue this object to be Subscribed via the registered [Client](#).*
- int [QueueUnsubscribe](#) ()  
*Queue this object to be Unsubscribed via the registered [Client](#).*
- int [Assign](#) ()  
*Perform post assignemnt operations.*
- uint8\_t [PubState](#) ()  
*Get the Publish state.*
- uint8\_t [SubState](#) ()  
*Get the Subscribe state of this object.*
- virtual void [SetPubState](#) (uint8\_t newState)  
*Set the Publish state.*
- virtual void [SetSubState](#) (uint8\_t newState)  
*Set the Subscribe state.*
- int [HandleSubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle a Subscribe Acknowledge message from the broker for our Subscribe request.*
- int [HandleUnsubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.*
- int [HandlePubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Publish Acknowledge message from the broker for our Publish request.*



## Additional Inherited Members

### Public Types inherited from MQTT::mqtt\_leaf

- enum [eObj\\_State\\_t](#)  
*Mqtt Object Publication/Subscription States.*

### Static Public Member Functions inherited from MQTT::mqtt\_leaf

- static int [MqttRxPublish](#) ([Client](#) \*mClient, int mqttFd, [PacketInfo::Msg](#) \*msg, void \*ctx)  
*Static message handler for Publish messages received by the registered [Client](#).*

### Protected Attributes inherited from MQTT::mqtt\_leaf

- [NBInterpolatedString](#) **topicName**  
*Topic name or additional branch.*
- [eObj\\_Serialize\\_t](#) **publishForm**  
*Default serialization form.*
- uint16\_t **mFlags**  
*Behavioral flags.*

## 16.125.1 Detailed Description

Easy MqttObj class for automatically publishing and subscribing boolean values to/from an [MQTT](#) topic. Use like a normal bool.

## 16.125.2 Constructor & Destructor Documentation

### 16.125.2.1 mqtt\_bool() [1/4]

```
MQTT::mqtt_bool::mqtt_bool (
 bool val,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a standalone [mqtt\\_bool](#) object.

#### Parameters

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>val</i>      | Initial value for the boolean                                                                                                      |
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>    | Behavioral flags to set on the created object                                                                                      |

### 16.125.2.2 mqtt\_bool() [2/4]

```
MQTT::mqtt_bool::mqtt_bool (
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a standalone [mqtt\\_bool](#) object.

#### Parameters

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |

|              |                                               |
|--------------|-----------------------------------------------|
| <i>flags</i> | Behavioral flags to set on the created object |
|--------------|-----------------------------------------------|

### 16.125.2.3 mqtt\_bool() [3/4]

```
MQTT::mqtt_bool::mqtt_bool (
 mqtt_leaf & parent,
 bool val,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a [mqtt\\_bool](#) object that has a parent.

#### Parameters

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>val</i>         | Initial value for the boolean                                                                                                      |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i>    | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

### 16.125.2.4 mqtt\_bool() [4/4]

```
MQTT::mqtt_bool::mqtt_bool (
 mqtt_leaf & parent,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a [mqtt\\_bool](#) object that has a parent.

#### Parameters

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i>    | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

## 16.125.3 Member Function Documentation

### 16.125.3.1 operator bool()

```
MQTT::mqtt_bool::operator bool () const [inline]
```

Boolean conversion operator for [mqtt\\_bool](#). Returns the value of the object as an boolean.

#### Returns

The boolean value of the object

### 16.125.3.2 operator"!()"

```
bool MQTT::mqtt_bool::operator! () const [inline]
```

Boolean Not operator for [mqtt\\_bool](#). Returns the negated value of the object as an boolean.

#### Returns

The negated boolean value of the object

### 16.125.3.3 operator=()

```
mqtt_bool & MQTT::mqtt_bool::operator= (
 bool i)
```

Assignment operator for [mqtt\\_bool](#). Assigns the value to the object, then performs any post assignment operations.

#### Parameters

|          |                                    |
|----------|------------------------------------|
| <i>i</i> | The value to assign to the object. |
|----------|------------------------------------|

#### Returns

A reference to the object being assigned to.

### 16.125.3.4 ParseFd()

```
virtual int MQTT::mqtt_bool::ParseFd (
 int mqttFd) [override], [protected], [virtual]
```

the data from a file descriptor according to this object's current default serialization method and assign the value.

#### Parameters

|               |                               |
|---------------|-------------------------------|
| <i>mqttFd</i> | File descriptor to read from. |
|---------------|-------------------------------|

#### Returns

eResult\_t return code

Implements [MQTT::mqtt\\_leaf](#).

### 16.125.3.5 RenderToBuffer()

```
virtual int MQTT::mqtt_bool::RenderToBuffer (
 eObj_Serialize_t form,
 uint8_t * buf,
 int buflen) [override], [virtual]
```

Render this object to an output buffer.

#### Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>form</i>   | Serialization format to render as |
| <i>*buf</i>   | Pointer to output buffer          |
| <i>buflen</i> | Size of the output buffer         |

#### Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

### 16.125.3.6 RenderToFd()

```
virtual int MQTT::mqtt_bool::RenderToFd (
 eObj_Serialize_t form,
 int fd) [override], [virtual]
```

Render this object to an output socket.

## Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>form</i> | Serialization format to render as |
| <i>fd</i>   | File descriptor to render to      |

## Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

**16.125.3.7 SubscribeEvent()**

```
virtual int MQTT::mqtt_bool::SubscribeEvent (
 int mqttFd,
 PacketInfo::Msg * msg) [override], [virtual]
```

Handle a message sent by the Broker triggered by this object's Subscription.

## Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>mqttFd</i> | File Descriptor to read any additional message data from |
| <i>*msg</i>   | Pointer to message received from the Broker              |

## Returns

eResult\_t return code

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.125.3.8 TestOrSetValue()**

```
virtual ValueTestResult MQTT::mqtt_bool::TestOrSetValue (
 ParsedJsonDataSet & pjs,
 bool setVal) [override], [virtual]
```

whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.

## Parameters

|                 |                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------|
| <i>&amp;pjs</i> | The JSON dataset to use                                                                                 |
| <i>setVal</i>   | true: Assign from the value at the current index, false: test if current index is valid for this object |

## Returns

ValueTestResult indicating the validity of the current index

Implements [MQTT::mqtt\\_leaf](#).

The documentation for this class was generated from the following file:

- mqtt.h

**16.126 MQTT::mqtt\_float Class Reference**

Easy MqttObj class for automatically publishing and subscribing floating point values to/from an [MQTT](#) topic. Use like a normal double.

```
#include <mqtt.h>
```

Inherits [MQTT::mqtt\\_leaf](#).

## Public Member Functions

- [mqtt\\_float](#) (double val, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_NoLocal))  
*Constructor for a standalone [mqtt\\_float](#) object.*
- [mqtt\\_float](#) (const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_NoLocal))  
*Constructor for a standalone [mqtt\\_float](#) object.*
- [mqtt\\_float](#) ([mqtt\\_leaf](#) &parent, double val, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_NoLocal))  
*Constructor for a [mqtt\\_float](#) object that has a parent.*
- [mqtt\\_float](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_NoLocal))  
*Constructor for a [mqtt\\_float](#) object that has a parent.*
- [mqtt\\_float](#) & operator= (double f)  
*Assignment operator for [mqtt\\_float](#). Assigns the value to the object, then performs any post assignment operations.*
- [operator double](#) () const  
*Double precision floating point conversion operator for [mqtt\\_uint](#). Returns the value of the object as double.*
- void [SetStringFormat](#) (char formatChar, uint8\_t precision)  
*the printf style format char to use when serializing this object in a string form.*
- virtual ValueTestResult [TestOrSetValue](#) ([ParsedJsonDataSet](#) &pjs, bool setVal) override  
*whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.*
- virtual int [SubscribeEvent](#) (int mqttFd, [PacketInfo::Msg](#) \*msg) override  
*Handle a message sent by the Broker triggered by this object's Subscription.*
- virtual int [RenderToBuffer](#) (eObj\_Serialize\_t form, uint8\_t \*buf, int buflen) override  
*Render this object to an output buffer.*
- virtual int [RenderToFd](#) (eObj\_Serialize\_t form, int fd) override  
*Render this object to an output socket.*

Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- [~mqtt\\_leaf](#) ()  
*~mqtt\_leaf Destructor for base [mqtt\\_leaf](#)*
- void [RemoveFromTree](#) ()  
*Remove this object from it's parent tree.*
- bool [NameMatch](#) (const char \*cp)  
*Determine whether a given name matches this object.*
- virtual int [GetPubTopicName](#) (NBString &s)  
*Get the Topic Name this object Publishes to.*
- virtual int [GetSubTopicFilter](#) (NBString &s)  
*Get the Subscription Topic Filter this object Subscribes to.*
- virtual int [GetRenderLength](#) (eObj\_Serialize\_t form=eObj\_Serialize\_CURRENT\_FORM)  
*Get the number of bytes used to serialize this object in a particular format.*
- int [Publish](#) (int waitForCompletion=-1)  
*Publishes the current [MQTT](#) object, optionally waiting for completion.*
- void [SetPublishInterval](#) (uint32\_t intervalPeriodTicks)  
*Sets or clears an automatic publishing interval for this [MQTT](#) object.*
- virtual void [SetSerializationForm](#) (eObj\_Serialize\_t form)  
*Set the default serialization format for this object.*
- eObj\_Serialize\_t [GetSerializationForm](#) ()  
*Get the default serialization format for this object.*
- virtual void [SetFlags](#) (uint16\_t newFlags)  
*Set additional behavioral flags for this object.*
- virtual void [ClearFlags](#) (uint16\_t delFlags)

*Clear behavioral flags from this object.*

- uint16\_t [GetFlags](#) ()

*Get the behavioral flags set for this object.*

- int [getPriority](#) ()

*Get the message priority for this object.*

### Protected Member Functions

- virtual int [ParseFd](#) (int mqttFd) override

*the data from a file descriptor according to this object's current default serialization method and assign the value.*

### Protected Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- [mqtt\\_leaf](#) (const char \*topic, [Client](#) \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_N

*Constructor for [mqtt\\_leaf](#) objects.*

- [mqtt\\_leaf](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|

*Constructor for [mqtt\\_leaf](#) objects.*

- virtual [mqtt\\_leaf](#) \* [GetChildren](#) ()

*Get the first child of this object.*

- virtual [mqtt\\_leaf](#) \* [AddChild](#) ([mqtt\\_leaf](#) \*child)

*Add an MqttObj as a child to this object.*

- virtual [mqtt\\_leaf](#) \* [RemoveFirstChild](#) ()

*Remove the first child from this object.*

- void [ForEachLeaf](#) (LeafCallBack \*p, void \*pd)

*Call a function on each element of the MqttObj tree with this object as the root.*

- void [RootParse](#) ([ParsedJsonDataSet](#) &pjs)

*Assign to this object tree from a JSON dataset.*

- bool [MatchId](#) ([ParsedJsonDataSet](#) &pjs)

*Determine whether the JSON dataset's current index matches this object's name.*

- int [QueuePublish](#) ()

*Queue this object to be Published via the registered [Client](#).*

- int [QueueSubscribe](#) ()

*Queue this object to be Subscribed via the registered [Client](#).*

- int [QueueUnsubscribe](#) ()

*Queue this object to be Unsubscribed via the registered [Client](#).*

- int [Assign](#) ()

*Perform post assignemnt operations.*

- uint8\_t [PubState](#) ()

*Get the Publish state.*

- uint8\_t [SubState](#) ()

*Get the Subscribe state of this object.*

- virtual void [SetPubState](#) (uint8\_t newState)

*Set the Publish state.*

- virtual void [SetSubState](#) (uint8\_t newState)

*Set the Subscribe state.*

- int [HandleSubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)

*Handle a Subscribe Acknowledge message from the broker for our Subscribe request.*

- int [HandleUnsubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)

*Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.*

- int [HandlePubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)

*Handle an Publish Acknowledge message from the broker for our Publish request.*

## Additional Inherited Members

### Public Types inherited from MQTT::mqtt\_leaf

- enum [eObj\\_State\\_t](#)  
*Mqtt Object Publication/Subscription States.*

### Static Public Member Functions inherited from MQTT::mqtt\_leaf

- static int [MqttRxPublish](#) ([Client](#) \*mClient, int mqttFd, [PacketInfo::Msg](#) \*msg, void \*ctx)  
*Static message handler for Publish messages received by the registered [Client](#).*

### Protected Attributes inherited from MQTT::mqtt\_leaf

- [NBInterpolatedString](#) **topicName**  
*Topic name or additional branch.*
- [eObj\\_Serialize\\_t](#) **publishForm**  
*Default serialization form.*
- uint16\_t **mFlags**  
*Behavioral flags.*

## 16.126.1 Detailed Description

Easy MqttObj class for automatically publishing and subscribing floating point values to/from an [MQTT](#) topic. Use like a normal double.

## 16.126.2 Constructor & Destructor Documentation

### 16.126.2.1 mqtt\_float() [1/4]

```
MQTT::mqtt_float::mqtt_float (
 double val,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a standalone [mqtt\\_float](#) object.

#### Parameters

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>val</i>      | Initial value for the float                                                                                                        |
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>    | Behavioral flags to set on the created object                                                                                      |

### 16.126.2.2 mqtt\_float() [2/4]

```
MQTT::mqtt_float::mqtt_float (
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a standalone [mqtt\\_float](#) object.

#### Parameters

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |

|              |                                               |
|--------------|-----------------------------------------------|
| <i>flags</i> | Behavioral flags to set on the created object |
|--------------|-----------------------------------------------|

### 16.126.2.3 mqtt\_float() [3/4]

```
MQTT::mqtt_float::mqtt_float (
 mqtt_leaf & parent,
 double val,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a [mqtt\\_float](#) object that has a parent.

#### Parameters

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>val</i>         | Initial value for the float                                                                                                        |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i>    | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

### 16.126.2.4 mqtt\_float() [4/4]

```
MQTT::mqtt_float::mqtt_float (
 mqtt_leaf & parent,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a [mqtt\\_float](#) object that has a parent.

#### Parameters

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i>    | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

## 16.126.3 Member Function Documentation

### 16.126.3.1 operator double()

```
MQTT::mqtt_float::operator double () const [inline]
```

Double precision floating point conversion operator for [mqtt\\_uint](#). Returns the value of the object as double.

#### Returns

The value of the object

### 16.126.3.2 operator=()

```
mqtt_float & MQTT::mqtt_float::operator= (
 double f)
```

Assignment operator for [mqtt\\_float](#). Assigns the value to the object, then performs any post assignment operations.



## Parameters

|          |                                    |
|----------|------------------------------------|
| <i>f</i> | The value to assign to the object. |
|----------|------------------------------------|

## Returns

A reference to the object being assigned to.

**16.126.3.3 ParseFd()**

```
virtual int MQTT::mqtt_float::ParseFd (
 int mqttFd) [override], [protected], [virtual]
```

the data from a file descriptor according to this object's current default serialization method and assign the value.

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>mqttFd</i> | File descriptor to read from. |
|---------------|-------------------------------|

## Returns

eResult\_t return code

Implements [MQTT::mqtt\\_leaf](#).

**16.126.3.4 RenderToBuffer()**

```
virtual int MQTT::mqtt_float::RenderToBuffer (
 eObj_Serialize_t form,
 uint8_t * buf,
 int buflen) [override], [virtual]
```

Render this object to an output buffer.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>form</i>   | Serialization format to render as |
| <i>*buf</i>   | Pointer to output buffer          |
| <i>buflen</i> | Size of the output buffer         |

## Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

**16.126.3.5 RenderToFd()**

```
virtual int MQTT::mqtt_float::RenderToFd (
 eObj_Serialize_t form,
 int fd) [override], [virtual]
```

Render this object to an output socket.

## Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>form</i> | Serialization format to render as |
| <i>fd</i>   | File descriptor to render to      |

## Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

### 16.126.3.6 SetStringFormat()

```
void MQTT::mqtt_float::SetStringFormat (
 char formatChar,
 uint8_t precision) [inline]
```

the printf style format char to use when serializing this object in a string form.

#### Parameters

|                   |                                      |
|-------------------|--------------------------------------|
| <i>formatChar</i> | The format character to use          |
| <i>precision</i>  | The precision to use when formatting |

### 16.126.3.7 SubscribeEvent()

```
virtual int MQTT::mqtt_float::SubscribeEvent (
 int mqttFd,
 PacketInfo::Msg * msg) [override], [virtual]
```

Handle a message sent by the Broker triggered by this object's Subscription.

#### Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>mqttFd</i> | File Descriptor to read any additional message data from |
| <i>*msg</i>   | Pointer to message received from the Broker              |

#### Returns

eResult\_t return code

Reimplemented from [MQTT::mqtt\\_leaf](#).

### 16.126.3.8 TestOrSetValue()

```
virtual ValueTestResult MQTT::mqtt_float::TestOrSetValue (
 ParsedJsonDataSet & pjs,
 bool setVal) [override], [virtual]
```

whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.

#### Parameters

|                 |                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------|
| <i>&amp;pjs</i> | The JSON dataset to use                                                                                 |
| <i>setVal</i>   | true: Assign from the value at the current index, false: test if current index is valid for this object |

#### Returns

ValueTestResult indicating the validity of the current index

Implements [MQTT::mqtt\\_leaf](#).

The documentation for this class was generated from the following file:

- mqtt.h

## 16.127 MQTT::mqtt\_int Class Reference

Easy MqttObj class for automatically publishing and subscribing unsigned integer values to/from an [MQTT](#) topic. Use like a normal int32\_t.

```
#include <mqtt.h>
```

Inherits [MQTT::mqtt\\_leaf](#).

## Public Member Functions

- [mqtt\\_int](#) (int val, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_S...)  
*Constructor for a standalone [mqtt\\_int](#) object.*
- [mqtt\\_int](#) (const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_S...)  
*Constructor for a standalone [mqtt\\_int](#) object.*
- [mqtt\\_int](#) ([mqtt\\_leaf](#) &parent, int val, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_S...)  
*Constructor for a standalone [mqtt\\_int](#) object.*
- [mqtt\\_int](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subs...)  
*Constructor for a standalone [mqtt\\_int](#) object.*
- [mqtt\\_int](#) & operator= (int i)  
*Assignment operator for [mqtt\\_int](#). Assigns the value to the object, then performs any post assignment operations.*
- operator int () const  
*Integer conversion operator for [mqtt\\_int](#). Returns the value of the object as an int.*
- [mqtt\\_int](#) & operator++ ()  
*Preincrement the object.*
- int32\_t operator++ (int)  
*Postincrement the object.*
- [mqtt\\_int](#) & operator-- ()  
*Predecrement the object.*
- int32\_t operator-- (int)  
*Postdecrement the object.*
- virtual ValueTestResult [TestOrSetValue](#) ([ParsedJsonDataSet](#) &pjs, bool setVal) override  
*whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.*
- virtual int [SubscribeEvent](#) (int mqttFd, [PacketInfo::Msg](#) \*msg) override  
*Handle a message sent by the Broker triggered by this object's Subscription.*
- virtual int [RenderToBuffer](#) ([eObj\\_Serialize\\_t](#) form, uint8\_t \*buf, int buflen) override  
*Render this object to an output buffer.*
- virtual int [RenderToFd](#) ([eObj\\_Serialize\\_t](#) form, int fd) override  
*Render this object to an output socket.*

Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- ~[mqtt\\_leaf](#) ()  
*~mqtt\_leaf Destructor for base [mqtt\\_leaf](#)*
- void [RemoveFromTree](#) ()  
*Remove this object from it's parent tree.*
- bool [NameMatch](#) (const char \*cp)  
*Determine whether a given name matches this object.*
- virtual int [GetPubTopicName](#) ([NBString](#) &s)  
*Get the Topic Name this object Publishes to.*
- virtual int [GetSubTopicFilter](#) ([NBString](#) &s)  
*Get the Subscription Topic Filter this object Subscribes to.*
- virtual int [GetRenderLength](#) ([eObj\\_Serialize\\_t](#) form=[eObj\\_Serialize\\_CURRENT\\_FORM](#))  
*Get the number of bytes used to serialize this object in a particular format.*
- int [Publish](#) (int waitForCompletion=-1)  
*Publishes the current [MQTT](#) object, optionally waiting for completion.*
- void [SetPublishInterval](#) (uint32\_t intervalPeriodTicks)  
*Sets or clears an automatic publishing interval for this [MQTT](#) object.*
- virtual void [SetSerializationForm](#) ([eObj\\_Serialize\\_t](#) form)  
*Set the default serialization format for this object.*

- [eObj\\_Serialize\\_t GetSerializationForm](#) ()  
*Get the default serialization format for this object.*
- virtual void [SetFlags](#) (uint16\_t newFlags)  
*Set additional behavioral flags for this object.*
- virtual void [ClearFlags](#) (uint16\_t delFlags)  
*Clear behavioral flags from this object.*
- uint16\_t [GetFlags](#) ()  
*Get the behavioral flags set for this object.*
- int [getPriority](#) ()  
*Get the message priority for this object.*

### Protected Member Functions

- virtual int [ParseFd](#) (int mqttFd) override  
*the data from a file descriptor according to this object's current default serialization method and assign the value.*

### Protected Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- [mqtt\\_leaf](#) (const char \*topic, [Client](#) \*mClient, uint16\_t flags=([eObj\\_Flag\\_PubOnWrite](#)|[eObj\\_Flag\\_Subscribe](#)|[eObj\\_Flag\\_Sub\\_N](#))  
*Constructor for [mqtt\\_leaf](#) objects.*
- [mqtt\\_leaf](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient, uint16\_t flags=([eObj\\_Flag\\_PubOnWrite](#)|[eObj\\_Flag\\_Subscribe](#)|[eObj\\_Flag\\_Sub\\_N](#))  
*Constructor for [mqtt\\_leaf](#) objects.*
- virtual [mqtt\\_leaf](#) \* [GetChildren](#) ()  
*Get the first child of this object.*
- virtual [mqtt\\_leaf](#) \* [AddChild](#) ([mqtt\\_leaf](#) \*child)  
*Add an MqttObj as a child to this object.*
- virtual [mqtt\\_leaf](#) \* [RemoveFirstChild](#) ()  
*Remove the first child from this object.*
- void [ForEachLeaf](#) (LeafCallBack \*p, void \*pd)  
*Call a function on each element of the MqttObj tree with this object as the root.*
- void [RootParse](#) ([ParsedJsonDataSet](#) &pjs)  
*Assign to this object tree from a JSON dataset.*
- bool [MatchId](#) ([ParsedJsonDataSet](#) &pjs)  
*Determine whether the JSON dataset's current index matches this object's name.*
- int [QueuePublish](#) ()  
*Queue this object to be Published via the registered [Client](#).*
- int [QueueSubscribe](#) ()  
*Queue this object to be Subscribed via the registered [Client](#).*
- int [QueueUnsubscribe](#) ()  
*Queue this object to be Unsubscribed via the registered [Client](#).*
- int [Assign](#) ()  
*Perform post assignemnt operations.*
- uint8\_t [PubState](#) ()  
*Get the Publish state.*
- uint8\_t [SubState](#) ()  
*Get the Subscribe state of this object.*
- virtual void [SetPubState](#) (uint8\_t newState)  
*Set the Publish state.*
- virtual void [SetSubState](#) (uint8\_t newState)  
*Set the Subscribe state.*
- int [HandleSubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle a Subscribe Acknowledge message from the broker for our Subscribe request.*

- int [HandleUnsubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.*
- int [HandlePubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Publish Acknowledge message from the broker for our Publish request.*

### Additional Inherited Members

### Public Types inherited from [MQTT::mqtt\\_leaf](#)

- enum [eObj\\_State\\_t](#)  
*Mqtt Object Publication/Subscription States.*

### Static Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- static int [MqttRxPublish](#) ([Client](#) \*mClient, int mqttFd, [PacketInfo::Msg](#) \*msg, void \*ctx)  
*Static message handler for Publish messages received by the registered [Client](#).*

### Protected Attributes inherited from [MQTT::mqtt\\_leaf](#)

- [NBInterpolatedString](#) **topicName**  
*Topic name or additional branch.*
- [eObj\\_Serialize\\_t](#) **publishForm**  
*Default serialization form.*
- uint16\_t **mFlags**  
*Behavioral flags.*

## 16.127.1 Detailed Description

Easy MqttObj class for automatically publishing and subscribing unsigned integer values to/from an [MQTT](#) topic. Use like a normal int32\_t.

## 16.127.2 Constructor & Destructor Documentation

### 16.127.2.1 mqtt\_int() [1/4]

```
MQTT::mqtt_int::mqtt_int (
 int val,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a standalone [mqtt\\_int](#) object.

#### Parameters

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>val</i>      | Initial value for the integer                                                                                                      |
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>    | Behavioral flags to set on the created object                                                                                      |

### 16.127.2.2 mqtt\_int() [2/4]

```
MQTT::mqtt_int::mqtt_int (
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a standalone [mqtt\\_int](#) object.

## Parameters

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>    | Behavioral flags to set on the created object                                                                                      |

**16.127.2.3 mqtt\_int() [3/4]**

```
MQTT::mqtt_int::mqtt_int (
 mqtt_leaf & parent,
 int val,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a standalone [mqtt\\_int](#) object.

## Parameters

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>val</i>         | Initial value for the integer                                                                                                      |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i>    | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

**16.127.2.4 mqtt\_int() [4/4]**

```
MQTT::mqtt_int::mqtt_int (
 mqtt_leaf & parent,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a standalone [mqtt\\_int](#) object.

## Parameters

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i>    | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

**16.127.3 Member Function Documentation****16.127.3.1 operator int()**

```
MQTT::mqtt_int::operator int () const [inline]
```

Integer conversion operator for [mqtt\\_int](#). Returns the value of the object as an int.

## Returns

The integer value of the object

**16.127.3.2 operator++()** [1/2]

`mqtt_int & MQTT::mqtt_int::operator++ () [inline]`  
 Preincrement the object.

**Returns**

Reference to the object

**16.127.3.3 operator++()** [2/2]

`int32_t MQTT::mqtt_int::operator++ (int ) [inline]`  
 Postincrement the object.

**Returns**

The integer value before being incremented

**16.127.3.4 operator--()** [1/2]

`mqtt_int & MQTT::mqtt_int::operator-- () [inline]`  
 Predecrement the object.

**Returns**

Reference to the object

**16.127.3.5 operator--()** [2/2]

`int32_t MQTT::mqtt_int::operator-- (int ) [inline]`  
 Postdecrement the object.

**Returns**

The integer value before being decremented

**16.127.3.6 operator=()**

`mqtt_int & MQTT::mqtt_int::operator= (int i)`

Assignment operator for `mqtt_int`. Assigns the value to the object, then performs any post assignment operations.

**Parameters**

|          |                                    |
|----------|------------------------------------|
| <i>i</i> | The value to assign to the object. |
|----------|------------------------------------|

**Returns**

A reference to the object being assigned to.

**16.127.3.7 ParseFd()**

`virtual int MQTT::mqtt_int::ParseFd (int mqttFd) [override], [protected], [virtual]`

the data from a file descriptor according to this object's current default serialization method and assign the value.

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>mqttFd</i> | File descriptor to read from. |
|---------------|-------------------------------|

## Returns

`eResult_t` return code

Implements [MQTT::mqtt\\_leaf](#).

**16.127.3.8 RenderToBuffer()**

```
virtual int MQTT::mqtt_int::RenderToBuffer (
 eObj_Serialize_t form,
 uint8_t * buf,
 int buflen) [override], [virtual]
```

Render this object to an output buffer.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>form</i>   | Serialization format to render as |
| <i>*buf</i>   | Pointer to output buffer          |
| <i>buflen</i> | Size of the output buffer         |

## Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

**16.127.3.9 RenderToFd()**

```
virtual int MQTT::mqtt_int::RenderToFd (
 eObj_Serialize_t form,
 int fd) [override], [virtual]
```

Render this object to an output socket.

## Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>form</i> | Serialization format to render as |
| <i>fd</i>   | File descriptor to render to      |

## Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

**16.127.3.10 SubscribeEvent()**

```
virtual int MQTT::mqtt_int::SubscribeEvent (
 int mqttFd,
 PacketInfo::Msg * msg) [override], [virtual]
```

Handle a message sent by the Broker triggered by this object's Subscription.

## Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>mqttFd</i> | File Descriptor to read any additional message data from |
| <i>*msg</i>   | Pointer to message received from the Broker              |



## Returns

eResult\_t return code

Reimplemented from [MQTT::mqtt\\_leaf](#).

## 16.127.3.11 TestOrSetValue()

```
virtual ValueTestResult MQTT::mqtt_int::TestOrSetValue (
 ParsedJsonDataSet & pjs,
 bool setVal) [override], [virtual]
```

whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.

## Parameters

|                 |                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------|
| <i>&amp;pjs</i> | The JSON dataset to use                                                                                 |
| <i>setVal</i>   | true: Assign from the value at the current index, false: test if current index is valid for this object |

## Returns

ValueTestResult indicating the validity of the current index

Implements [MQTT::mqtt\\_leaf](#).

The documentation for this class was generated from the following file:

- mqtt.h

## 16.128 MQTT::mqtt\_IO Class Reference

Easy MqttObj class for automatically publishing and subscribing hardware GPIO pins to/from an [MQTT](#) topic. Use like a normal [PinIO](#).

```
#include <mqtt.h>
```

Inherits [MQTT::mqtt\\_leaf](#).

Inherited by [MQTT::mqtt\\_IO\\_Input](#), and [MQTT::mqtt\\_IO\\_Output](#).

## Public Member Functions

- [mqtt\\_IO](#) (PinIO ioPin, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_...)  
*Constructor for a standalone mqtt\_IO object.*
- [mqtt\\_IO](#) ([mqtt\\_leaf](#) &parent, PinIO ioPin, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_...)  
*Constructor for a mqtt\_IO object that has a parent.*
- virtual void [SetFlags](#) (uint16\_t newFlags) override  
*additional behavioral flags for this object*
- virtual void [ClearFlags](#) (uint16\_t delFlags) override  
*behavioral flags from this object*
- int [SetFixedDirection](#) (int direction)  
*Set a fixed direction for the GPIO pin.*
- [mqtt\\_IO](#) & [operator=](#) (bool val)  
*Assignment operator for mqtt\_IO. Assigns the value to the hardware GPIO, then performs any post assignment operations.*
- [operator bool](#) ()  
*Boolean conversion operator for mqtt\_IO. Returns the current state of the underlying GPIO.*
- bool [operator!](#) ()  
*Boolean Not operator for mqtt\_IO. Returns the negation of the current state of the underlying GPIO.*
- virtual ValueTestResult [TestOrSetValue](#) ([ParsedJsonDataSet](#) &pjs, bool setVal) override

*whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.*

- virtual int [SubscribeEvent](#) (int mqttFd, [PacketInfo::Msg](#) \*msg) override  
*Handle a message sent by the Broker triggered by this object's Subscription.*
- virtual int [RenderToBuffer](#) ([eObj\\_Serialize\\_t](#) form, uint8\_t \*buf, int buflen) override  
*Render this object to an output buffer.*
- virtual int [RenderToFd](#) ([eObj\\_Serialize\\_t](#) form, int fd) override  
*Render this object to an output socket.*

## Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- [~mqtt\\_leaf](#) ()  
*~mqtt\_leaf Destructor for base [mqtt\\_leaf](#)*
- void [RemoveFromTree](#) ()  
*Remove this object from it's parent tree.*
- bool [NameMatch](#) (const char \*cp)  
*Determine whether a given name matches this object.*
- virtual int [GetPubTopicName](#) ([NBString](#) &s)  
*Get the Topic Name this object Publishes to.*
- virtual int [GetSubTopicFilter](#) ([NBString](#) &s)  
*Get the Subscription Topic Filter this object Subscribes to.*
- virtual int [GetRenderLength](#) ([eObj\\_Serialize\\_t](#) form=[eObj\\_Serialize\\_CURRENT\\_FORM](#))  
*Get the number of bytes used to serialize this object in a particular format.*
- int [Publish](#) (int waitForCompletion=-1)  
*Publishes the current [MQTT](#) object, optionally waiting for completion.*
- void [SetPublishInterval](#) (uint32\_t intervalPeriodTicks)  
*Sets or clears an automatic publishing interval for this [MQTT](#) object.*
- virtual void [SetSerializationForm](#) ([eObj\\_Serialize\\_t](#) form)  
*Set the default serialization format for this object.*
- [eObj\\_Serialize\\_t](#) [GetSerializationForm](#) ()  
*Get the default serialization format for this object.*
- uint16\_t [GetFlags](#) ()  
*Get the behavioral flags set for this object.*
- int [getPriority](#) ()  
*Get the message priority for this object.*

## Static Public Member Functions

- static void [PinNotify](#) (int port\_num, int pin, void \*ctx)  
*Static notification function for pin state changes.*

## Static Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- static int [MqttRxPublish](#) ([Client](#) \*mClient, int mqttFd, [PacketInfo::Msg](#) \*msg, void \*ctx)  
*Static message handler for Publish messages received by the registered [Client](#).*

## Protected Member Functions

- virtual int [ParseFd](#) (int mqttFd) override  
*the data from a file descriptor according to this object's current default serialization method and assign the value.*

## Protected Member Functions inherited from MQTT::mqtt\_leaf

- [mqtt\\_leaf](#) (const char \*topic, [Client](#) \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_N  
*Constructor for mqtt\_leaf objects.*
- [mqtt\\_leaf](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|  
*Constructor for mqtt\_leaf objects.*
- virtual [mqtt\\_leaf](#) \* [GetChildren](#) ()  
*Get the first child of this object.*
- virtual [mqtt\\_leaf](#) \* [AddChild](#) ([mqtt\\_leaf](#) \*child)  
*Add an MqttObj as a child to this object.*
- virtual [mqtt\\_leaf](#) \* [RemoveFirstChild](#) ()  
*Remove the first child from this object.*
- void [ForEachLeaf](#) (LeafCallBack \*p, void \*pd)  
*Call a function on each element of the MqttObj tree with this object as the root.*
- void [RootParse](#) ([ParsedJsonDataSet](#) &pjs)  
*Assign to this object tree from a JSON dataset.*
- bool [MatchId](#) ([ParsedJsonDataSet](#) &pjs)  
*Determine whether the JSON dataset's current index matches this object's name.*
- int [QueuePublish](#) ()  
*Queue this object to be Published via the registered Client.*
- int [QueueSubscribe](#) ()  
*Queue this object to be Subscribed via the registered Client.*
- int [QueueUnsubscribe](#) ()  
*Queue this object to be Unsubscribed via the registered Client.*
- int [Assign](#) ()  
*Perform post assignemnt operations.*
- uint8\_t [PubState](#) ()  
*Get the Publish state.*
- uint8\_t [SubState](#) ()  
*Get the Subscribe state of this object.*
- virtual void [SetPubState](#) (uint8\_t newState)  
*Set the Publish state.*
- virtual void [SetSubState](#) (uint8\_t newState)  
*Set the Subscribe state.*
- int [HandleSubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle a Subscribe Acknowledge message from the broker for our Subscribe request.*
- int [HandleUnsubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.*
- int [HandlePubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle a Publish Acknowledge message from the broker for our Publish request.*

## Additional Inherited Members

## Public Types inherited from MQTT::mqtt\_leaf

- enum [eObj\\_State\\_t](#)  
*Mqtt Object Publication/Subscription States.*

## Protected Attributes inherited from [MQTT::mqtt\\_leaf](#)

- [NBInterpolatedString](#) **topicName**  
*Topic name or additional branch.*
- [eObj\\_Serialize\\_t](#) **publishForm**  
*Default serialization form.*
- [uint16\\_t](#) **mFlags**  
*Behavioral flags.*

### 16.128.1 Detailed Description

Easy MqttObj class for automatically publishing and subscribing hardware GPIO pins to/from an [MQTT](#) topic. Use like a normal [PinIO](#).

### 16.128.2 Constructor & Destructor Documentation

#### 16.128.2.1 mqtt\_IO() [1/2]

```
MQTT::mqtt_IO::mqtt_IO (
 PinIO ioPin,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a standalone [mqtt\\_IO](#) object.

##### Parameters

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>ioPin</i>    | GPIO pin to control/report                                                                                                         |
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>    | Behavioral flags to set on the created object                                                                                      |

#### 16.128.2.2 mqtt\_IO() [2/2]

```
MQTT::mqtt_IO::mqtt_IO (
 mqtt_leaf & parent,
 PinIO ioPin,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a [mqtt\\_IO](#) object that has a parent.

##### Parameters

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>ioPin</i>       | GPIO pin to control/report                                                                                                         |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i>    | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

## 16.128.3 Member Function Documentation

### 16.128.3.1 ClearFlags()

```
virtual void MQTT::mqtt_IO::ClearFlags (
 uint16_t delFlags) [override], [virtual]
```

behavioral flags from this object

#### Parameters

|                 |                                          |
|-----------------|------------------------------------------|
| <i>delFlags</i> | The flags to be cleared from this object |
|-----------------|------------------------------------------|

Reimplemented from [MQTT::mqtt\\_leaf](#).

### 16.128.3.2 operator bool()

```
MQTT::mqtt_IO::operator bool () [inline]
```

Boolean conversion operator for [mqtt\\_IO](#). Returns the current state of the underlying GPIO.

#### Returns

The state of the underlying GPIO

### 16.128.3.3 operator"!()

```
bool MQTT::mqtt_IO::operator! () [inline]
```

Boolean Not operator for [mqtt\\_IO](#). Returns the negation of the current state of the underlying GPIO.

#### Returns

The negation of the current state of the underlying GPIO

### 16.128.3.4 operator=()

```
mqtt_IO & MQTT::mqtt_IO::operator= (
 bool val)
```

Assignment operator for [mqtt\\_IO](#). Assigns the value to the hardware GPIO, then performs any post assignment operations.

#### Parameters

|            |                                 |
|------------|---------------------------------|
| <i>val</i> | The value to assign to the pin. |
|------------|---------------------------------|

#### Returns

A reference to the object being assigned to.

### 16.128.3.5 ParseFd()

```
virtual int MQTT::mqtt_IO::ParseFd (
 int mqttFd) [override], [protected], [virtual]
```

the data from a file descriptor according to this object's current default serialization method and assign the value.

#### Parameters

|               |                               |
|---------------|-------------------------------|
| <i>mqttFd</i> | File descriptor to read from. |
|---------------|-------------------------------|

#### Returns

eResult\_t return code

Implements [MQTT::mqtt\\_leaf](#).

### 16.128.3.6 PinNotify()

```
static void MQTT::mqtt_IO::PinNotify (
 int port_num,
 int pin,
 void * ctx) [static]
```

Static notification function for pin state changes.

#### Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>port_num</i> | The GPIO port number that triggered this call.          |
| <i>pin</i>      | The GPIO pin index of the pin that triggered this call. |
| <i>ctx</i>      | The notification context associated with this call.     |

### 16.128.3.7 RenderToBuffer()

```
virtual int MQTT::mqtt_IO::RenderToBuffer (
 eObj_Serialize_t form,
 uint8_t * buf,
 int buflen) [override], [virtual]
```

Render this object to an output buffer.

#### Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>form</i>   | Serialization format to render as |
| <i>*buf</i>   | Pointer to output buffer          |
| <i>buflen</i> | Size of the output buffer         |

#### Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

### 16.128.3.8 RenderToFd()

```
virtual int MQTT::mqtt_IO::RenderToFd (
 eObj_Serialize_t form,
 int fd) [override], [virtual]
```

Render this object to an output socket.

#### Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>form</i> | Serialization format to render as |
| <i>fd</i>   | File descriptor to render to      |

#### Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

### 16.128.3.9 SetFixedDirection()

```
int MQTT::mqtt_IO::SetFixedDirection (
 int direction)
```

Set a fixed direction for the GPIO pin.

## Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>direction</i> | The direction to point: <0: input, 0: unfixed, >0: output |
|------------------|-----------------------------------------------------------|

## Returns

The direction the GPIO is fixed to: <0: input, 0: unfixed, >0: output

**16.128.3.10 SetFlags()**

```
virtual void MQTT::mqtt_IO::SetFlags (
 uint16_t newFlags) [override], [virtual]
```

additional behavioral flags for this object

## Parameters

|                 |                                |
|-----------------|--------------------------------|
| <i>newFlags</i> | The set of new flags to be set |
|-----------------|--------------------------------|

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.128.3.11 SubscribeEvent()**

```
virtual int MQTT::mqtt_IO::SubscribeEvent (
 int mqttFd,
 PacketInfo::Msg * msg) [override], [virtual]
```

Handle a message sent by the Broker triggered by this object's Subscription.

## Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>mqttFd</i> | File Descriptor to read any additional message data from |
| <i>*msg</i>   | Pointer to message received from the Broker              |

## Returns

eResult\_t return code

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.128.3.12 TestOrSetValue()**

```
virtual ValueTestResult MQTT::mqtt_IO::TestOrSetValue (
 ParsedJsonDataSet & pjs,
 bool setVal) [override], [virtual]
```

whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.

## Parameters

|                 |                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------|
| <i>&amp;pjs</i> | The JSON dataset to use                                                                                 |
| <i>setVal</i>   | true: Assign from the value at the current index, false: test if current index is valid for this object |

## Returns

ValueTestResult indicating the validity of the current index

Implements [MQTT::mqtt\\_leaf](#).

The documentation for this class was generated from the following file:

- mqtt.h

## 16.129 MQTT::mqtt\_IO\_Input Class Reference

MqttObj connecting hardware GPIO pins to an Mqtt Broker. This class defaults the pin to an Input state.

#include <mqtt.h>

Inherits [MQTT::mqtt\\_IO](#).

### Public Member Functions

- [mqtt\\_IO\\_Input](#) (PinIO ioPin, const char \*topic, [Client](#) \*client=NULL, uint16\_t flags=0)  
*Constructor for a standalone [mqtt\\_IO\\_Input](#) object.*
- [mqtt\\_IO\\_Input](#) ([mqtt\\_leaf](#) &parent, PinIO ioPin, const char \*topic, [Client](#) \*client=NULL, uint16\_t flags=0)  
*Constructor for a [mqtt\\_IO\\_Input](#) object that has a parent.*
- [mqtt\\_IO](#) & [operator=](#) (bool val)  
*Assignment operator for [mqtt\\_IO](#). Assigns the value to the hardware GPIO, then performs any post assignment operations.*

### Public Member Functions inherited from [MQTT::mqtt\\_IO](#)

- [mqtt\\_IO](#) (PinIO ioPin, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=([eObj\\_Flag\\_PubOnWrite](#)|[eObj\\_Flag\\_Subscribe](#)))  
*Constructor for a standalone [mqtt\\_IO](#) object.*
- [mqtt\\_IO](#) ([mqtt\\_leaf](#) &parent, PinIO ioPin, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=([eObj\\_Flag\\_PubOnWrite](#)|[eObj\\_Flag\\_Subscribe](#)))  
*Constructor for a [mqtt\\_IO](#) object that has a parent.*
- virtual void [SetFlags](#) (uint16\_t newFlags) override  
*additional behavioral flags for this object*
- virtual void [ClearFlags](#) (uint16\_t delFlags) override  
*behavioral flags from this object*
- int [SetFixedDirection](#) (int direction)  
*Set a fixed direction for the GPIO pin.*
- [mqtt\\_IO](#) & [operator=](#) (bool val)  
*Assignment operator for [mqtt\\_IO](#). Assigns the value to the hardware GPIO, then performs any post assignment operations.*
- [operator bool](#) ()  
*Boolean conversion operator for [mqtt\\_IO](#). Returns the current state of the underlying GPIO.*
- bool [operator!](#) ()  
*Boolean Not operator for [mqtt\\_IO](#). Returns the negation of the current state of the underlying GPIO.*
- virtual ValueTestResult [TestOrSetValue](#) ([ParsedJsonDataSet](#) &pjs, bool setVal) override  
*whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.*
- virtual int [SubscribeEvent](#) (int mqttFd, [PacketInfo::Msg](#) \*msg) override  
*Handle a message sent by the Broker triggered by this object's Subscription.*
- virtual int [RenderToBuffer](#) ([eObj\\_Serialize\\_t](#) form, uint8\_t \*buf, int buflen) override  
*Render this object to an output buffer.*
- virtual int [RenderToFd](#) ([eObj\\_Serialize\\_t](#) form, int fd) override  
*Render this object to an output socket.*

### Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- [~mqtt\\_leaf](#) ()  
*~mqtt\_leaf Destructor for base [mqtt\\_leaf](#)*
- void [RemoveFromTree](#) ()  
*Remove this object from it's parent tree.*
- bool [NameMatch](#) (const char \*cp)  
*Determine whether a given name matches this object.*



- virtual int [GetPubTopicName](#) (NBString &s)  
*Get the Topic Name this object Publishes to.*
- virtual int [GetSubTopicFilter](#) (NBString &s)  
*Get the Subscription Topic Filter this object Subscribes to.*
- virtual int [GetRenderLength](#) (eObj\_Serialize\_t form=eObj\_Serialize\_CURRENT\_FORM)  
*Get the number of bytes used to serialize this object in a particular format.*
- int [Publish](#) (int waitForCompletion=-1)  
*Publishes the current MQTT object, optionally waiting for completion.*
- void [SetPublishInterval](#) (uint32\_t intervalPeriodTicks)  
*Sets or clears an automatic publishing interval for this MQTT object.*
- virtual void [SetSerializationForm](#) (eObj\_Serialize\_t form)  
*Set the default serialization format for this object.*
- eObj\_Serialize\_t [GetSerializationForm](#) ()  
*Get the default serialization format for this object.*
- uint16\_t [GetFlags](#) ()  
*Get the behavioral flags set for this object.*
- int [getPriority](#) ()  
*Get the message priority for this object.*

#### Additional Inherited Members

#### Public Types inherited from MQTT::mqtt\_leaf

- enum [eObj\\_State\\_t](#)  
*Mqtt Object Publication/Subscription States.*

#### Static Public Member Functions inherited from MQTT::mqtt\_IO

- static void [PinNotify](#) (int port\_num, int pin, void \*ctx)  
*Static notification function for pin state changes.*

#### Static Public Member Functions inherited from MQTT::mqtt\_leaf

- static int [MqttRxPublish](#) (Client \*mClient, int mqttFd, PacketInfo::Msg \*msg, void \*ctx)  
*Static message handler for Publish messages received by the registered Client.*

#### Protected Member Functions inherited from MQTT::mqtt\_IO

- virtual int [ParseFd](#) (int mqttFd) override  
*the data from a file descriptor according to this object's current default serialization method and assign the value.*

#### Protected Member Functions inherited from MQTT::mqtt\_leaf

- [mqtt\\_leaf](#) (const char \*topic, Client \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_N...)  
*Constructor for mqtt\_leaf objects.*
- [mqtt\\_leaf](#) (mqtt\_leaf &parent, const char \*topic, Client \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|...)  
*Constructor for mqtt\_leaf objects.*
- virtual [mqtt\\_leaf](#) \* [GetChildren](#) ()  
*Get the first child of this object.*
- virtual [mqtt\\_leaf](#) \* [AddChild](#) (mqtt\_leaf \*child)  
*Add an MqttObj as a child to this object.*
- virtual [mqtt\\_leaf](#) \* [RemoveFirstChild](#) ()  
*Remove the first child from this object.*

- void [ForEachLeaf](#) (LeafCallBack \*p, void \*pd)  
*Call a function on each element of the MqttObj tree with this object as the root.*
- void [RootParse](#) (ParsedJsonDataSet &pjs)  
*Assign to this object tree from a JSON dataset.*
- bool [MatchId](#) (ParsedJsonDataSet &pjs)  
*Determine whether the JSON dataset's current index matches this object's name.*
- int [QueuePublish](#) ()  
*Queue this object to be Published via the registered [Client](#).*
- int [QueueSubscribe](#) ()  
*Queue this object to be Subscribed via the registered [Client](#).*
- int [QueueUnsubscribe](#) ()  
*Queue this object to be Unsubscribed via the registered [Client](#).*
- int [Assign](#) ()  
*Perform post assignemnt operations.*
- uint8\_t [PubState](#) ()  
*Get the Publish state.*
- uint8\_t [SubState](#) ()  
*Get the Subscribe state of this object.*
- virtual void [SetPubState](#) (uint8\_t newState)  
*Set the Publish state.*
- virtual void [SetSubState](#) (uint8\_t newState)  
*Set the Subscribe state.*
- int [HandleSubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle a Subscribe Acknowledge message from the broker for our Subscribe request.*
- int [HandleUnsubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.*
- int [HandlePubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Publish Acknowledge message from the broker for our Publish request.*

## Protected Attributes inherited from [MQTT::mqtt\\_leaf](#)

- [NBInterpolatedString](#) **topicName**  
*Topic name or additional branch.*
- [eObj\\_Serialize\\_t](#) **publishForm**  
*Default serialization form.*
- uint16\_t **mFlags**  
*Behavioral flags.*

### 16.129.1 Detailed Description

MqttObj connecting hardware GPIO pins to an Mqtt Broker. This class defaults the pin to an Input state.

### 16.129.2 Constructor & Destructor Documentation

#### 16.129.2.1 mqtt\_IO\_Input() [1/2]

```
MQTT::mqtt_IO_Input::mqtt_IO_Input (
 PinIO ioPin,
 const char * topic,
 Client * client = NULL,
 uint16_t flags = 0) [inline]
```

Constructor for a standalone [mqtt\\_IO\\_Input](#) object.

## Parameters

|                |                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>ioPin</i>   | GPIO pin to control/report                                                                                                         |
| <i>*topic</i>  | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*client</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>   | Behavioral flags to set on the created object                                                                                      |

## 16.129.2.2 mqtt\_IO\_Input() [2/2]

```
MQTT::mqtt_IO_Input::mqtt_IO_Input (
 mqtt_leaf & parent,
 PinIO ioPin,
 const char * topic,
 Client * client = NULL,
 uint16_t flags = 0) [inline]
```

Constructor for a [mqtt\\_IO\\_Input](#) object that has a parent.

## Parameters

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>ioPin</i>       | GPIO pin to control/report                                                                                                         |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*client</i>     | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

## 16.129.3 Member Function Documentation

## 16.129.3.1 operator=()

```
mqtt_IO & MQTT::mqtt_IO::operator= (
 bool val)
```

Assignment operator for [mqtt\\_IO](#). Assigns the value to the hardware GPIO, then performs any post assignment operations.

## Parameters

|            |                                 |
|------------|---------------------------------|
| <i>val</i> | The value to assign to the pin. |
|------------|---------------------------------|

## Returns

A reference to the object being assigned to.

The documentation for this class was generated from the following file:

- mqtt.h

## 16.130 MQTT::mqtt\_IO\_Output Class Reference

MqttObj connecting hardware GPIO pins to an Mqtt Broker. This class defaults the pin to an Output state.

```
#include <mqtt.h>
```

Inherits [MQTT::mqtt\\_IO](#).

## Public Member Functions

- `mqtt_IO_Output` (`PinIO` ioPin, `const char *`topic, `Client *`client=NULL, `uint16_t` flags=0)  
*Constructor for a standalone `mqtt_IO_Output` object.*
- `mqtt_IO_Output` (`mqtt_Leaf` &parent, `PinIO` ioPin, `const char *`topic, `Client *`client=NULL, `uint16_t` flags=0)  
*Constructor for a `mqtt_IO_Output` object that has a parent.*
- `mqtt_IO` & `operator=` (`bool` val)  
*Assignment operator for `mqtt_IO`. Assigns the value to the hardware GPIO, then performs any post assignment operations.*

## Public Member Functions inherited from `MQTT::mqtt_IO`

- `mqtt_IO` (`PinIO` ioPin, `const char *`topic, `Client *`mClient=NULL, `uint16_t` flags=(`eObj_Flag_PubOnWrite`|`eObj_Flag_Subscribe`))  
*Constructor for a standalone `mqtt_IO` object.*
- `mqtt_IO` (`mqtt_Leaf` &parent, `PinIO` ioPin, `const char *`topic, `Client *`mClient=NULL, `uint16_t` flags=(`eObj_Flag_PubOnWrite`|`eObj_Flag_Subscribe`))  
*Constructor for a `mqtt_IO` object that has a parent.*
- virtual `void` `SetFlags` (`uint16_t` newFlags) override  
*additional behavioral flags for this object*
- virtual `void` `ClearFlags` (`uint16_t` delFlags) override  
*behavioral flags from this object*
- `int` `SetFixedDirection` (`int` direction)  
*Set a fixed direction for the GPIO pin.*
- `mqtt_IO` & `operator=` (`bool` val)  
*Assignment operator for `mqtt_IO`. Assigns the value to the hardware GPIO, then performs any post assignment operations.*
- `operator bool` ()  
*Boolean conversion operator for `mqtt_IO`. Returns the current state of the underlying GPIO.*
- `bool` `operator!` ()  
*Boolean Not operator for `mqtt_IO`. Returns the negation of the current state of the underlying GPIO.*
- virtual `ValueTestResult` `TestOrSetValue` (`ParsedJsonDataSet` &pjs, `bool` setVal) override  
*whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.*
- virtual `int` `SubscribeEvent` (`int` mqttFd, `PacketInfo::Msg *`msg) override  
*Handle a message sent by the Broker triggered by this object's Subscription.*
- virtual `int` `RenderToBuffer` (`eObj_Serialize_t` form, `uint8_t *`buf, `int` buflen) override  
*Render this object to an output buffer.*
- virtual `int` `RenderToFd` (`eObj_Serialize_t` form, `int` fd) override  
*Render this object to an output socket.*

## Public Member Functions inherited from `MQTT::mqtt_Leaf`

- `~mqtt_Leaf` ()  
*~mqtt\_Leaf Destructor for base `mqtt_Leaf`*
- `void` `RemoveFromTree` ()  
*Remove this object from it's parent tree.*
- `bool` `NameMatch` (`const char *`cp)  
*Determine whether a given name matches this object.*
- virtual `int` `GetPubTopicName` (`NBString` &s)  
*Get the Topic Name this object Publishes to.*
- virtual `int` `GetSubTopicFilter` (`NBString` &s)  
*Get the Subscription Topic Filter this object Subscribes to.*
- virtual `int` `GetRenderLength` (`eObj_Serialize_t` form=`eObj_Serialize_CURRENT_FORM`)  
*Get the number of bytes used to serialize this object in a particular format.*

- int [Publish](#) (int waitForCompletion=-1)  
*Publishes the current [MQTT](#) object, optionally waiting for completion.*
- void [SetPublishInterval](#) (uint32\_t intervalPeriodTicks)  
*Sets or clears an automatic publishing interval for this [MQTT](#) object.*
- virtual void [SetSerializationForm](#) (eObj\_Serialize\_t form)  
*Set the default serialization format for this object.*
- [eObj\\_Serialize\\_t](#) [GetSerializationForm](#) ()  
*Get the default serialization format for this object.*
- uint16\_t [GetFlags](#) ()  
*Get the behavioral flags set for this object.*
- int [getPriority](#) ()  
*Get the message priority for this object.*

### Additional Inherited Members

### Public Types inherited from [MQTT::mqtt\\_leaf](#)

- enum [eObj\\_State\\_t](#)  
*Mqtt Object Publication/Subscription States.*

### Static Public Member Functions inherited from [MQTT::mqtt\\_IO](#)

- static void [PinNotify](#) (int port\_num, int pin, void \*ctx)  
*Static notification function for pin state changes.*

### Static Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- static int [MqttRxPublish](#) ([Client](#) \*mClient, int mqttFd, [PacketInfo::Msg](#) \*msg, void \*ctx)  
*Static message handler for Publish messages received by the registered [Client](#).*

### Protected Member Functions inherited from [MQTT::mqtt\\_IO](#)

- virtual int [ParseFd](#) (int mqttFd) override  
*the data from a file descriptor according to this object's current default serialization method and assign the value.*

### Protected Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- [mqtt\\_leaf](#) (const char \*topic, [Client](#) \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_N  
*Constructor for [mqtt\\_leaf](#) objects.*
- [mqtt\\_leaf](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|  
*Constructor for [mqtt\\_leaf](#) objects.*
- virtual [mqtt\\_leaf](#) \* [GetChildren](#) ()  
*Get the first child of this object.*
- virtual [mqtt\\_leaf](#) \* [AddChild](#) ([mqtt\\_leaf](#) \*child)  
*Add an [MqttObj](#) as a child to this object.*
- virtual [mqtt\\_leaf](#) \* [RemoveFirstChild](#) ()  
*Remove the first child from this object.*
- void [ForEachLeaf](#) (LeafCallBack \*p, void \*pd)  
*Call a function on each element of the [MqttObj](#) tree with this object as the root.*
- void [RootParse](#) ([ParsedJsonDataSet](#) &pjs)  
*Assign to this object tree from a JSON dataset.*
- bool [MatchId](#) ([ParsedJsonDataSet](#) &pjs)  
*Determine whether the JSON dataset's current index matches this object's name.*

- `int QueuePublish ()`  
*Queue this object to be Published via the registered [Client](#).*
- `int QueueSubscribe ()`  
*Queue this object to be Subscribed via the registered [Client](#).*
- `int QueueUnsubscribe ()`  
*Queue this object to be Unsubscribed via the registered [Client](#).*
- `int Assign ()`  
*Perform post assignemnt operations.*
- `uint8_t PubState ()`  
*Get the Publish state.*
- `uint8_t SubState ()`  
*Get the Subscribe state of this object.*
- `virtual void SetPubState (uint8_t newState)`  
*Set the Publish state.*
- `virtual void SetSubState (uint8_t newState)`  
*Set the Subscribe state.*
- `int HandleSubAck (int mqttFd, PacketInfo::Msg *msg)`  
*Handle a Subscribe Acknowledge message from the broker for our Subscribe request.*
- `int HandleUnsubAck (int mqttFd, PacketInfo::Msg *msg)`  
*Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.*
- `int HandlePubAck (int mqttFd, PacketInfo::Msg *msg)`  
*Handle an Publish Acknowledge message from the broker for our Publish request.*

### Protected Attributes inherited from [MQTT::mqtt\\_leaf](#)

- [NBInterpolatedString](#) **topicName**  
*Topic name or additional branch.*
- [eObj\\_Serialize\\_t](#) **publishForm**  
*Default serialization form.*
- `uint16_t mFlags`  
*Behavioral flags.*

## 16.130.1 Detailed Description

MqttObj connecting hardware GPIO pins to an Mqtt Broker. This class defaults the pin to an Output state.

## 16.130.2 Constructor & Destructor Documentation

### 16.130.2.1 mqtt\_IO\_Output() [1/2]

```
MQTT::mqtt_IO_Output::mqtt_IO_Output (
 PinIO ioPin,
 const char * topic,
 Client * client = NULL,
 uint16_t flags = 0) [inline]
```

Constructor for a standalone [mqtt\\_IO\\_Output](#) object.

#### Parameters

|                |                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>ioPin</i>   | GPIO pin to control/report                                                                                                         |
| <i>*topic</i>  | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*client</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>   | Behavioral flags to set on the created object                                                                                      |

### 16.130.2.2 mqtt\_IO\_Output() [2/2]

```
MQTT::mqtt_IO_Output::mqtt_IO_Output (
 mqtt_leaf & parent,
 PinIO ioPin,
 const char * topic,
 Client * client = NULL,
 uint16_t flags = 0) [inline]
```

Constructor for a [mqtt\\_IO\\_Output](#) object that has a parent.

#### Parameters

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>ioPin</i>       | GPIO pin to control/report                                                                                                         |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*client</i>     | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

## 16.130.3 Member Function Documentation

### 16.130.3.1 operator=()

```
mqtt_IO & MQTT::mqtt_IO::operator= (
 bool val)
```

Assignment operator for [mqtt\\_IO](#). Assigns the value to the hardware GPIO, then performs any post assignment operations.

#### Parameters

|            |                                 |
|------------|---------------------------------|
| <i>val</i> | The value to assign to the pin. |
|------------|---------------------------------|

#### Returns

A reference to the object being assigned to.

The documentation for this class was generated from the following file:

- [mqtt.h](#)

## 16.131 MQTT::mqtt\_leaf Class Reference

Base class for all Mqtt Objects (MqttObj). Mqtt Objects provide a seamless bridge/synchronization to communicate information through an [MQTT](#) Broker via a [Client](#) object.

#include <mqtt.h>

Inherits [TimeOutElement](#).

Inherited by [MQTT::ConfigExposer](#), [MQTT::SubHandler](#), [MQTT::mqtt\\_IO](#), [MQTT::mqtt\\_bool](#), [MQTT::mqtt\\_float](#), [MQTT::mqtt\\_int](#), [MQTT::mqtt\\_obj](#), [MQTT::mqtt\\_string](#), and [MQTT::mqtt\\_uint](#).

#### Public Types

- enum [eObj\\_State\\_t](#)  
*Mqtt Object Publication/Subscription States.*

## Public Member Functions

- `~mqtt_leaf ()`  
*~mqtt\_leaf Destructor for base [mqtt\\_leaf](#)*
- `void RemoveFromTree ()`  
*Remove this object from it's parent tree.*
- `bool NameMatch (const char *cp)`  
*Determine whether a given name matches this object.*
- `virtual ValueTestResult TestOrSetValue (ParsedJsonDataSet &pjs, bool setVal)=0`  
*Test whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.*
- `virtual int GetPubTopicName (NBString &s)`  
*Get the Topic Name this object Publishes to.*
- `virtual int GetSubTopicFilter (NBString &s)`  
*Get the Subscription Topic Filter this object Subscribes to.*
- `virtual int SubscribeEvent (int mqttFd, PacketInfo::Msg *msg)`  
*Handle a message sent by the Broker triggered by this object's Subscription.*
- `virtual int GetRenderLength (eObj_Serialize_t form=eObj_Serialize_CURRENT_FORM)`  
*Get the number of bytes used to serialize this object in a particular format.*
- `int Publish (int waitForCompletion=-1)`  
*Publishes the current [MQTT](#) object, optionally waiting for completion.*
- `void SetPublishInterval (uint32_t intervalPeriodTicks)`  
*Sets or clears an automatic publishing interval for this [MQTT](#) object.*
- `virtual void SetSerializationForm (eObj_Serialize_t form)`  
*Set the default serialization format for this object.*
- `eObj_Serialize_t GetSerializationForm ()`  
*Get the default serialization format for this object.*
- `virtual void SetFlags (uint16_t newFlags)`  
*Set additional behavioral flags for this object.*
- `virtual void ClearFlags (uint16_t delFlags)`  
*Clear behavioral flags from this object.*
- `uint16_t GetFlags ()`  
*Get the behavioral flags set for this object.*
- `int getPriority ()`  
*Get the message priority for this object.*
- `virtual int RenderToBuffer (eObj_Serialize_t form, uint8_t *buf, int buflen)=0`  
*Render this object to an output buffer.*
- `virtual int RenderToFd (eObj_Serialize_t form, int fd)=0`  
*Render this object to an output socket.*

## Static Public Member Functions

- `static int MqttRxPublish (Client *mClient, int mqttFd, PacketInfo::Msg *msg, void *ctx)`  
*Static message handler for Publish messages received by the registered [Client](#).*

## Protected Member Functions

- `mqtt_leaf (const char *topic, Client *mClient, uint16_t flags=(eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_N`  
*Constructor for [mqtt\\_leaf](#) objects.*
- `mqtt_leaf (mqtt_leaf &parent, const char *topic, Client *mClient, uint16_t flags=(eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|`  
*Constructor for [mqtt\\_leaf](#) objects.*
- `virtual mqtt_leaf * GetChildren ()`  
*Get the first child of this object.*



- virtual [mqtt\\_leaf](#) \* [AddChild](#) ([mqtt\\_leaf](#) \*child)  
*Add an MqttObj as a child to this object.*
- virtual [mqtt\\_leaf](#) \* [RemoveFirstChild](#) ()  
*Remove the first child from this object.*
- void [ForEachLeaf](#) (LeafCallBack \*p, void \*pd)  
*Call a function on each element of the MqttObj tree with this object as the root.*
- void [RootParse](#) ([ParsedJsonDataSet](#) &pjs)  
*Assign to this object tree from a JSON dataset.*
- bool [MatchId](#) ([ParsedJsonDataSet](#) &pjs)  
*Determine whether the JSON dataset's current index matches this object's name.*
- int [QueuePublish](#) ()  
*Queue this object to be Published via the registered [Client](#).*
- int [QueueSubscribe](#) ()  
*Queue this object to be Subscribed via the registered [Client](#).*
- int [QueueUnsubscribe](#) ()  
*Queue this object to be Unsubscribed via the registered [Client](#).*
- int [Assign](#) ()  
*Perform post assignemnt operations.*
- uint8\_t [PubState](#) ()  
*Get the Publish state.*
- uint8\_t [SubState](#) ()  
*Get the Subscribe state of this object.*
- virtual void [SetPubState](#) (uint8\_t newState)  
*Set the Publish state.*
- virtual void [SetSubState](#) (uint8\_t newState)  
*Set the Subscribe state.*
- virtual int [ParseFd](#) (int mqttFd)=0  
*Parse the data from a file descriptor according to this object's current default serialization method and assign the value.*
- int [HandleSubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle a Subscribe Acknowledge message from the broker for our Subscribe request.*
- int [HandleUnsubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.*
- int [HandlePubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Publish Acknowledge message from the broker for our Publish request.*

### Protected Attributes

- [NBInterpolatedString](#) **topicName**  
*Topic name or additional branch.*
- [eObj\\_Serialize\\_t](#) **publishForm**  
*Default serialization form.*
- uint16\_t **mFlags**  
*Behavioral flags.*

### 16.131.1 Detailed Description

Base class for all Mqtt Objects (MqttObj). Mqtt Objects provide a seamless bridge/synchronization to communicate information through an MQTT Broker via a [Client](#) object.

## 16.131.2 Constructor & Destructor Documentation

### 16.131.2.1 mqtt\_leaf() [1/2]

```
MQTT::mqtt_leaf::mqtt_leaf (
 const char * topic,
 Client * mClient,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

[protected]

Constructor for [mqtt\\_leaf](#) objects.

#### Parameters

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>    | Behavioral flags to set on the created object                                                                                      |

### 16.131.2.2 mqtt\_leaf() [2/2]

```
MQTT::mqtt_leaf::mqtt_leaf (
 mqtt_leaf & parent,
 const char * topic,
 Client * mClient,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

[protected]

Constructor for [mqtt\\_leaf](#) objects.

#### Parameters

|                    |                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent object.                                                                                                                   |
| <i>*topic</i>      | Additional Topic Name for the leaf, added as another level in the topic path after the parent objects topic (parent='/foo', this='/foo/bar'). |
| <i>*mClient</i>    | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> .            |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                                 |

## 16.131.3 Member Function Documentation

### 16.131.3.1 AddChild()

```
virtual mqtt_leaf * MQTT::mqtt_leaf::AddChild (
 mqtt_leaf * child) [protected], [virtual]
```

Add an MqttObj as a child to this object.

#### Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>*child</i> | Pointer to the child object to add |
|---------------|------------------------------------|

#### Returns

Pointer to the first child of this object

Reimplemented in [MQTT::mqtt\\_obj](#).

**16.131.3.2 Assign()**

```
int MQTT::mqtt_leaf::Assign () [protected]
```

Perform post assignment operations.

**Returns**

eResult\_t return code

**16.131.3.3 ClearFlags()**

```
virtual void MQTT::mqtt_leaf::ClearFlags (
 uint16_t delFlags) [virtual]
```

Clear behavioral flags from this object.

**Parameters**

|                 |                                          |
|-----------------|------------------------------------------|
| <i>delFlags</i> | The flags to be cleared from this object |
|-----------------|------------------------------------------|

Reimplemented in [MQTT::mqtt\\_IO](#).

**16.131.3.4 ForEachLeaf()**

```
void MQTT::mqtt_leaf::ForEachLeaf (
 LeafCallBack * p,
 void * pd) [protected]
```

Call a function on each element of the MqttObj tree with this object as the root.

**Parameters**

|            |                                                                    |
|------------|--------------------------------------------------------------------|
| <i>*p</i>  | Pointer to the function to be executed on each element of the tree |
| <i>*pd</i> | Pointer to extra data to pass to the function being called         |

**16.131.3.5 GetChildren()**

```
virtual mqtt_leaf * MQTT::mqtt_leaf::GetChildren () [protected], [virtual]
```

Get the first child of this object.

**Returns**

Pointer to the first child of this object

Reimplemented in [MQTT::mqtt\\_obj](#).

**16.131.3.6 GetFlags()**

```
uint16_t MQTT::mqtt_leaf::GetFlags () [inline]
```

Get the behavioral flags set for this object.

**Returns**

The flags to set for this object

**16.131.3.7 getPriority()**

```
int MQTT::mqtt_leaf::getPriority () [inline]
```

Get the message priority for this object.

**Returns**

The message priority for this object.

**16.131.3.8 GetPubTopicName()**

```
virtual int MQTT::mqtt_leaf::GetPubTopicName (
 NBString & s) [virtual]
```

Get the Topic Name this object Publishes to.

**Parameters**

|               |                                                          |
|---------------|----------------------------------------------------------|
| <b>&amp;s</b> | The <a href="#">NBString</a> to copy the Topic Name into |
|---------------|----------------------------------------------------------|

**Returns**

eResult\_t return code

Reimplemented in [MQTT::ConfigExposer](#).

**16.131.3.9 GetRenderLength()**

```
virtual int MQTT::mqtt_leaf::GetRenderLength (
 eObj_Serialize_t form = eObj_Serialize_CURRENT_FORM) [virtual]
```

Get the number of bytes used to serialize this object in a particular format.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <b>form</b> | Serialization format to use |
|-------------|-----------------------------|

**Returns**

Length of serialized data

Reimplemented in [MQTT::ConfigExposer](#), and [MQTT::mqtt\\_obj](#).

**16.131.3.10 GetSerializationForm()**

```
eObj_Serialize_t MQTT::mqtt_leaf::GetSerializationForm () [inline]
```

Get the default serialization format for this object.

**Returns**

eObj\_Serialize\_t Serialization format in use

**16.131.3.11 GetSubTopicFilter()**

```
virtual int MQTT::mqtt_leaf::GetSubTopicFilter (
 NBString & s) [virtual]
```

Get the Subscription Topic Filter this object Subscribes to.

**Parameters**

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <b>&amp;s</b> | The <a href="#">NBString</a> to copy the Subscription Topic Filter into |
|---------------|-------------------------------------------------------------------------|

**Returns**

eResult\_t return code

Reimplemented in [MQTT::ConfigExposer](#), and [MQTT::mqtt\\_obj](#).

**16.131.3.12 HandlePubAck()**

```
int MQTT::mqtt_leaf::HandlePubAck (
 int mqttFd,
 PacketInfo::Msg * msg) [protected]
```

Handle an Publish Acknowledge message from the broker for our Publish request.

## Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>mqttFd</i> | File descriptor to read additional message data from             |
| <i>*msg</i>   | Pointer to message struct containing the Publish Acknowledgement |

## Returns

eResult\_t return code

**16.131.3.13 HandleSubAck()**

```
int MQTT::mqtt_leaf::HandleSubAck (
 int mqttFd,
 PacketInfo::Msg * msg) [protected]
```

Handle a Subscribe Acknowledge message from the broker for our Subscribe request.

## Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>mqttFd</i> | File descriptor to read additional message data from               |
| <i>*msg</i>   | Pointer to message struct containing the Subscribe Acknowledgement |

## Returns

eResult\_t return code

**16.131.3.14 HandleUnsubAck()**

```
int MQTT::mqtt_leaf::HandleUnsubAck (
 int mqttFd,
 PacketInfo::Msg * msg) [protected]
```

Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.

## Parameters

|               |                                                                      |
|---------------|----------------------------------------------------------------------|
| <i>mqttFd</i> | File descriptor to read additional message data from                 |
| <i>*msg</i>   | Pointer to message struct containing the Unsubscribe Acknowledgement |

## Returns

eResult\_t return code

**16.131.3.15 MatchId()**

```
bool MQTT::mqtt_leaf::MatchId (
 ParsedJsonDataSet & pjs) [protected]
```

Determine whether the JSON dataset's current index matches this object's name.

## Parameters

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>&amp;pjs</i> | Reference to the JSON dataset to be assigned from |
|-----------------|---------------------------------------------------|

## Returns

Whether the JSON dataset's current index matches this object's name.

**16.131.3.16 MqttRxPublish()**

```
static int MQTT::mqtt_leaf::MqttRxPublish (
 Client * mClient,
 int mqttFd,
 PacketInfo::Msg * msg,
 void * ctx) [static]
```

Static message handler for Publish messages received by the registered [Client](#).

**Parameters**

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| <i>*mClient</i> | Pointer to the <a href="#">Client</a> that received the message |
| <i>mqttFd</i>   | File Descriptor to read any additional data from                |
| <i>*msg</i>     | Pointer to the structure containing the received message        |
| <i>*ctx</i>     | Pointer to the leaf this message is for                         |

**Returns**

The message priority for this object.

**16.131.3.17 NameMatch()**

```
bool MQTT::mqtt_leaf::NameMatch (
 const char * cp)
```

Determine whether a given name matches this object.

**Parameters**

|            |                       |
|------------|-----------------------|
| <i>*cp</i> | Name to check against |
|------------|-----------------------|

**Returns**

Whether the given name matches this object

**16.131.3.18 ParseFd()**

```
virtual int MQTT::mqtt_leaf::ParseFd (
 int mqttFd) [protected], [pure virtual]
```

Parse the data from a file descriptor according to this object's current default serialization method and assign the value.

**Parameters**

|               |                               |
|---------------|-------------------------------|
| <i>mqttFd</i> | File descriptor to read from. |
|---------------|-------------------------------|

**Returns**

eResult\_t return code

Implemented in [MQTT::ConfigExposer](#), [MQTT::mqtt\\_bool](#), [MQTT::mqtt\\_float](#), [MQTT::mqtt\\_int](#), [MQTT::mqtt\\_IO](#), [MQTT::mqtt\\_obj](#), [MQTT::mqtt\\_string](#), [MQTT::mqtt\\_uint](#), and [MQTT::SubHandler](#).

**16.131.3.19 Publish()**

```
int MQTT::mqtt_leaf::Publish (
 int waitForCompletion = -1)
```

Publishes the current [MQTT](#) object, optionally waiting for completion.

This method queues the object for publication via [MQTT](#). If `waitForCompletion` is greater than 0, or if the object is configured to wait (`eObj_Flag_WaitForPub`), the function will block until the publish operation is complete.

## Parameters

|                          |                                              |
|--------------------------|----------------------------------------------|
| <i>waitForCompletion</i> | Set to 1 to block until the publish is done. |
|--------------------------|----------------------------------------------|

## Returns

eResult\_Success on successful queuing and (if applicable) completion; error codes may propagate from [QueuePublish\(\)](#).

## See also

[QueuePublish](#)

**16.131.3.20 PubState()**

```
uint8_t MQTT::mqtt_leaf::PubState () [inline], [protected]
```

Get the Publish state.

## Returns

eObj\_State\_t state of this object

**16.131.3.21 QueuePublish()**

```
int MQTT::mqtt_leaf::QueuePublish () [protected]
```

Queue this object to be Published via the registered [Client](#).

## Returns

eResult\_t return code

## See also

[Client::Publish](#)

**16.131.3.22 QueueSubscribe()**

```
int MQTT::mqtt_leaf::QueueSubscribe () [protected]
```

Queue this object to be Subscribed via the registered [Client](#).

## Returns

eResult\_t return code

## See also

[Client::Subscribe](#)

**16.131.3.23 QueueUnsubscribe()**

```
int MQTT::mqtt_leaf::QueueUnsubscribe () [protected]
```

Queue this object to be Unsubscribed via the registered [Client](#).

## Returns

eResult\_t return code

## See also

[Client::Unsubscribe](#)

**16.131.3.24 RemoveFirstChild()**

```
virtual mqtt_leaf * MQTT::mqtt_leaf::RemoveFirstChild () [protected], [virtual]
```

Remove the first child from this object.

**Returns**

Pointer to the new first child of this object

Reimplemented in [MQTT::mqtt\\_obj](#).

**16.131.3.25 RenderToBuffer()**

```
virtual int MQTT::mqtt_leaf::RenderToBuffer (
 eObj_Serialize_t form,
 uint8_t * buf,
 int buflen) [pure virtual]
```

Render this object to an output buffer.

**Parameters**

|               |                                   |
|---------------|-----------------------------------|
| <i>form</i>   | Serialization format to render as |
| <i>*buf</i>   | Pointer to output buffer          |
| <i>buflen</i> | Size of the output buffer         |

**Returns**

Length of the rendered data

Implemented in [MQTT::ConfigExposer](#), [MQTT::mqtt\\_bool](#), [MQTT::mqtt\\_float](#), [MQTT::mqtt\\_int](#), [MQTT::mqtt\\_IO](#), [MQTT::mqtt\\_obj](#), [MQTT::mqtt\\_string](#), [MQTT::mqtt\\_uint](#), and [MQTT::SubHandler](#).

**16.131.3.26 RenderToFd()**

```
virtual int MQTT::mqtt_leaf::RenderToFd (
 eObj_Serialize_t form,
 int fd) [pure virtual]
```

Render this object to an output socket.

**Parameters**

|             |                                   |
|-------------|-----------------------------------|
| <i>form</i> | Serialization format to render as |
| <i>fd</i>   | File descriptor to render to      |

**Returns**

Length of the rendered data

Implemented in [MQTT::ConfigExposer](#), [MQTT::mqtt\\_bool](#), [MQTT::mqtt\\_float](#), [MQTT::mqtt\\_int](#), [MQTT::mqtt\\_IO](#), [MQTT::mqtt\\_obj](#), [MQTT::mqtt\\_string](#), [MQTT::mqtt\\_uint](#), and [MQTT::SubHandler](#).

**16.131.3.27 RootParse()**

```
void MQTT::mqtt_leaf::RootParse (
 ParsedJsonDataSet & pjs) [protected]
```

Assign to this object tree from a JSON dataset.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>&amp;pjs</i> | Reference to the JSON dataset to be assigned from |
|-----------------|---------------------------------------------------|



**16.131.3.28 SetFlags()**

```
virtual void MQTT::mqtt_leaf::SetFlags (
 uint16_t newFlags) [virtual]
```

Set additional behavioral flags for this object.

**Parameters**

|                 |                                |
|-----------------|--------------------------------|
| <i>newFlags</i> | The set of new flags to be set |
|-----------------|--------------------------------|

Reimplemented in [MQTT::mqtt\\_IO](#).

**16.131.3.29 SetPublishInterval()**

```
void MQTT::mqtt_leaf::SetPublishInterval (
 uint32_t intervalPeriodTicks)
```

Sets or clears an automatic publishing interval for this [MQTT](#) object.

If a non-zero interval is specified, the object will be automatically published at the given interval (in system ticks). If the interval is zero, the automatic publishing is disabled by deregistering the object from the client's scheduler.

**Parameters**

|                            |                                                                                    |
|----------------------------|------------------------------------------------------------------------------------|
| <i>intervalPeriodTicks</i> | The interval period in system ticks. A value of zero disables periodic publishing. |
|----------------------------|------------------------------------------------------------------------------------|

**Note**

This operation has no effect if the [MQTT](#) client (`mClient`) is not initialized.

**16.131.3.30 SetPubState()**

```
virtual void MQTT::mqtt_leaf::SetPubState (
 uint8_t newState) [protected], [virtual]
```

Set the Publish state.

**Parameters**

|                 |                                       |
|-----------------|---------------------------------------|
| <i>newState</i> | The new Publish state for this object |
|-----------------|---------------------------------------|

Reimplemented in [MQTT::mqtt\\_obj](#).

**16.131.3.31 SetSerializationForm()**

```
virtual void MQTT::mqtt_leaf::SetSerializationForm (
 eObj_Serialize_t form) [virtual]
```

Set the default serialization format for this object.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>form</i> | Serialization format to use |
|-------------|-----------------------------|

**16.131.3.32 SetSubState()**

```
virtual void MQTT::mqtt_leaf::SetSubState (
 uint8_t newState) [protected], [virtual]
```

Set the Subscribe state.

## Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>newState</i> | The new Subscribe state for this object |
|-----------------|-----------------------------------------|

Reimplemented in [MQTT::mqtt\\_obj](#).

**16.131.3.33 SubscribeEvent()**

```
virtual int MQTT::mqtt_leaf::SubscribeEvent (
 int mqttFd,
 PacketInfo::Msg * msg) [virtual]
```

Handle a message sent by the Broker triggered by this object's Subscription.

## Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>mqttFd</i> | File Descriptor to read any additional message data from |
| <i>*msg</i>   | Pointer to message received from the Broker              |

## Returns

eResult\_t return code

Reimplemented in [MQTT::ConfigExposer](#), [MQTT::mqtt\\_bool](#), [MQTT::mqtt\\_float](#), [MQTT::mqtt\\_int](#), [MQTT::mqtt\\_IO](#), [MQTT::mqtt\\_obj](#), [MQTT::mqtt\\_string](#), [MQTT::mqtt\\_uint](#), and [MQTT::SubHandler](#).

**16.131.3.34 SubState()**

```
uint8_t MQTT::mqtt_leaf::SubState () [inline], [protected]
```

Get the Subscribe state of this object.

## Returns

eObj\_State\_t state

**16.131.3.35 TestOrSetValue()**

```
virtual ValueTestResult MQTT::mqtt_leaf::TestOrSetValue (
 ParsedJsonDataSet & pjs,
 bool setVal) [pure virtual]
```

Test whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.

## Parameters

|                 |                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------|
| <i>&amp;pjs</i> | The JSON dataset to use                                                                                 |
| <i>setVal</i>   | true: Assign from the value at the current index, false: test if current index is valid for this object |

## Returns

ValueTestResult indicating the validity of the current index

Implemented in [MQTT::ConfigExposer](#), [MQTT::mqtt\\_bool](#), [MQTT::mqtt\\_float](#), [MQTT::mqtt\\_int](#), [MQTT::mqtt\\_IO](#), [MQTT::mqtt\\_obj](#), [MQTT::mqtt\\_string](#), [MQTT::mqtt\\_uint](#), and [MQTT::SubHandler](#).

The documentation for this class was generated from the following file:

- [mqtt.h](#)

## 16.132 MQTT::mqtt\_obj Class Reference

Base class for easy [MQTT](#) Objects ([MqttObj](#)). [mqtt\\_obj](#) are comprised of higher order parent objects that contain registered child objects.

#include <mqtt.h>

Inherits [MQTT::mqtt\\_leaf](#).

Inherited by [ControlSet](#), [PIDSet](#), and [TempRange](#).

### Public Member Functions

- virtual [ValueTestResult](#) [TestOrSetValue](#) ([ParsedJsonDataSet](#) &pjs, bool setVal) override  
*whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.*
- virtual int [RenderToBuffer](#) ([eObj\\_Serialize\\_t](#) form, uint8\_t \*buf, int buflen) override  
*Render this object to an output buffer.*
- virtual int [RenderToFd](#) ([eObj\\_Serialize\\_t](#) form, int fd) override  
*Render this object to an output socket.*

### Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- ~[mqtt\\_leaf](#) ()  
*~mqtt\_leaf Destructor for base [mqtt\\_leaf](#)*
- void [RemoveFromTree](#) ()  
*Remove this object from it's parent tree.*
- bool [NameMatch](#) (const char \*cp)  
*Determine whether a given name matches this object.*
- virtual int [GetPubTopicName](#) ([NBString](#) &s)  
*Get the Topic Name this object Publishes to.*
- int [Publish](#) (int waitForCompletion=-1)  
*Publishes the current [MQTT](#) object, optionally waiting for completion.*
- void [SetPublishInterval](#) (uint32\_t intervalPeriodTicks)  
*Sets or clears an automatic publishing interval for this [MQTT](#) object.*
- virtual void [SetSerializationForm](#) ([eObj\\_Serialize\\_t](#) form)  
*Set the default serialization format for this object.*
- [eObj\\_Serialize\\_t](#) [GetSerializationForm](#) ()  
*Get the default serialization format for this object.*
- virtual void [SetFlags](#) (uint16\_t newFlags)  
*Set additional behavioral flags for this object.*
- virtual void [ClearFlags](#) (uint16\_t delFlags)  
*Clear behavioral flags from this object.*
- uint16\_t [GetFlags](#) ()  
*Get the behavioral flags set for this object.*
- int [getPriority](#) ()  
*Get the message priority for this object.*

### Protected Member Functions

- [mqtt\\_obj](#) (const char \*topic, [Client](#) \*mClient, uint16\_t flags=([eObj\\_Flag\\_PubOnWrite](#)|[eObj\\_Flag\\_Subscribe](#)|[eObj\\_Flag\\_Sub\\_N](#))  
*Constructor for a standalone [mqtt\\_obj](#) object.*
- [mqtt\\_obj](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient, uint16\_t flags=([eObj\\_Flag\\_PubOnWrite](#)|[eObj\\_Flag\\_Subscribe](#)|[eObj\\_Flag\\_Sub\\_N](#))  
*Constructor for an [mqtt\\_obj](#) with a known parent object.*
- ~[mqtt\\_obj](#) ()  
*~mqtt\_obj Destructor for base [mqtt\\_obj](#)*

- `mqtt_leaf * FindChild (const char *topic)`  
*the child object that is registered for the given Topic Name.*
- virtual `mqtt_leaf * GetChildren ()` override  
*the first child of this object.*
- virtual `mqtt_leaf * AddChild (mqtt_leaf *child)` override  
*an MqttObj as a child to this object.*
- virtual `mqtt_leaf * RemoveFirstChild ()` override  
*Remove the first child from this object.*
- virtual void `SetPubState (uint8_t newState)` override  
*the Publish state*
- virtual void `SetSubState (uint8_t newState)` override  
*the Subscribe state*
- virtual int `ParseFd (int mqttFd)` override  
*the data from a file descriptor according to this object's current default serialization method and assign the value.*
- virtual int `GetSubTopicFilter (NString &s)` override  
*the Subscription Topic Filter this object Subscribes to.*
- virtual int `SubscribeEvent (int mqttFd, PacketInfo::Msg *msg)` override  
*Handle a message sent by the Broker triggered by this object's Subscription.*
- virtual int `GetRenderLength (eObj_Serialize_t form=eObj_Serialize_CURRENT_FORM)` override  
*the number of bytes used to serialize this object in a particular format.*

### Protected Member Functions inherited from MQTT::mqtt\_leaf

- `mqtt_leaf (const char *topic, Client *mClient, uint16_t flags=(eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_N`  
*Constructor for mqtt\_leaf objects.*
- `mqtt_leaf (mqtt_leaf &parent, const char *topic, Client *mClient, uint16_t flags=(eObj_Flag_PubOnWrite|eObj_Flag_Subscribe`  
*Constructor for mqtt\_leaf objects.*
- void `ForEachLeaf (LeafCallBack *p, void *pd)`  
*Call a function on each element of the MqttObj tree with this object as the root.*
- void `RootParse (ParsedJsonDataSet &pjs)`  
*Assign to this object tree from a JSON dataset.*
- bool `MatchId (ParsedJsonDataSet &pjs)`  
*Determine whether the JSON dataset's current index matches this object's name.*
- int `QueuePublish ()`  
*Queue this object to be Published via the registered Client.*
- int `QueueSubscribe ()`  
*Queue this object to be Subscribed via the registered Client.*
- int `QueueUnsubscribe ()`  
*Queue this object to be Unsubscribed via the registered Client.*
- int `Assign ()`  
*Perform post assignemnt operations.*
- uint8\_t `PubState ()`  
*Get the Publish state.*
- uint8\_t `SubState ()`  
*Get the Subscribe state of this object.*
- int `HandleSubAck (int mqttFd, PacketInfo::Msg *msg)`  
*Handle a Subscribe Acknowledge message from the broker for our Subscribe request.*
- int `HandleUnsubAck (int mqttFd, PacketInfo::Msg *msg)`  
*Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.*
- int `HandlePubAck (int mqttFd, PacketInfo::Msg *msg)`  
*Handle an Publish Acknowledge message from the broker for our Publish request.*

### Additional Inherited Members

#### Public Types inherited from MQTT::mqtt\_leaf

- enum [eObj\\_State\\_t](#)  
*Mqtt Object Publication/Subscription States.*

#### Static Public Member Functions inherited from MQTT::mqtt\_leaf

- static int [MqttRxPublish](#) ([Client](#) \*mClient, int mqttFd, [PacketInfo::Msg](#) \*msg, void \*ctx)  
*Static message handler for Publish messages received by the registered [Client](#).*

#### Protected Attributes inherited from MQTT::mqtt\_leaf

- [NBInterpolatedString](#) **topicName**  
*Topic name or additional branch.*
- [eObj\\_Serialize\\_t](#) **publishForm**  
*Default serialization form.*
- [uint16\\_t](#) **mFlags**  
*Behavioral flags.*

### 16.132.1 Detailed Description

Base class for easy MQTT Objects (MqttObj). [mqtt\\_obj](#) are comprised of higher order parent objects that contain registered child objects.

### 16.132.2 Constructor & Destructor Documentation

#### 16.132.2.1 mqtt\_obj() [1/2]

```
MQTT::mqtt_obj::mqtt_obj (
 const char * topic,
 Client * mClient,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

[protected]

Constructor for a standalone [mqtt\\_obj](#) object.

##### Parameters

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>    | Behavioral flags to set on the created object                                                                                      |

#### 16.132.2.2 mqtt\_obj() [2/2]

```
MQTT::mqtt_obj::mqtt_obj (
 mqtt_leaf & parent,
 const char * topic,
 Client * mClient,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

[protected]

Constructor for an [mqtt\\_obj](#) with a known parent object.

##### Parameters

|                    |                             |
|--------------------|-----------------------------|
| <i>&amp;parent</i> | Reference to parent object. |
|--------------------|-----------------------------|

## Parameters

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>    | Behavioral flags to set on the created object                                                                                      |

**16.132.3 Member Function Documentation****16.132.3.1 AddChild()**

virtual [mqtt\\_leaf](#) \* MQTT::mqtt\_obj::AddChild (   
                   [mqtt\\_leaf](#) \* child) [override], [protected], [virtual]   
 an MqttObj as a child to this object.

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>*child</i> | Pointer to the child object to add |
|---------------|------------------------------------|

## Returns

Pointer to the first child of this object

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.132.3.2 FindChild()**

[mqtt\\_leaf](#) \* MQTT::mqtt\_obj::FindChild (   
                   const char \* topic) [protected]   
 the child object that is registered for the given Topic Name.

## Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>*topic</i> | The Topic Name to find the child for |
|---------------|--------------------------------------|

## Returns

Pointer to the child that matches the given Topic Name

**16.132.3.3 GetChildren()**

virtual [mqtt\\_leaf](#) \* MQTT::mqtt\_obj::GetChildren () [override], [protected], [virtual]   
 the first child of this object.

## Returns

Pointer to the first child of this object

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.132.3.4 GetRenderLength()**

virtual int MQTT::mqtt\_obj::GetRenderLength (   
                   eObj\_Serialize\_t form = eObj\_Serialize\_CURRENT\_FORM) [override], [protected],   
 [virtual]   
 the number of bytes used to serialize this object in a particular format.

## Parameters

|             |                             |
|-------------|-----------------------------|
| <i>form</i> | Serialization format to use |
|-------------|-----------------------------|

## Returns

Length of serialized data

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.132.3.5 GetSubTopicFilter()**

```
virtual int MQTT::mqtt_obj::GetSubTopicFilter (
 NBString & s) [override], [protected], [virtual]
```

the Subscription Topic Filter this object Subscribes to.

## Parameters

|    |                                                                         |
|----|-------------------------------------------------------------------------|
| &s | The <a href="#">NBString</a> to copy the Subscription Topic Filter into |
|----|-------------------------------------------------------------------------|

## Returns

eResult\_t return code

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.132.3.6 ParseFd()**

```
virtual int MQTT::mqtt_obj::ParseFd (
 int mqttFd) [override], [protected], [virtual]
```

the data from a file descriptor according to this object's current default serialization method and assign the value.

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>mqttFd</i> | File descriptor to read from. |
|---------------|-------------------------------|

## Returns

eResult\_t return code

Implements [MQTT::mqtt\\_leaf](#).

**16.132.3.7 RemoveFirstChild()**

```
virtual mqtt_leaf * MQTT::mqtt_obj::RemoveFirstChild () [override], [protected], [virtual]
```

Remove the first child from this object.

## Returns

Pointer to the new first child of this object

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.132.3.8 RenderToBuffer()**

```
virtual int MQTT::mqtt_obj::RenderToBuffer (
 eObj_Serialize_t form,
 uint8_t * buf,
 int buflen) [override], [virtual]
```

Render this object to an output buffer.

**Parameters**

|               |                                   |
|---------------|-----------------------------------|
| <i>form</i>   | Serialization format to render as |
| <i>*buf</i>   | Pointer to output buffer          |
| <i>buflen</i> | Size of the output buffer         |

**Returns**

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

**16.132.3.9 RenderToFd()**

```
virtual int MQTT::mqtt_obj::RenderToFd (
 eObj_Serialize_t form,
 int fd) [override], [virtual]
```

Render this object to an output socket.

**Parameters**

|             |                                   |
|-------------|-----------------------------------|
| <i>form</i> | Serialization format to render as |
| <i>fd</i>   | File descriptor to render to      |

**Returns**

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

**16.132.3.10 SetPubState()**

```
virtual void MQTT::mqtt_obj::SetPubState (
 uint8_t newState) [override], [protected], [virtual]
```

the Publish state

**Parameters**

|                 |                                       |
|-----------------|---------------------------------------|
| <i>newState</i> | The new Publish state for this object |
|-----------------|---------------------------------------|

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.132.3.11 SetSubState()**

```
virtual void MQTT::mqtt_obj::SetSubState (
 uint8_t newState) [override], [protected], [virtual]
```

the Subscribe state

**Parameters**

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>newState</i> | The new Subscribe state for this object |
|-----------------|-----------------------------------------|

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.132.3.12 SubscribeEvent()**

```
virtual int MQTT::mqtt_obj::SubscribeEvent (
 int mqttFd,
 PacketInfo::Msg * msg) [override], [protected], [virtual]
```

Handle a message sent by the Broker triggered by this object's Subscription.



## Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>mqttFd</i> | File Descriptor to read any additional message data from |
| <i>*msg</i>   | Pointer to message received from the Broker              |

## Returns

eResult\_t return code

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.132.3.13 TestOrSetValue()**

```
virtual ValueTestResult MQTT::mqtt_obj::TestOrSetValue (
 ParsedJsonDataSet & pjs,
 bool setVal) [override], [virtual]
```

whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.

## Parameters

|                 |                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------|
| <i>&amp;pjs</i> | The JSON dataset to use                                                                                 |
| <i>setVal</i>   | true: Assign from the value at the current index, false: test if current index is valid for this object |

## Returns

ValueTestResult indicating the validity of the current index

Implements [MQTT::mqtt\\_leaf](#).

The documentation for this class was generated from the following file:

- mqtt.h

**16.133 MQTT::mqtt\_string Class Reference**

Easy MqttObj class for automatically publishing and subscribing string values to/from an [MQTT](#) topic. Use like a normal [NBString](#).

```
#include <mqtt.h>
```

Inherits [MQTT::mqtt\\_leaf](#).

**Public Member Functions**

- [mqtt\\_string](#) (const char \*val, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=0)  
*Constructor for a standalone mqtt\_string object.*
- [mqtt\\_string](#) (const [NBString](#) &val, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=0)  
*Constructor for a standalone mqtt\_string object.*
- [mqtt\\_string](#) (const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=0)  
*Constructor for a standalone mqtt\_string object.*
- [mqtt\\_string](#) ([mqtt\\_leaf](#) &parent, const char \*val, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=0)  
*Constructor for a mqtt\_string object that has a parent.*
- [mqtt\\_string](#) ([mqtt\\_leaf](#) &parent, const [NBString](#) &val, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=0)  
*Constructor for a mqtt\_string object that has a parent.*
- [mqtt\\_string](#) ([mqtt\\_leaf](#) &parent, const [NBString](#) &val, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=0)  
*Constructor for a mqtt\_string object that has a parent.*
- [mqtt\\_string](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient=NULL, uint16\_t flags=0)  
*Constructor for a mqtt\_string object that has a parent.*
- [mqtt\\_string](#) & operator= (const [NBString](#) &s)

- Assignment operator for [mqtt\\_string](#). Assigns the value to the object, then performs any post assignment operations.
- [mqtt\\_string](#) & [operator=](#) (const char \*s)
  - Assignment operator for [mqtt\\_string](#). Assigns the value to the object, then performs any post assignment operations.
- [operator NBString](#) () const
  - [NBString](#) conversion operator for [mqtt\\_string](#). Returns the value of the object as an [NBString](#).
- const char [operator\[\]](#) (size\_t pos) const
  - operator that returns the const char value of the character at the index. Operates on const objects.
- char & [operator\[\]](#) (size\_t pos)
  - operator that returns a reference to the character at the given position.
- virtual ValueTestResult [TestOrSetValue](#) (ParsedJsonDataSet &pjs, bool setVal) override
  - Test whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.
- virtual int [SubscribeEvent](#) (int mqttFd, PacketInfo::Msg \*msg) override
  - Handle a message sent by the Broker triggered by this object's Subscription.
- virtual int [RenderToBuffer](#) (eObj\_Serialize\_t form, uint8\_t \*buf, int buflen) override
  - Render this object to an output buffer.
- virtual int [RenderToFd](#) (eObj\_Serialize\_t form, int fd) override
  - Render this object to an output socket.

## Public Member Functions inherited from MQTT::mqtt\_leaf

- ~[mqtt\\_leaf](#) ()
  - ~mqtt\_leaf Destructor for base [mqtt\\_leaf](#)
- void [RemoveFromTree](#) ()
  - Remove this object from it's parent tree.
- bool [NameMatch](#) (const char \*cp)
  - Determine whether a given name matches this object.
- virtual int [GetPubTopicName](#) (NBString &s)
  - Get the Topic Name this object Publishes to.
- virtual int [GetSubTopicFilter](#) (NBString &s)
  - Get the Subscription Topic Filter this object Subscribes to.
- virtual int [GetRenderLength](#) (eObj\_Serialize\_t form=eObj\_Serialize\_CURRENT\_FORM)
  - Get the number of bytes used to serialize this object in a particular format.
- int [Publish](#) (int waitForCompletion=-1)
  - Publishes the current [MQTT](#) object, optionally waiting for completion.
- void [SetPublishInterval](#) (uint32\_t intervalPeriodTicks)
  - Sets or clears an automatic publishing interval for this [MQTT](#) object.
- virtual void [SetSerializationForm](#) (eObj\_Serialize\_t form)
  - Set the default serialization format for this object.
- eObj\_Serialize\_t [GetSerializationForm](#) ()
  - Get the default serialization format for this object.
- virtual void [SetFlags](#) (uint16\_t newFlags)
  - Set additional behavioral flags for this object.
- virtual void [ClearFlags](#) (uint16\_t delFlags)
  - Clear behavioral flags from this object.
- uint16\_t [GetFlags](#) ()
  - Get the behavioral flags set for this object.
- int [getPriority](#) ()
  - Get the message priority for this object.

### Protected Member Functions

- virtual int [ParseFd](#) (int mqttFd) override  
*the data from a file descriptor according to this object's current default serialization method and assign the value.*

### Protected Member Functions inherited from MQTT::mqtt\_leaf

- [mqtt\\_leaf](#) (const char \*topic, [Client](#) \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_N  
*Constructor for [mqtt\\_leaf](#) objects.*
- [mqtt\\_leaf](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|  
*Constructor for [mqtt\\_leaf](#) objects.*
- virtual [mqtt\\_leaf](#) \* [GetChildren](#) ()  
*Get the first child of this object.*
- virtual [mqtt\\_leaf](#) \* [AddChild](#) ([mqtt\\_leaf](#) \*child)  
*Add an MqttObj as a child to this object.*
- virtual [mqtt\\_leaf](#) \* [RemoveFirstChild](#) ()  
*Remove the first child from this object.*
- void [ForEachLeaf](#) (LeafCallBack \*p, void \*pd)  
*Call a function on each element of the MqttObj tree with this object as the root.*
- void [RootParse](#) ([ParsedJsonDataSet](#) &pjs)  
*Assign to this object tree from a JSON dataset.*
- bool [MatchId](#) ([ParsedJsonDataSet](#) &pjs)  
*Determine whether the JSON dataset's current index matches this object's name.*
- int [QueuePublish](#) ()  
*Queue this object to be Published via the registered [Client](#).*
- int [QueueSubscribe](#) ()  
*Queue this object to be Subscribed via the registered [Client](#).*
- int [QueueUnsubscribe](#) ()  
*Queue this object to be Unsubscribed via the registered [Client](#).*
- int [Assign](#) ()  
*Perform post assignemnt operations.*
- uint8\_t [PubState](#) ()  
*Get the Publish state.*
- uint8\_t [SubState](#) ()  
*Get the Subscribe state of this object.*
- virtual void [SetPubState](#) (uint8\_t newState)  
*Set the Publish state.*
- virtual void [SetSubState](#) (uint8\_t newState)  
*Set the Subscribe state.*
- int [HandleSubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle a Subscribe Acknowledge message from the broker for our Subscribe request.*
- int [HandleUnsubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.*
- int [HandlePubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Publish Acknowledge message from the broker for our Publish request.*

### Additional Inherited Members

### Public Types inherited from MQTT::mqtt\_leaf

- enum [eObj\\_State\\_t](#)  
*Mqtt Object Publication/Subscription States.*

## Static Public Member Functions inherited from MQTT::mqtt\_leaf

- static int [MqttRxPublish](#) ([Client](#) \*mClient, int mqttFd, [PacketInfo::Msg](#) \*msg, void \*ctx)  
*Static message handler for Publish messages received by the registered [Client](#).*

## Protected Attributes inherited from MQTT::mqtt\_leaf

- [NBInterpolatedString](#) **topicName**  
*Topic name or additional branch.*
- [eObj\\_Serialize\\_t](#) **publishForm**  
*Default serialization form.*
- uint16\_t **mFlags**  
*Behavioral flags.*

### 16.133.1 Detailed Description

Easy MqttObj class for automatically publishing and subscribing string values to/from an [MQTT](#) topic. Use like a normal [NBString](#).

### 16.133.2 Constructor & Destructor Documentation

#### 16.133.2.1 mqtt\_string() [1/6]

```
MQTT::mqtt_string::mqtt_string (
 const char * val,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = 0)
```

Constructor for a standalone [mqtt\\_string](#) object.

##### Parameters

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>val</i>      | Initial value for the string                                                                                                       |
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>    | Behavioral flags to set on the created object                                                                                      |

#### 16.133.2.2 mqtt\_string() [2/6]

```
MQTT::mqtt_string::mqtt_string (
 const NBString & val,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = 0)
```

Constructor for a standalone [mqtt\\_string](#) object.

##### Parameters

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>val</i>      | Initial value for the string                                                                                                       |
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>    | Behavioral flags to set on the created object                                                                                      |

**16.133.2.3 mqtt\_string() [3/6]**

```
MQTT::mqtt_string::mqtt_string (
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = 0)
```

Constructor for a standalone [mqtt\\_string](#) object.

**Parameters**

|                 |                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>*topic</i>   | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>    | Behavioral flags to set on the created object                                                                                      |

**16.133.2.4 mqtt\_string() [4/6]**

```
MQTT::mqtt_string::mqtt_string (
 mqtt_leaf & parent,
 const char * val,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = 0)
```

Constructor for a [mqtt\\_string](#) object that has a parent.

**Parameters**

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>val</i>         | Initial value for the string                                                                                                       |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i>    | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

**16.133.2.5 mqtt\_string() [5/6]**

```
MQTT::mqtt_string::mqtt_string (
 mqtt_leaf & parent,
 const NBString & val,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = 0)
```

Constructor for a [mqtt\\_string](#) object that has a parent.

**Parameters**

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>val</i>         | Initial value for the string                                                                                                       |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i>    | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

### 16.133.2.6 mqtt\_string() [6/6]

```
MQTT::mqtt_string::mqtt_string (
 mqtt_leaf & parent,
 const char * topic,
 Client * mClient = NULL,
 uint16_t flags = 0)
```

Constructor for a [mqtt\\_string](#) object that has a parent.

#### Parameters

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*mClient</i>    | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

## 16.133.3 Member Function Documentation

### 16.133.3.1 operator NBString()

```
MQTT::mqtt_string::operator NBString () const [inline]
```

[NBString](#) conversion operator for [mqtt\\_string](#). Returns the value of the object as an [NBString](#).

#### Returns

The [NBString](#) value of the object

### 16.133.3.2 operator=() [1/2]

```
mqtt_string & MQTT::mqtt_string::operator= (
 const char * s)
```

Assignment operator for [mqtt\\_string](#). Assigns the value to the object, then performs any post assignment operations.

#### Parameters

|          |                                    |
|----------|------------------------------------|
| <i>s</i> | The value to assign to the object. |
|----------|------------------------------------|

#### Returns

A reference to the object being assigned to.

### 16.133.3.3 operator=() [2/2]

```
mqtt_string & MQTT::mqtt_string::operator= (
 const NBString & s)
```

Assignment operator for [mqtt\\_string](#). Assigns the value to the object, then performs any post assignment operations.

#### Parameters

|          |                                    |
|----------|------------------------------------|
| <i>s</i> | The value to assign to the object. |
|----------|------------------------------------|

#### Returns

A reference to the object being assigned to.

**16.133.3.4 operator[]()** [1/2]

```
char & MQTT::mqtt_string::operator[] (
 size_t pos) [inline]
```

operator that returns a reference to the character at the given position.

**Returns**

Reference to the character at the given position

**16.133.3.5 operator[]()** [2/2]

```
const char MQTT::mqtt_string::operator[] (
 size_t pos) const [inline]
```

operator that returns the const char value of the character an the index. Operates on const objects.

**Returns**

The value of the characterat the position

**16.133.3.6 ParseFd()**

```
virtual int MQTT::mqtt_string::ParseFd (
 int mqttFd) [override], [protected], [virtual]
```

the data from a file descriptor according to this object's current default serialization method and assign the value.

**Parameters**

|               |                               |
|---------------|-------------------------------|
| <i>mqttFd</i> | File descriptor to read from. |
|---------------|-------------------------------|

**Returns**

eResult\_t return code

Implements [MQTT::mqtt\\_leaf](#).

**16.133.3.7 RenderToBuffer()**

```
virtual int MQTT::mqtt_string::RenderToBuffer (
 eObj_Serialize_t form,
 uint8_t * buf,
 int buflen) [override], [virtual]
```

Render this object to an output buffer.

**Parameters**

|               |                                   |
|---------------|-----------------------------------|
| <i>form</i>   | Serialization format to render as |
| <i>*buf</i>   | Pointer to output buffer          |
| <i>buflen</i> | Size of the output buffer         |

**Returns**

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

**16.133.3.8 RenderToFd()**

```
virtual int MQTT::mqtt_string::RenderToFd (
 eObj_Serialize_t form,
 int fd) [override], [virtual]
```

Render this object to an output socket.

## Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>form</i> | Serialization format to render as |
| <i>fd</i>   | File descriptor to render to      |

## Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

**16.133.3.9 SubscribeEvent()**

```
virtual int MQTT::mqtt_string::SubscribeEvent (
 int mqttFd,
 PacketInfo::Msg * msg) [override], [virtual]
```

Handle a message sent by the Broker triggered by this object's Subscription.

## Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>mqttFd</i> | File Descriptor to read any additional message data from |
| <i>*msg</i>   | Pointer to message received from the Broker              |

## Returns

eResult\_t return code

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.133.3.10 TestOrSetValue()**

```
virtual ValueTestResult MQTT::mqtt_string::TestOrSetValue (
 ParsedJsonDataSet & pjs,
 bool setVal) [override], [virtual]
```

Test whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.

## Parameters

|                 |                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------|
| <i>&amp;pjs</i> | The JSON dataset to use                                                                                 |
| <i>setVal</i>   | true: Assign from the value at the current index, false: test if current index is valid for this object |

## Returns

ValueTestResult indicating the validity of the current index

Implements [MQTT::mqtt\\_leaf](#).

The documentation for this class was generated from the following file:

- mqtt.h

**16.134 MQTT::mqtt\_uint Class Reference**

Easy MqttObj class for automatically publishing and subscribing unsigned integer values to/from an [MQTT](#) topic. Use like a normal uint32\_t.

```
#include <mqtt.h>
```

Inherits [MQTT::mqtt\\_leaf](#).



## Public Member Functions

- [mqtt\\_uint](#) (uint32\_t val, const char \*topic, [Client](#) \*client, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_NoL)  
*Constructor for a standalone [mqtt\\_uint](#) object.*
- [mqtt\\_uint](#) (const char \*topic, [Client](#) \*client, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_NoL)  
*Constructor for a standalone [mqtt\\_uint](#) object.*
- [mqtt\\_uint](#) ([mqtt\\_leaf](#) &parent, uint32\_t val, const char \*topic, [Client](#) \*client, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_NoL)  
*Constructor for a [mqtt\\_uint](#) object that has a parent.*
- [mqtt\\_uint](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*client, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_NoL)  
*Constructor for a [mqtt\\_uint](#) object that has a parent.*
- [mqtt\\_uint](#) & operator= (uint32\_t i)  
*Assignment operator for [mqtt\\_uint](#). Assigns the value to the object, then performs any post assignment operations.*
- [operator unsigned](#) () const  
*Unsigned integer conversion operator for [mqtt\\_uint](#). Returns the value of the object as an unsigned int.*
- [mqtt\\_uint](#) & operator++ ()  
*Preincrement the object.*
- uint32\_t [operator++](#) (int)  
*Postincrement the object.*
- [mqtt\\_uint](#) & operator-- ()  
*Predecrement the object.*
- uint32\_t [operator--](#) (int)  
*Postdecrement the object.*
- virtual ValueTestResult [TestOrSetValue](#) ([ParsedJsonDataSet](#) &pjs, bool setVal) override  
*whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.*
- virtual int [SubscribeEvent](#) (int mqttFd, [PacketInfo::Msg](#) \*msg) override  
*Handle a message sent by the Broker triggered by this object's Subscription.*
- virtual int [RenderToBuffer](#) ([eObj\\_Serialize\\_t](#) form, uint8\_t \*buf, int buflen) override  
*Render this object to an output buffer.*
- virtual int [RenderToFd](#) ([eObj\\_Serialize\\_t](#) form, int fd) override  
*Render this object to an output socket.*

Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- [~mqtt\\_leaf](#) ()  
*[~mqtt\\_leaf](#) Destructor for base [mqtt\\_leaf](#)*
- void [RemoveFromTree](#) ()  
*Remove this object from it's parent tree.*
- bool [NameMatch](#) (const char \*cp)  
*Determine whether a given name matches this object.*
- virtual int [GetPubTopicName](#) ([NBString](#) &s)  
*Get the Topic Name this object Publishes to.*
- virtual int [GetSubTopicFilter](#) ([NBString](#) &s)  
*Get the Subscription Topic Filter this object Subscribes to.*
- virtual int [GetRenderLength](#) ([eObj\\_Serialize\\_t](#) form=[eObj\\_Serialize\\_CURRENT\\_FORM](#))  
*Get the number of bytes used to serialize this object in a particular format.*
- int [Publish](#) (int waitForCompletion=-1)  
*Publishes the current [MQTT](#) object, optionally waiting for completion.*
- void [SetPublishInterval](#) (uint32\_t intervalPeriodTicks)  
*Sets or clears an automatic publishing interval for this [MQTT](#) object.*
- virtual void [SetSerializationForm](#) ([eObj\\_Serialize\\_t](#) form)  
*Set the default serialization format for this object.*

- [eObj\\_Serialize\\_t GetSerializationForm](#) ()  
*Get the default serialization format for this object.*
- virtual void [SetFlags](#) (uint16\_t newFlags)  
*Set additional behavioral flags for this object.*
- virtual void [ClearFlags](#) (uint16\_t delFlags)  
*Clear behavioral flags from this object.*
- uint16\_t [GetFlags](#) ()  
*Get the behavioral flags set for this object.*
- int [getPriority](#) ()  
*Get the message priority for this object.*

### Protected Member Functions

- virtual int [ParseFd](#) (int mqttFd) override  
*the data from a file descriptor according to this object's current default serialization method and assign the value.*

### Protected Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- [mqtt\\_leaf](#) (const char \*topic, [Client](#) \*mClient, uint16\_t flags=([eObj\\_Flag\\_PubOnWrite](#)|[eObj\\_Flag\\_Subscribe](#)|[eObj\\_Flag\\_Sub\\_N](#))  
*Constructor for [mqtt\\_leaf](#) objects.*
- [mqtt\\_leaf](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient, uint16\_t flags=([eObj\\_Flag\\_PubOnWrite](#)|[eObj\\_Flag\\_Subscribe](#)|[eObj\\_Flag\\_Sub\\_N](#))  
*Constructor for [mqtt\\_leaf](#) objects.*
- virtual [mqtt\\_leaf](#) \* [GetChildren](#) ()  
*Get the first child of this object.*
- virtual [mqtt\\_leaf](#) \* [AddChild](#) ([mqtt\\_leaf](#) \*child)  
*Add an MqttObj as a child to this object.*
- virtual [mqtt\\_leaf](#) \* [RemoveFirstChild](#) ()  
*Remove the first child from this object.*
- void [ForEachLeaf](#) (LeafCallBack \*p, void \*pd)  
*Call a function on each element of the MqttObj tree with this object as the root.*
- void [RootParse](#) ([ParsedJsonDataSet](#) &pjs)  
*Assign to this object tree from a JSON dataset.*
- bool [MatchId](#) ([ParsedJsonDataSet](#) &pjs)  
*Determine whether the JSON dataset's current index matches this object's name.*
- int [QueuePublish](#) ()  
*Queue this object to be Published via the registered [Client](#).*
- int [QueueSubscribe](#) ()  
*Queue this object to be Subscribed via the registered [Client](#).*
- int [QueueUnsubscribe](#) ()  
*Queue this object to be Unsubscribed via the registered [Client](#).*
- int [Assign](#) ()  
*Perform post assignemnt operations.*
- uint8\_t [PubState](#) ()  
*Get the Publish state.*
- uint8\_t [SubState](#) ()  
*Get the Subscribe state of this object.*
- virtual void [SetPubState](#) (uint8\_t newState)  
*Set the Publish state.*
- virtual void [SetSubState](#) (uint8\_t newState)  
*Set the Subscribe state.*
- int [HandleSubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle a Subscribe Acknowledge message from the broker for our Subscribe request.*

- int [HandleUnsubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.*
- int [HandlePubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)  
*Handle an Publish Acknowledge message from the broker for our Publish request.*

### Additional Inherited Members

### Public Types inherited from [MQTT::mqtt\\_leaf](#)

- enum [eObj\\_State\\_t](#)  
*Mqtt Object Publication/Subscription States.*

### Static Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- static int [MqttRxPublish](#) ([Client](#) \*mClient, int mqttFd, [PacketInfo::Msg](#) \*msg, void \*ctx)  
*Static message handler for Publish messages received by the registered [Client](#).*

### Protected Attributes inherited from [MQTT::mqtt\\_leaf](#)

- [NBInterpolatedString](#) **topicName**  
*Topic name or additional branch.*
- [eObj\\_Serialize\\_t](#) **publishForm**  
*Default serialization form.*
- uint16\_t **mFlags**  
*Behavioral flags.*

## 16.134.1 Detailed Description

Easy MqttObj class for automatically publishing and subscribing unsigned integer values to/from an [MQTT](#) topic. Use like a normal uint32\_t.

## 16.134.2 Constructor & Destructor Documentation

### 16.134.2.1 mqtt\_uint() [1/4]

```
MQTT::mqtt_uint::mqtt_uint (
 uint32_t val,
 const char * topic,
 Client * client,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a standalone [mqtt\\_uint](#) object.

#### Parameters

|                |                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>val</i>     | Initial value for the integer                                                                                                      |
| <i>*topic</i>  | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*client</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>   | Behavioral flags to set on the created object                                                                                      |

### 16.134.2.2 mqtt\_uint() [2/4]

```
MQTT::mqtt_uint::mqtt_uint (
 const char * topic,
 Client * client,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a standalone [mqtt\\_uint](#) object.

## Parameters

|                |                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>*topic</i>  | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*client</i> | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>   | Behavioral flags to set on the created object                                                                                      |

**16.134.2.3 mqtt\_uint() [3/4]**

```
MQTT::mqtt_uint::mqtt_uint (
 mqtt_leaf & parent,
 uint32_t val,
 const char * topic,
 Client * client,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a [mqtt\\_uint](#) object that has a parent.

## Parameters

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>val</i>         | Initial value for the integer                                                                                                      |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*client</i>     | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

**16.134.2.4 mqtt\_uint() [4/4]**

```
MQTT::mqtt_uint::mqtt_uint (
 mqtt_leaf & parent,
 const char * topic,
 Client * client,
 uint16_t flags = (eObj_Flag_PubOnWrite|eObj_Flag_Subscribe|eObj_Flag_Sub_NoLocal))
```

Constructor for a [mqtt\\_uint](#) object that has a parent.

## Parameters

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>&amp;parent</i> | Reference to parent MqttObj                                                                                                        |
| <i>*topic</i>      | Mqtt Topic Name to use for Publish/Subscribe.                                                                                      |
| <i>*client</i>     | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>       | Behavioral flags to set on the created object                                                                                      |

**16.134.3 Member Function Documentation****16.134.3.1 operator unsigned()**

```
MQTT::mqtt_uint::operator unsigned () const [inline]
```

Unsigned integer conversion operator for [mqtt\\_uint](#). Returns the value of the object as an unsigned int.

## Returns

The unsigned integer value of the object

**16.134.3.2 operator++()** [1/2]

`mqtt_uint & MQTT::mqtt_uint::operator++ () [inline]`  
 Preincrement the object.

**Returns**

Reference to the object

**16.134.3.3 operator++()** [2/2]

`uint32_t MQTT::mqtt_uint::operator++ (int ) [inline]`  
 Postincrement the object.

**Returns**

The unsigned integer value before being incremented

**16.134.3.4 operator--()** [1/2]

`mqtt_uint & MQTT::mqtt_uint::operator-- () [inline]`  
 Predecrement the object.

**Returns**

Reference to the object

**16.134.3.5 operator--()** [2/2]

`uint32_t MQTT::mqtt_uint::operator-- (int ) [inline]`  
 Postdecrement the object.

**Returns**

The unsigned integer value before being decremented

**16.134.3.6 operator=()**

`mqtt_uint & MQTT::mqtt_uint::operator= (uint32_t i)`

Assignment operator for `mqtt_uint`. Assigns the value to the object, then performs any post assignment operations.

**Parameters**

|          |                                    |
|----------|------------------------------------|
| <i>i</i> | The value to assign to the object. |
|----------|------------------------------------|

**Returns**

A reference to the object being assigned to.

**16.134.3.7 ParseFd()**

`virtual int MQTT::mqtt_uint::ParseFd (int mqttFd) [override], [protected], [virtual]`

the data from a file descriptor according to this object's current default serialization method and assign the value.

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>mqttFd</i> | File descriptor to read from. |
|---------------|-------------------------------|

## Returns

eResult\_t return code

Implements [MQTT::mqtt\\_leaf](#).

**16.134.3.8 RenderToBuffer()**

```
virtual int MQTT::mqtt_uint::RenderToBuffer (
 eObj_Serialize_t form,
 uint8_t * buf,
 int buflen) [override], [virtual]
```

Render this object to an output buffer.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>form</i>   | Serialization format to render as |
| <i>*buf</i>   | Pointer to output buffer          |
| <i>buflen</i> | Size of the output buffer         |

## Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

**16.134.3.9 RenderToFd()**

```
virtual int MQTT::mqtt_uint::RenderToFd (
 eObj_Serialize_t form,
 int fd) [override], [virtual]
```

Render this object to an output socket.

## Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>form</i> | Serialization format to render as |
| <i>fd</i>   | File descriptor to render to      |

## Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

**16.134.3.10 SubscribeEvent()**

```
virtual int MQTT::mqtt_uint::SubscribeEvent (
 int mqttFd,
 PacketInfo::Msg * msg) [override], [virtual]
```

Handle a message sent by the Broker triggered by this object's Subscription.

## Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>mqttFd</i> | File Descriptor to read any additional message data from |
| <i>*msg</i>   | Pointer to message received from the Broker              |

**Returns**

eResult\_t return code

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.134.3.11 TestOrSetValue()**

```
virtual ValueTestResult MQTT::mqtt_uint::TestOrSetValue (
 ParsedJsonDataSet & pjs,
 bool setVal) [override], [virtual]
```

whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.

**Parameters**

|                 |                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------|
| <i>&amp;pjs</i> | The JSON dataset to use                                                                                 |
| <i>setVal</i>   | true: Assign from the value at the current index, false: test if current index is valid for this object |

**Returns**

ValueTestResult indicating the validity of the current index

Implements [MQTT::mqtt\\_leaf](#).

The documentation for this class was generated from the following file:

- mqtt.h

**16.135 MQTT::PacketInfo::Msg Struct Reference**

[PacketInfo::Msg](#) message data structure.

```
#include <mqtt_msg_reqs.h>
```

**Public Member Functions**

- Pkt::eType\_t [GetType](#) ()
- uint8\_t [GetFlags](#) ()
- [Msg](#) ()

**16.135.1 Detailed Description**

[PacketInfo::Msg](#) message data structure.

**16.135.2 Constructor & Destructor Documentation****16.135.2.1 Msg()**

```
MQTT::PacketInfo::Msg::Msg () [inline]
```

**16.135.3 Member Function Documentation****16.135.3.1 GetFlags()**

```
uint8_t MQTT::PacketInfo::Msg::GetFlags () [inline]
```

**16.135.3.2 GetType()**

```
Pkt::eType_t MQTT::PacketInfo::Msg::GetType () [inline]
```

The documentation for this struct was generated from the following file:

- mqtt\_msg\_reqs.h

## 16.136 NBInterpolatedString Class Reference

A wrapper for [NBString::Interpolate\(\)](#) ideal for easy access to ephemeral values.

```
#include <nbstring.h>
```

Inherits [NBString](#).

### Public Member Functions

- **NBInterpolatedString** ()  
*Construct a [NBInterpolatedString](#) object.*
- **NBInterpolatedString** (const [NBString](#) &str)  
*Construct a new [NBInterpolatedString](#) object from an existing [NBInterpolatedString](#) object.*
- **NBInterpolatedString** (const [NBString](#) &str, size\_t pos, size\_t len=npos)  
*Construct a new [NBInterpolatedString](#) object from a substring of an existing [NBInterpolatedString](#) object.*
- **NBInterpolatedString** (const char \*s)  
*Construct a [NBInterpolatedString](#) object from a character string (null terminated array of characters)*
- **NBInterpolatedString** (const char \*s, size\_t n)  
*Construct a [NBInterpolatedString](#) object from a character string up to the specified amount.*
- **~NBInterpolatedString** ()  
*[NBInterpolatedString](#) destructor.*
- **NBString & Append** (const char \*str, size\_t len)  
*Append a string to an existing [NBString](#) object.*
- **NBString & FdAppend** (int fd, size\_t len)  
*Append from a file descriptor to an existing [NBString](#) object.*
- **NBInterpolatedString & Interpolate** ()  
*Perform a string interpolation in-place. This allows NBStrings to easily access the NetBurner Config System, as in:*

### Public Member Functions inherited from [NBString](#)

- **NBString** ()  
*Construct a [NBString](#) object.*
- **NBString** (const [NBString](#) &str)  
*Construct a new [NBString](#) object from an existing [NBString](#) object.*
- **NBString** (const [NBString](#) &str, size\_t pos, size\_t len=npos)  
*Construct a new [NBString](#) object from a substring of an existing [NBString](#) object.*
- **NBString** (const char \*s)  
*Construct a [NBString](#) object from a character string (null terminated array of characters)*
- **NBString** (const char \*s, size\_t n)  
*Construct a [NBString](#) object from a character string up to the specified amount.*
- **NBString** (const [JsonRef](#) &jr)  
*Construct a [NBString](#) object using the current string referenced by a [JsonRef](#). This allows for using Large Strings without an intermediate malloc in the JsonParser.*
- **~NBString** ()  
*[NBString](#) destructor.*
- const char \* **c\_str** () const  
*Method to pass a [NBString](#) as a constant char \*.*
- **NBString substr** (size\_t pos=0, size\_t len=npos) const  
*Creates a new [NBString](#) object from a substring of the existing [NBString](#) object.*
- int **compare** (const [NBString](#) &str) const  
*Compares two [NBString](#) objects.*
- int **compare** (const char \*s) const  
*Compares the [NBString](#) object to a character string.*
- size\_t **find** (const [NBString](#) str, size\_t pos=0) const



- Find a substring within a [NBString](#) object.*

  - [size\\_t find](#) (const char \*str, size\_t pos=0) const

*Find a substring within a character string.*

  - [size\\_t find](#) (const char \*s, size\_t pos, size\_t n) const

*Find a substring within a character string.*

  - [size\\_t find](#) (char c, size\_t pos=0) const

*Find the first occurrence of a character within the [NBString](#) object.*

  - [size\\_t replace](#) (const char \*findStr, char rep, size\_t startPos=0, size\_t end=0)

*Replace all occurrences of all "<findStr>" characters within the [NBString](#) object.*

  - [size\\_t replace](#) (char c, char rep, size\_t startPos=0, size\_t end=0)

*Replace all occurrences of all "<findStr>" characters within the [NBString](#) object.*

  - void **shrink** (int n)

*Reduces the length of the string by n.*

  - [size\\_t size](#) () const

*Returns the current size of memory allocated for the string.*

  - [size\\_t length](#) () const

*Returns the length of the string.*

  - void **clear** ()

*Clear a [NBString](#) object and free allocated memory.*

  - bool **empty** () const

*Check if a string is empty.*

  - [NBString & Append](#) (const char \*str, size\_t len)

*Append a string to an existing [NBString](#) object.*

  - [NBString & FdAppend](#) (int fd, size\_t len)

*Append from a file descriptor to an existing [NBString](#) object.*

  - [NBString & Reserve](#) (size\_t len)

*Reserve an additional buffer amount.*

  - int **vsprintf** (const char \*format, va\_list &vl)

*Print to a string with formatting.*

  - int **sprintf** (const char \*format,...)

*sprintf to a string with formatting to a character array*

  - int **sprintf** ([NBString](#) const format,...)

*sprintf to a string with formatting as a [NBString](#) object*

  - int **siprintf** (const char \*format,...)

*isprintf (integer) to a string with formatting to a character array*

  - int **siprintf** ([NBString](#) const format,...)

*isprintf (integer) to a string with formatting to a [NBString](#) object*

  - int **stoi** () const

*Parse the string and convert to an integer number.*

  - long **stol** () const

*Parse the string value and convert to a long integer number.*

  - unsigned int **stoui** () const

*Parse the string value and convert to a const integer number.*

  - unsigned long **stoul** () const

*Parse the string value and convert to a const unsigned long integer number.*

  - double **stod** () const

*Parse the string value and convert to a const double float number.*

  - [IPADDR to\\_ipaddr](#) () const

*Parse the string value and convert to an IPADDR IP address.*

  - [size\\_t copy](#) (char \*s, size\_t len, size\_t pos=0) const

*Copy a substring, does not null terminate.*

- `size_t strcpy` (char \*s, size\_t len, size\_t pos=0) const  
*Copy a substring, with null termination.*
- `bool Interpolate (NBString &dest)`  
*Perform a string interpolation and place the finished interpolation in the Destination string. This allows NBStrings to easily access the NetBurner Config System, as in:*
- `bool Interpolate ()`  
*Perform an in-place string interpolation. This allows NBStrings to easily access the NetBurner Config System, as in:*

### 16.136.1 Detailed Description

A wrapper for `NBString::Interpolate()` ideal for easy access to ephemeral values.

The NetBurner string interpolation system is a powerful and easy way to read system data and should be used carefully. For example, it would be unwise to allow untrusted user input to affect these interpolation commands, in case there is any information in the Config system that you don't want public. Normal sanity checking of regular user input is also advisable.

Syntax is similar to `vim` or `ex` commands, but simplified. There are currently two commands:

- `d` deletes matching substrings
- `s` replaces matching substrings

{ and } surround the interpolation command including any Config system variable name/path. Colons separate names, commands, and arguments. Semicolons allow command chaining. { { or } } are escaped curly braces. \ : can escape colons (actually "\ : " when writing C++ code, since \ is also a C++ escape character)

For example, if the config variable `AppData.foo` is "a\_b\_c\_d", then

```
NBInterpolatedString example("{AppData.foo:d:_;}");
example.Interpolate();
```

Would result in `example` equalling "abcd".

Commands can be chained with ; – if the config variable `AppData.foo` is "2\_3\_4\_5es", then

```
NBInterpolatedString example("{AppData.foo:d:_;s:2345:potato;}");
example.Interpolate();
```

Would result in `example` equalling "potatoes".

Commands are processed from left to right. If the config same variable `AppData.foo` of "2\_3\_4\_5es" is interpolated like this:

```
NBInterpolatedString example("{AppData.foo:s:2345:potato;d:_;}");
example.Interpolate();
```

It would instead would yield "2345es".

In our PubSub, Config, and SubHandler [MQTT](#) examples, we remove the first three octets of a known MAC address as well as the remaining semicolons with an interpolator like this: `{Sys.NetIf.Ethernet0.MAC:d:00\:\:03\:\:F4\:\;;d:\:\:}`

In order, this does the following:

- `Sys.NetIf.Ethernet0.MAC` obtains the matching Config string (visible at <http://<your device ip or hostname>:20034/UI.html>)
- `:` separates commands from config variable path
- `d:` delete the following literal substring:
- `00\:\:03\:\:F4\:\:` "00:03:F4:"
- `;` end command, begin new command with previous command's output as input
- `d:` delete the following literal substring:
- `\:\:` any/all colons remaining in the string

Thus a MAC address like 00:03:F4:12:34:56 will become 123456 in the resulting output.

### 16.136.2 Constructor & Destructor Documentation

#### 16.136.2.1 NBInterpolatedString() [1/4]

```
NBInterpolatedString::NBInterpolatedString (
 const NBString & str)
```

Construct a new `NBInterpolatedString` object from an existing `NBInterpolatedString` object.

## Parameters

|            |                                                      |
|------------|------------------------------------------------------|
| <i>str</i> | Existing <a href="#">NBInterpolatedString</a> object |
|------------|------------------------------------------------------|

**16.136.2.2 NBInterpolatedString()** [2/4]

```
NBInterpolatedString::NBInterpolatedString (
 const NBString & str,
 size_t pos,
 size_t len = npos)
```

Construct a new [NBInterpolatedString](#) object from a substring of an existing [NBInterpolatedString](#) object.

## Parameters

|            |                                                      |
|------------|------------------------------------------------------|
| <i>str</i> | Existing <a href="#">NBInterpolatedString</a> object |
| <i>pos</i> | Starting position to copy                            |
| <i>len</i> | Ending position to copy                              |

**16.136.2.3 NBInterpolatedString()** [3/4]

```
NBInterpolatedString::NBInterpolatedString (
 const char * s)
```

Construct a [NBInterpolatedString](#) object from a character string (null terminated array of characters)

## Parameters

|          |                             |
|----------|-----------------------------|
| <i>s</i> | String to initialize object |
|----------|-----------------------------|

**16.136.2.4 NBInterpolatedString()** [4/4]

```
NBInterpolatedString::NBInterpolatedString (
 const char * s,
 size_t n)
```

Construct a [NBInterpolatedString](#) object from a character string up to the specified amount.

## Parameters

|          |                              |
|----------|------------------------------|
| <i>s</i> | String to initialize object  |
| <i>n</i> | Number of characters to copy |

**16.136.3 Member Function Documentation****16.136.3.1 Append()**

```
NBString & NBInterpolatedString::Append (
 const char * str,
 size_t len)
```

Append a string to an existing [NBString](#) object.

Append a string to an existing [NBString](#) object

## Parameters

|            |                                |
|------------|--------------------------------|
| <i>str</i> | String to append               |
| <i>len</i> | Number of characters to append |

**Returns**

The updated [NBString](#) object

**16.136.3.2 FdAppend()**

```
NBString & NBInterpolatedString::FdAppend (
 int fd,
 size_t len)
```

Append from a file descriptor to an existing [NBString](#) object.

Append from a file descriptor to an existing [NBString](#) object

**Parameters**

|            |                                |
|------------|--------------------------------|
| <i>fd</i>  | File descriptor to append from |
| <i>len</i> | Number of characters to append |

**Returns**

The updated [NBString](#) object

**16.136.3.3 Interpolate()**

```
NBInterpolatedString & NBInterpolatedString::Interpolate ()
```

Perform a string interpolation in-place. This allows [NBStrings](#) to easily access the NetBurner Config System, as in:

```
NBInterpolatedString test("My MAC: {Sys.NetIf.Ethernet0.MAC} \n");
test.Interpolate();
```

**See also**

[NBString::Interpolate\(\)](#)

The documentation for this class was generated from the following file:

- nbstring.h

**16.137 NBString Class Reference**

Lightweight alternative to C++ CString class.

```
#include <nbstring.h>
```

Inherited by [NBInterpolatedString](#), [String](#), [String](#), [String](#), and [String](#).

**Public Member Functions**

- **NBString ()**  
*Construct a [NBString](#) object.*
- **NBString (const [NBString](#) &str)**  
*Construct a new [NBString](#) object from an existing [NBString](#) object.*
- **NBString (const [NBString](#) &str, size\_t pos, size\_t len=npos)**  
*Construct a new [NBString](#) object from a substring of an existing [NBString](#) object.*
- **NBString (const char \*s)**  
*Construct a [NBString](#) object from a character string (null terminated array of characters)*
- **NBString (const char \*s, size\_t n)**  
*Construct a [NBString](#) object from a character string up to the specified amount.*
- **NBString (const [JsonRef](#) &jr)**  
*Construct a [NBString](#) object using the current string referenced by a [JsonRef](#). This allows for using Large Strings without an intermediate malloc in the [JsonParser](#).*

- **~NSString** ()  
*NSString destructor.*
- const char \* **c\_str** () const  
*Method to pass a NSString as a constant char \*.*
- **NSString substr** (size\_t pos=0, size\_t len=npos) const  
*Creates a new NSString object from a substring of the existing NSString object.*
- int **compare** (const NSString &str) const  
*Compares two NSString objects.*
- int **compare** (const char \*s) const  
*Compares the NSString object to a character string.*
- size\_t **find** (const NSString str, size\_t pos=0) const  
*Find a substring within a NSString object.*
- size\_t **find** (const char \*str, size\_t pos=0) const  
*Find a substring within a character string.*
- size\_t **find** (const char \*s, size\_t pos, size\_t n) const  
*Find a substring within a character string.*
- size\_t **find** (char c, size\_t pos=0) const  
*Find the first occurrence of a character within the NSString object.*
- size\_t **replace** (const char \*findStr, char rep, size\_t startPos=0, size\_t end=0)  
*Replace all occurrences of all "<findStr>" characters within the NSString object.*
- size\_t **replace** (char c, char rep, size\_t startPos=0, size\_t end=0)  
*Replace all occurrences of all "<findStr>" characters within the NSString object.*
- void **shrink** (int n)  
*Reduces the length of the string by n.*
- size\_t **size** () const  
*Returns the current size of memory allocated for the string.*
- size\_t **length** () const  
*Returns the length of the string.*
- void **clear** ()  
*Clear a NSString object and free allocated memory.*
- bool **empty** () const  
*Check if a string is empty.*
- **NSString & Append** (const char \*str, size\_t len)  
*Append a string to an existing NSString object.*
- **NSString & FdAppend** (int fd, size\_t len)  
*Append from a file descriptor to an existing NSString object.*
- **NSString & Reserve** (size\_t len)  
*Reserve an additional buffer amount.*
- int **vsprintf** (const char \*format, va\_list &vl)  
*Print to a string with formatting.*
- int **sprintf** (const char \*format,...)  
*sprintf to a string with formatting to a character array*
- int **sprintf** (NSString const format,...)  
*sprintf to a string with formatting as a NSString object*
- int **isprintf** (const char \*format,...)  
*isprintf (integer) to a string with formatting to a character array*
- int **isprintf** (NSString const format,...)  
*isprintf (integer) to a string with formatting to a NSString object*
- int **stoi** () const  
*Parse the string and convert to an integer number.*
- long **stol** () const

- Parse the string value and convert to a long integer number.*
- unsigned int `stoui ()` const
  - Parse the string value and convert to a const integer number.*
- unsigned long `stoul ()` const
  - Parse the string value and convert to a const unsigned long integer number.*
- double `stod ()` const
  - Parse the string value and convert to a const double float number.*
- `IPADDR to_ipaddr ()` const
  - Parse the string value and convert to an IPADDR IP address.*
- `size_t copy (char *s, size_t len, size_t pos=0)` const
  - Copy a substring, does not null terminate.*
- `size_t strcpy (char *s, size_t len, size_t pos=0)` const
  - Copy a substring, with null termination.*
- bool `Interpolate (NBString &dest)`
  - Perform a string interpolation and place the finished interpolation in the Destination string. This allows NBStrings to easily access the NetBurner Config System, as in:*
- bool `Interpolate ()`
  - Perform an in-place string interpolation. This allows NBStrings to easily access the NetBurner Config System, as in:*

## Friends

- void `swap (NBString &x, NBString &y)`
  - Swaps the contents of two NBString objects.*

## 16.137.1 Detailed Description

Lightweight alternative to C++ CString class.

## 16.137.2 Constructor & Destructor Documentation

### 16.137.2.1 NBString() [1/5]

```
NBString::NBString (
 const NBString & str)
```

Construct a new NBString object from an existing NBString object.

#### Parameters

|            |                          |
|------------|--------------------------|
| <i>str</i> | Existing NBString object |
|------------|--------------------------|

### 16.137.2.2 NBString() [2/5]

```
NBString::NBString (
 const NBString & str,
 size_t pos,
 size_t len = npos)
```

Construct a new NBString object from a substring of an existing NBString object.

#### Parameters

|            |                           |
|------------|---------------------------|
| <i>str</i> | Existing NBString object  |
| <i>pos</i> | Starting position to copy |
| <i>len</i> | Ending position to copy   |

### 16.137.2.3 NSString() [3/5]

```
NSString::NSString (
 const char * s)
```

Construct a [NSString](#) object from a character string (null terminated array of characters)

#### Parameters

|          |                             |
|----------|-----------------------------|
| <i>s</i> | String to initialize object |
|----------|-----------------------------|

### 16.137.2.4 NSString() [4/5]

```
NSString::NSString (
 const char * s,
 size_t n)
```

Construct a [NSString](#) object from a character string up to the specified amount.

#### Parameters

|          |                              |
|----------|------------------------------|
| <i>s</i> | String to initialize object  |
| <i>n</i> | Number of characters to copy |

### 16.137.2.5 NSString() [5/5]

```
NSString::NSString (
 const JsonRef & jr)
```

Construct a [NSString](#) object using the current string referenced by a [JsonRef](#). This allows for using Large Strings without an intermediate malloc in the JsonParser.

#### Parameters

|           |                                              |
|-----------|----------------------------------------------|
| <i>jr</i> | <a href="#">JsonRef</a> to initialize object |
|-----------|----------------------------------------------|

## 16.137.3 Member Function Documentation

### 16.137.3.1 Append()

```
NSString & NSString::Append (
 const char * str,
 size_t len)
```

Append a string to an existing [NSString](#) object.

Append a string to an existing [NSString](#) object

#### Parameters

|            |                                |
|------------|--------------------------------|
| <i>str</i> | String to append               |
| <i>len</i> | Number of characters to append |

#### Returns

The updated [NSString](#) object

**16.137.3.2 c\_str()**

```
const char * NBString::c_str () const
```

Method to pass a [NBString](#) as a constant char \*.

Will cause an allocation if it is pointing to a constant string longer than the currently allocated length.

**Returns**

A null terminated character array representing the string

**16.137.3.3 clear()**

```
void NBString::clear ()
```

Clear a [NBString](#) object and free allocated memory.

Clear a [NBString](#) object and free allocated memory

**16.137.3.4 compare() [1/2]**

```
int NBString::compare (
 const char * s) const
```

Compares the [NBString](#) object to a character string.

**Parameters**

|          |                   |
|----------|-------------------|
| <i>s</i> | String to compare |
|----------|-------------------|

**Return values**

|           |                                                        |
|-----------|--------------------------------------------------------|
| <i>0</i>  | If identical                                           |
| <i>-1</i> | If <a href="#">NBString</a> object is less than str    |
| <i>1</i>  | If <a href="#">NBString</a> object is greater than str |

**16.137.3.5 compare() [2/2]**

```
int NBString::compare (
 const NBString & str) const
```

Compares two [NBString](#) objects.

**Parameters**

|            |                                            |
|------------|--------------------------------------------|
| <i>str</i> | <a href="#">NBString</a> object to compare |
|------------|--------------------------------------------|

**Return values**

|           |                                                        |
|-----------|--------------------------------------------------------|
| <i>0</i>  | If identical                                           |
| <i>-1</i> | If <a href="#">NBString</a> object is less than str    |
| <i>1</i>  | If <a href="#">NBString</a> object is greater than str |

**16.137.3.6 copy()**

```
size_t NBString::copy (
 char * s,
 size_t len,
 size_t pos = 0) const
```

Copy a substring, does not null terminate.



## Parameters

|            |                           |
|------------|---------------------------|
| <i>s</i>   | Destination               |
| <i>len</i> | Size of destination       |
| <i>pos</i> | Starting position of copy |

## Returns

Number of bytes copied

**16.137.3.7 empty()**

```
bool NSString::empty () const
```

Check if a string is empty.

Check if a [NSString](#) object is empty

## Return values

|              |                      |
|--------------|----------------------|
| <i>true</i>  | String is null       |
| <i>false</i> | String contains data |

**16.137.3.8 FdAppend()**

```
NSString & NSString::FdAppend (
 int fd,
 size_t len)
```

Append from a file descriptor to an existing [NSString](#) object.

Append from a file descriptor to an existing [NSString](#) object

## Parameters

|            |                                |
|------------|--------------------------------|
| <i>fd</i>  | File descriptor to append from |
| <i>len</i> | Number of characters to append |

## Returns

The updated [NSString](#) object

**16.137.3.9 find() [1/4]**

```
size_t NSString::find (
 char c,
 size_t pos = 0) const
```

Find the first occurrence of a character within the [NSString](#) object.

## Parameters

|            |                                   |
|------------|-----------------------------------|
| <i>c</i>   | Character to search for           |
| <i>pos</i> | Starting position to start search |

## Return values

|               |                        |
|---------------|------------------------|
| <i>&gt;=0</i> | Position in string     |
| <i>-1</i>     | If character not found |

**16.137.3.10 find() [2/4]**

```
size_t NBString::find (
 const char * s,
 size_t pos,
 size_t n) const
```

Find a substring within a character string.

**Parameters**

|            |                                         |
|------------|-----------------------------------------|
| <i>s</i>   | String to search                        |
| <i>pos</i> | Starting position to start search       |
| <i>n</i>   | Number of characters that have to match |

**Return values**

|          |                             |
|----------|-----------------------------|
| $\geq 0$ | Position in searched string |
| -1       | If substring not found      |

**16.137.3.11 find() [3/4]**

```
size_t NBString::find (
 const char * str,
 size_t pos = 0) const
```

Find a substring within a character string.

**Parameters**

|            |                                   |
|------------|-----------------------------------|
| <i>str</i> | String to search                  |
| <i>pos</i> | Starting position to start search |

**Return values**

|          |                             |
|----------|-----------------------------|
| $\geq 0$ | Position in searched string |
| -1       | If substring not found      |

**16.137.3.12 find() [4/4]**

```
size_t NBString::find (
 const NBString str,
 size_t pos = 0) const
```

Find a substring within a [NBString](#) object.

**Parameters**

|            |                                   |
|------------|-----------------------------------|
| <i>str</i> | NString object to search          |
| <i>pos</i> | Starting position to start search |

**Return values**

|          |                             |
|----------|-----------------------------|
| $\geq 0$ | Position in searched string |
| -1       | If substring not found      |

**16.137.3.13 Interpolate() [1/2]**

```
bool NBString::Interpolate () [inline]
```

Perform an in-place string interpolation. This allows NBStrings to easily access the NetBurner Config System, as in:

```
NBString test("My MAC: {Sys.NetIf.Ethernet0.MAC} \n");
test.Interpolate();
```

**See also**

[NBInterpolatedString](#)

**Returns**

Whether the interpolation was successful

**16.137.3.14 Interpolate() [2/2]**

```
bool NBString::Interpolate (
 NBString & dest)
```

Perform a string interpolation and place the finished interpolation in the Destination string. This allows NBStrings to easily access the NetBurner Config System, as in:

```
NBString test("My MAC: {Sys.NetIf.Ethernet0.MAC} \n");
NBString interpolated;
test.Interpolate(interpolated);
```

**See also**

[NBInterpolatedString](#)

**Parameters**

|             |                    |
|-------------|--------------------|
| <i>dest</i> | Destination string |
|-------------|--------------------|

**Returns**

Whether the interpolation was successful

**16.137.3.15 length()**

```
size_t NBString::length () const
```

Returns the length of the string.

**Returns**

The length of the string

**16.137.3.16 replace() [1/2]**

```
size_t NBString::replace (
 char c,
 char rep,
 size_t startPos = 0,
 size_t end = 0)
```

Replace all occurrences of all "<findStr>" characters within the [NBString](#) object.

**Parameters**

|                 |                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------|
| <i>c</i>        | Character to be replaced within the <a href="#">NBString</a>                              |
| <i>rep</i>      | Character to replace with                                                                 |
| <i>startPos</i> | Starting position to start replacing                                                      |
| <i>end</i>      | Replace all matching characters before this position, 0 = end of <a href="#">NBString</a> |

## Return values

|              |                                |
|--------------|--------------------------------|
| <i>&gt;0</i> | Number of Replaced Characters  |
| <i>-1</i>    | If no matching character found |

**16.137.3.17 replace()** [2/2]

```
size_t NSString::replace (
 const char * findStr,
 char rep,
 size_t startPos = 0,
 size_t end = 0)
```

Replace all occurrences of all "<findStr>" characters within the [NSString](#) object.

## Parameters

|                 |                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------|
| <i>findStr</i>  | String of characters to be replaced within the <a href="#">NSString</a>                   |
| <i>rep</i>      | Character to replace with                                                                 |
| <i>startPos</i> | Starting position to start replacing                                                      |
| <i>end</i>      | Replace all matching characters before this position, 0 = end of <a href="#">NSString</a> |

## Return values

|              |                                |
|--------------|--------------------------------|
| <i>&gt;0</i> | Number of Replaced Characters  |
| <i>-1</i>    | If no matching character found |

**16.137.3.18 Reserve()**

```
NSString & NSString::Reserve (
 size_t len)
```

Reserve an additional buffer amount.

Reserve an additional buffer amount to prepare for a series of Appends

## Parameters

|            |                                                     |
|------------|-----------------------------------------------------|
| <i>len</i> | Number of additional characters to prepare to store |
|------------|-----------------------------------------------------|

## Returns

The updated [NSString](#) object

**16.137.3.19 sprintf()** [1/2]

```
int NSString::sprintf (
 const char * format,
 ...)
```

isprintf (integer) to a string with formatting to a character array

## Parameters

|               |                         |
|---------------|-------------------------|
| <i>format</i> | printf style formatting |
|---------------|-------------------------|

**16.137.3.20 siprintf()** [2/2]

```
int NSString::siprintf (
 NSString const format,
 ...)
```

isprintf (integer) to a string with formatting to a [NSString](#) object

**Parameters**

|               |                         |
|---------------|-------------------------|
| <i>format</i> | printf style formatting |
|---------------|-------------------------|

**16.137.3.21 size()**

```
size_t NSString::size () const
```

Returns the current size of memory allocated for the string.

**Returns**

Total size of the object in bytes

**16.137.3.22 sprintf()** [1/2]

```
int NSString::sprintf (
 const char * format,
 ...)
```

sprintf to a string with formatting to a character array

**Parameters**

|               |                         |
|---------------|-------------------------|
| <i>format</i> | printf style formatting |
|---------------|-------------------------|

**16.137.3.23 sprintf()** [2/2]

```
int NSString::sprintf (
 NSString const format,
 ...)
```

sprintf to a string with formatting as a [NSString](#) object

**Parameters**

|               |                         |
|---------------|-------------------------|
| <i>format</i> | printf style formatting |
|---------------|-------------------------|

**16.137.3.24 stod()**

```
double NSString::stod () const
```

Parse the string value and convert to a const double float number.

**Returns**

const double float value

**16.137.3.25 stoi()**

```
int NSString::stoi () const
```

Parse the string and convert to an integer number.

**Returns**

integer value

**16.137.3.26 stol()**

```
long NBString::stol () const
```

Parse the string value and convert to a long integer number.

**Returns**

long integer value

**16.137.3.27 stoui()**

```
unsigned int NBString::stoui () const
```

Parse the string value and convert to a const integer number.

**Returns**

unsigned integer value

**16.137.3.28 stoul()**

```
unsigned long NBString::stoul () const
```

Parse the string value and convert to a const unsigned long integer number.

**Returns**

unsigned long integer value

**16.137.3.29 strcpy()**

```
size_t NBString::strcpy (
 char * s,
 size_t len,
 size_t pos = 0) const
```

Copy a substring, with null termination.

**Parameters**

|            |                           |
|------------|---------------------------|
| <i>s</i>   | Destination               |
| <i>len</i> | Size of destination       |
| <i>pos</i> | Starting position of copy |

**Returns**

Number of bytes copied

**16.137.3.30 substr()**

```
NBString NBString::substr (
 size_t pos = 0,
 size_t len = npos) const
```

Creates a new [NBString](#) object from a substring of the existing [NBString](#) object.

**Parameters**

|            |                                |
|------------|--------------------------------|
| <i>pos</i> | Starting position of substring |
| <i>len</i> | Number of characters to copy   |

**Returns**

A [NBString](#) object

**16.137.3.31 to\_ipaddr()**

```
IPADDR NBString::to_ipaddr () const
```

Parse the string value and convert to an IPADDR IP address.

**Returns**

const double float value

**16.137.3.32 vsiprintf()**

```
int NBString::vsiprintf (
 const char * format,
 va_list & vl)
```

Print to a string with formatting.

**Parameters**

|               |                         |
|---------------|-------------------------|
| <i>format</i> | printf style formatting |
| <i>vl</i>     | Variable argument list  |

**16.137.4 Friends And Related Symbol Documentation****16.137.4.1 swap**

```
void swap (
 NBString & x,
 NBString & y) [friend]
```

Swaps the contents of two [NBString](#) objects.

Useful for optimizing sorts. Friend function that takes two [NBString](#) objects as parameters and swaps their contents.

The documentation for this class was generated from the following file:

- nbstring.h

**16.138 NTPServer Class Reference**

```
#include <NTPServer.h>
```

**Classes**

- class [NtpTimeStamp](#)

**Public Member Functions**

- [NTPServer](#) ([IEEE1588Timer](#) &timer, GPSChip &gps, uint16\_t port=123)
- bool [start](#) ()
- void [stop](#) ()
- bool [isRunning](#) () const
- uint16\_t [getPort](#) () const

**Static Public Member Functions**

- static void [serverTask](#) (void \*param)

**16.138.1 Detailed Description**

[NTPServer](#) class - Encapsulates NTP server functionality

This class provides a complete NTP server implementation that uses [IEEE1588Timer](#) for precise timestamping and GPS synchronization. It handles NTP packet processing and response generation.

## 16.138.2 Constructor & Destructor Documentation

### 16.138.2.1 NTPServer()

```
NTPServer::NTPServer (
 IEEE1588Timer & timer,
 GPSChip & gps,
 uint16_t port = 123)
```

Constructor

#### Parameters

|              |                                                                     |
|--------------|---------------------------------------------------------------------|
| <i>timer</i> | Reference to <a href="#">IEEE1588Timer</a> for precise timestamping |
| <i>gps</i>   | Reference to GPSChip for GPS synchronization status                 |
| <i>port</i>  | NTP server port (default 123)                                       |

## 16.138.3 Member Function Documentation

### 16.138.3.1 getPort()

```
uint16_t NTPServer::getPort () const [inline]
```

Get server port

#### Returns

NTP server port number

### 16.138.3.2 isRunning()

```
bool NTPServer::isRunning () const [inline]
```

Get server running status

#### Returns

true if server is running

### 16.138.3.3 serverTask()

```
void NTPServer::serverTask (
 void * param) [static]
```

Static task wrapper for RTOS

#### Parameters

|              |                                               |
|--------------|-----------------------------------------------|
| <i>param</i> | Pointer to <a href="#">NTPServer</a> instance |
|--------------|-----------------------------------------------|

### 16.138.3.4 start()

```
bool NTPServer::start ()
```

Start the NTP server Registers UDP FIFO and creates server task

#### Returns

true if server started successfully



**16.138.3.5 stop()**

```
void NTPServer::stop ()
```

Stop the NTP server Cleanup method for future use

The documentation for this class was generated from the following files:

- NTPServer.h
- NTPServer.cpp

**16.139 NTPServer::NtpTimeStamp Class Reference**

```
#include <NTPServer.h>
```

**Public Member Functions**

- bool [SetFrom](#) (uint32\_t tt, uint32\_t From1588)
- [NtpTimeStamp](#) ()
- [NtpTimeStamp](#) & [operator=](#) (UDPPacket &upkt)
- [NtpTimeStamp](#) & [operator=](#) (const [NtpTimeStamp](#) &src)

**Static Public Member Functions**

- static [NtpTimeStamp Now](#) ()

**Public Attributes**

- beuint32\_t **sec**  
*Seconds since NTP epoch (big endian)*
- beuint32\_t **frac**  
*Fractional seconds (big endian)*

**16.139.1 Detailed Description**

[NtpTimeStamp](#) struct - Represents an NTP timestamp

NTP timestamps are 64-bit values with 32-bit seconds and 32-bit fraction The epoch is January 1, 1900 00:00:00 UTC

**16.139.2 Constructor & Destructor Documentation****16.139.2.1 NtpTimeStamp()**

```
NTPServer::NtpTimeStamp::NtpTimeStamp () [inline]
```

Default constructor - initializes to zero

**16.139.3 Member Function Documentation****16.139.3.1 Now()**

```
NTPServer::NtpTimeStamp NTPServer::NtpTimeStamp::Now () [static]
```

Create timestamp for current time

Returns

[NtpTimeStamp](#) representing current time

**16.139.3.2 operator=() [1/2]**

```
NTPServer::NtpTimeStamp & NTPServer::NtpTimeStamp::operator= (
 const NtpTimeStamp & src)
```

Copy assignment operator

## Parameters

|            |                          |
|------------|--------------------------|
| <i>src</i> | Source timestamp to copy |
|------------|--------------------------|

## Returns

Reference to this timestamp

**16.139.3.3 operator=()** [2/2]

```
NTPServer::NtpTimeStamp & NTPServer::NtpTimeStamp::operator= (
 UDPacket & upkt)
```

Extract timestamp from UDP packet Uses packet timing information to create precise timestamp

## Parameters

|             |                         |
|-------------|-------------------------|
| <i>upkt</i> | Reference to UDP packet |
|-------------|-------------------------|

## Returns

Reference to this timestamp

**16.139.3.4 SetFrom()**

```
bool NTPServer::NtpTimeStamp::SetFrom (
 uint32_t tt,
 uint32_t From1588)
```

Set timestamp from OS tick and 1588 timer values

## Parameters

|                 |                  |
|-----------------|------------------|
| <i>tt</i>       | OS tick count    |
| <i>From1588</i> | 1588 timer value |

## Returns

true if timestamp was set successfully, false if no valid GPS time

The documentation for this class was generated from the following files:

- NTPServer.h
- NTPServer.cpp

**16.140 ParsedJsonDataSet Class Reference**

A class to create, read, and modify a JSON object.

```
#include <json_lexer.h>
```

Inherits [buffer\\_object](#).

Inherited by [AcmeServletBuffer](#), and [JsonLexerFDAdapter](#).

**Public Member Functions****Parsing**

*Ingesting raw JSON data from some source, or resetting. Enables compatibility with [buffer\\_object](#) and [WebClient](#).*

- void **EnableLargeStrings** (bool b)  
*Call to allow allocation of strings larger than 1500 bytes.*
- virtual int **WriteData** (const unsigned char \*pCopyFrom, int numBytes)  
*Writes the passed in data to the JSON data set.*
- virtual int **ReadFrom** (int fd)  
*Reads in data from the specified file descriptor and parses it into the JSON data set.*
- bool **CopyObject** (ParsedJsonDataSet &src\_set)  
*Copies the provided JSON data set into the current one.*
- void **ClearObject** ()  
*Clears all data from the object and resets it to its initial state.*

### Querying the Data Structure

*Allows easy retrieval of entities by array index or object key name*

#### Example

```
printf("User %d name %s",
 (int)myJsonRef("users")[0]("id"),
 (const char *)myJsonRef("users")[0]("name")
);
```

- **JsonRef operator[]** (int i)  
*Get a [JsonRef](#) from an array by numerical index.*
- **JsonRef operator()** (const char \*name)  
*Get a [JsonRef](#) representing any entity from a parent object by key name.*
- **JsonRef name** (const char \*name)  
*Get a [JsonRef](#) representing any entity from a parent object by key name.*
- **JsonRef object** (const char \*name)  
*Get a [JsonRef](#) representing an object from a parent object by key name.*

### Traversing the Data Structure

*Used for scanning through the JSON elements*

- **JsonRef start** ()  
*Begin traversal: check for ref validity and reset the position.*
- **JsonRef next** ()  
*Traverse: check for ref validity and increment the position.*
- **json\_primitive\_type GetFirst** ()  
*Get the first element of the JSON data set.*
- **json\_primitive\_type GetNext** ()  
*Get the element at the next position of the JSON data set.*
- **json\_primitive\_type GetCurrent** ()  
*Get the element at the current position of the JSON data set.*
- **json\_primitive\_type GetRawCurrent** ()  
*Get the element at the current position of the JSON data set, including non-public types.*
- **json\_primitive\_type GetNextName** ()  
*Iterates the current element of the JSON data set until it finds one of type NAME.*
- **json\_primitive\_type GetNextObject** ()  
*Get the next element of type name that exists in the current OBJECT after the current position.*
- **json\_primitive\_type GetNextArray** ()  
*Iterates the current element of the JSON data set until it finds one of type BEGIN\_ARRAY.*
- **json\_primitive\_type GetNextNameInCurrentObject** ()  
*Get the next element of type NAME that exists in the current OBJECT after the current position.*
- **json\_primitive\_type GetNextNameInCurrentArray** ()  
*Get the next element of type NAME that exists in the current ARRAY after the current position.*
- **json\_primitive\_type GetNextNumberInCurrentArray** ()  
*Get the next element of type NUMBER that exists in the current ARRAY after the current position.*
- **json\_primitive\_type GetNextStringInCurrentArray** ()  
*Get the next element of type STRING that exists in the current ARRAY after the current position.*
- **json\_primitive\_type GetNextBoolInCurrentArray** ()  
*Get the next element of type TRUE\_EL that exists in the current ARRAY after the current position.*

- [json\\_primitive\\_type GetNextObjectInCurrentArray \(\)](#)  
*Get the next element of type BEGIN\_OBJECT that exists in the current ARRAY after the current position.*
- [json\\_primitive\\_type SkipCurrentValue \(\)](#)  
*Skips over the current value, and get the next element. If called inside an ARRAY or OBJECT, it will walk to the end of the ARRAY or OBJECT and return the next element it finds. If it reaches the end of the ARRAY or OBJECT before it finds a value, it will return NOTFOUND.*
- void [ResetPosition \(\)](#)  
*Resets the parser position to the beginning of the JSON data set.*
- [JsonRef GetParsePosition \(\)](#)  
*Gets the current parse position object for the JSON data set.*
- [JsonRef SetParsePosition \(JsonRef pos\)](#)  
*Sets the current parse position object for the JSON data set.*

### Advanced Finding and Returning

Advanced functions for scanning and returning specific objects. All of these can return NOTFOUND.

#### Example

*This function works with dotted string notation to find sub elements:*

```
{
 "SUCCESS": true,
 "ACTION": null,
 "JDATA": {
 "foo": 7,
 "bar": {
 "b1": "Sb1",
 "b2": {
 "s1": "Sub1",
 "s2": "Sub2",
 "s3": "Sub3",
 "s4": {
 "Sub4a": "Test4a",
 "Sub4b": "Test4b"
 },
 "s5": "Sub5",
 "s6": "Sub6"
 },
 "b3": true,
 "b4": null
 },
 "bogus": "Bstring",
 "nubog": 1234.57,
 "av": [
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10
],
 "aftaav": true
 }
}
```

*You could return the "Sub4b" element's "Test4b" string with:*

```
myParsedJsonDataSet.FindFullName("JDATA.bar.b2.s4.Sub4b");
```

- [json\\_primitive\\_type FindFullName \(const char \\*name\)](#)  
*Find the element in the data set with the given name and move the parser to the next element. This searches a full name path, such as ob1.ob2.ob3.ob4.*
- [json\\_primitive\\_type FindFullAtName \(const char \\*name\)](#)  
*Find the element in the data set with the given name and move the parser to that element. This searches a full name path, such as ob1.ob2.ob3.ob4.*
- [json\\_primitive\\_type FindElementAfterName \(const char \\*name\)](#)  
*Finds name in current object points at element after name. This only supports simple, single element names. This searches from the current position to the end.*
- [json\\_primitive\\_type FindGlobalElementAfterName \(const char \\*name\)](#)  
*Finds name in current object points at element after name. This only supports simple, single element names. It starts the search from the parser's current position, and if it doesn't find it, it will start the search over from the beginning.*

- [json\\_primitive\\_type FindElementAfterNameInCurrentObject](#) (const char \*name)  
*Looks for elements with the current name in the current OBJECT only. Does not search sub objects.*
- [json\\_primitive\\_type FindElementAfterNameInCurrentArray](#) (const char \*name)  
*Looks for elements with the current name in the current ARRAY only. Does not search sub arrays.*
- bool [CurrentBool](#) ()  
*Returns true if the current element a TRUE\_EL primitive type.*
- bool [PermissiveCurrentBool](#) ()  
*Returns true if the current element a TRUE\_EL primitive type, is "True", "true", or is a non-zero number.*
- double [CurrentNumber](#) ()  
*Get the number value of the current element.*
- const char \* [CurrentString](#) ()  
*Get the string value of the current element.*
- int [CopyCurrentString](#) (char \*buf, int buflen)  
*Copy the string value of the current element.*
- int [CopyCurrentString](#) (NBString &nbs)  
*Copy the string value of the current element.*
- const char \* [CurrentName](#) ()  
*Get the name of the current element.*
- bool [FindFullNameBoolean](#) (const char \*name)  
*Find the boolean value of a given element.*
- bool [FindGlobalBoolean](#) (const char \*name)  
*Find the boolean value of a given element. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.*
- bool [FindBoolean](#) (const char \*name)  
*Find the boolean value of the element after the element with the given name.*
- bool [FindBooleanInCurentObject](#) (const char \*name)  
*Find the boolean value of the element within the current object. Does not search sub-objects.*
- bool [FindFullNamePermissiveBoolean](#) (const char \*name)  
*Find the permissive boolean value of a given element.*
- bool [FindGlobalPermissiveBoolean](#) (const char \*name)  
*Find the permissive boolean value of a given element. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.*
- bool [FindPermissiveBoolean](#) (const char \*name)  
*Find the permissive boolean value of the element after the element with the provided name.*
- bool [FindPermissiveBooleanInCurentObject](#) (const char \*name)  
*Find the permissive boolean value of the element with the given name in the current object. Does not search sub-objects.*
- const char \* [FindFullNameString](#) (const char \*name)  
*Find the string value of the element with the given name in the current object.*
- int [FindAndCopyFullNameString](#) (const char \*name, char \*buf, int buflen)  
*Find and copy the string value of the element with the given name in the current object. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.*
- int [FindAndCopyFullNameString](#) (const char \*name, NBString &nbs)  
*Find and copy the string value of the element with the given name in the current object. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.*
- const char \* [FindGlobalString](#) (const char \*name)  
*Find the string value of a given element. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.*
- int [FindAndCopyGlobalString](#) (const char \*name, char \*buf, int buflen)  
*Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names. It then copies the found string to the provided buffer. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.*
- int [FindAndCopyGlobalString](#) (const char \*name, NBString &nbs)  
*Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names. It then copies the found string to the provided buffer. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.*
- const char \* [FindString](#) (const char \*name)  
*Find the string value of the element after the element with the given name.*
- int [FindAndCopyString](#) (const char \*name, char \*buf, int buflen)

*Find and copy the string value of the element with the given name. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.*

- int [FindAndCopyString](#) (const char \*name, NBString &nbs)

*Find and copy the string value of the element with the given name. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.*

- const char \* [FindStringInCurentObject](#) (const char \*name)

*Find the string value of the element with the given name in the current object. Does not search sub-objects.*

- double [FindFullNameNumber](#) (const char \*name)

*Find the number value of the element with the given name in the current object.*

- double [FindGlobalNumber](#) (const char \*name)

*Find the number value of a given element. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.*

- double [FindNumber](#) (const char \*name)

*Find the number value of the element after the element with the given name.*

- double [FindNumberInCurentObject](#) (const char \*name)

*Find the number value of the element with the given name in the current object. Does not search sub-objects.*

- bool [FindGlobalObject](#) (const char \*name)

*Find the OBJECT with the given name. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.*

- bool [FindObject](#) (const char \*name)

*Find the OBJECT of the element after the element with the given name.*

- bool [FindObjectInCurentObject](#) (const char \*name)

*Find the OBJECT of the element with the given name in the current OBJECT. Does not search sub-objects.*

## Utility

*Ingesting raw JSON data from some source, or resetting*

- void **DumpState** ()  
*Outputs what's in the parse tree to stdout.*
- int [PrintObject](#) (bool pretty=false)  
*Prints the JSON data set to stdout.*
- int [PrintObjectToBuffer](#) (char \*buffer, int maxlen, bool pretty=false)  
*Prints the JSON data set to a provided buffer.*
- int [PrintObjectToFd](#) (int fd, bool pretty=false)  
*Prints the JSON data set to a specified file descriptor.*
- int [PrintObjectToString](#) (NBString &s, bool pretty=false)  
*Prints the JSON data set to a NBString object.*
- int **PrintChildren** (bool pretty=false)  
*Output child objects of this [JsonRef](#) as JSON text to stdout.*
- int **PrintChildrenToFd** (int fd, bool pretty=false)  
*Output child objects of this [JsonRef](#) as JSON text to a file descriptor.*
- int **PrintChildrenToBuffer** (char \*buffer, int maxlen, bool pretty=false)  
*Output child objects of this [JsonRef](#) as JSON text to a buffer pointer.*
- int [GetPrintSize](#) (bool pretty=false)  
*Calculates how many characters the JSON data set would take to print.*

## Building

*Programmatically creating or modifying a JSON dataset*

- [ParsedJsonDataSet](#) \* **StartBuilding** ()  
*Use to start building the JSON data set.*
- [ParsedJsonDataSet](#) \* [AddObjectStart](#) (const char \*name)  
*Use to start an object in the JSON data set.*
- [ParsedJsonDataSet](#) \* [AddMyMac](#) (const char \*name)  
*Add the device MAC address to the JSON data set.*
- [ParsedJsonDataSet](#) \* [Add](#) (const char \*name, int i)  
*Add a name/value pair to the JSON data set where the value is an int.*
- [ParsedJsonDataSet](#) \* [Add](#) (const char \*name, short i)  
*Add a name/value pair to the JSON data set where the value is a short.*
- [ParsedJsonDataSet](#) \* [Add](#) (const char \*name, long i)

- Add a name/value pair to the JSON data set where the value is a long.*
- [ParsedJsonDataSet \\* Add](#) (const char \*name, unsigned int i)
- Add a name/value pair to the JSON data set where the value is an unsigned int.*
- [ParsedJsonDataSet \\* Add](#) (const char \*name, unsigned short i)
- Add a name/value pair to the JSON data set where the value is a unsigned short.*
- [ParsedJsonDataSet \\* Add](#) (const char \*name, unsigned long i)
- Add a name/value pair to the JSON data set where the value is a unsigned long.*
- [ParsedJsonDataSet \\* Add](#) (const char \*name, double d)
- Add a name/value pair to the JSON data set where the value is a double.*
- [ParsedJsonDataSet \\* Add](#) (const char \*name, const char \*str)
- Add a name/value pair to the JSON data set where the value is a string.*
- [ParsedJsonDataSet \\* Add](#) (const char \*name, bool b)
- Add a name/value pair to the JSON data set where the value is a bool.*
- [ParsedJsonDataSet \\* Add](#) (const char \*name, IPADDR4 i4)
- Add a name/value pair to the JSON data set where the value is an IPv4 address.*
- [ParsedJsonDataSet \\* Add](#) (const char \*name, const IPADDR &i)
- Add a name/value pair to the JSON data set where the value is an IP address.*
- [ParsedJsonDataSet \\* Add](#) (const char \*name, const MACADDR &ma)
- Add a name/value pair to the JSON data set where the value is a MAC address.*
- [ParsedJsonDataSet \\* AddNull](#) (const char \*name)
- Add a name/value pair to the JSON data set where the value is null.*
- [ParsedJsonDataSet \\* AddArrayStart](#) (const char \*name)
- Add an ARRAY start to the JSON data set.*
- [ParsedJsonDataSet \\* EndArray](#) ()
- Add an ARRAY end to the JSON data set.*
- [ParsedJsonDataSet \\* AddArrayElement](#) (int i)
- Add an integer value to the current array.*
- [ParsedJsonDataSet \\* AddArrayElement](#) (short i)
- Add a short value to the current array.*
- [ParsedJsonDataSet \\* AddArrayElement](#) (long i)
- Add a long value to the current array.*
- [ParsedJsonDataSet \\* AddArrayElement](#) (unsigned int i)
- Add an unsigned int value to the current array.*
- [ParsedJsonDataSet \\* AddArrayElement](#) (unsigned short i)
- Add an unsigned short value to the current array.*
- [ParsedJsonDataSet \\* AddArrayElement](#) (unsigned long i)
- Add an unsigned long value to the current array.*
- [ParsedJsonDataSet \\* AddArrayElement](#) (double d)
- Add a double to the current array.*
- [ParsedJsonDataSet \\* AddArrayElement](#) (const char \*str)
- Add a string to the current array.*
- [ParsedJsonDataSet \\* AddArrayElement](#) (bool b)
- Add a bool to the current array.*
- [ParsedJsonDataSet \\* AddArrayElement](#) (const IPADDR &i)
- Add an IP address to the current array.*
- [ParsedJsonDataSet \\* AddArrayElementArray](#) ()
- Add the start of an array element to the current array.*
- [ParsedJsonDataSet \\* AddArrayObjectStart](#) ()
- Add the start of an object element to the current array.*
- [ParsedJsonDataSet \\* AddNullArrayElement](#) ()
- Add a null element to the current array.*
- [ParsedJsonDataSet \\* EndObject](#) ()
- Add an end to the current object.*
- [ParsedJsonDataSet \\* DoneBuilding](#) ()
- Add an end JSON data set and finish building.*

## Friends

- class [JsonRef](#)

### 16.140.1 Detailed Description

A class to create, read, and modify a JSON object.

Many of the methods on [ParsedJsonDataSet](#) return a [JsonRef](#) which is a pointer to a variable or sub object inside the [ParsedJsonDataSet](#). So, once data is parsed and traversal or querying has begun most of the work will be inside the [JsonRef](#) class.

### 16.140.2 Member Function Documentation

#### 16.140.2.1 Add() [1/12]

```
ParsedJsonDataSet * ParsedJsonDataSet::Add (
 const char * name,
 bool b)
```

Add a name/value pair to the JSON data set where the value is a bool.

##### Parameters

|             |                         |
|-------------|-------------------------|
| <i>name</i> | The name of the pair.   |
| <i>b</i>    | The value for the pair. |

#### 16.140.2.2 Add() [2/12]

```
ParsedJsonDataSet * ParsedJsonDataSet::Add (
 const char * name,
 const char * str)
```

Add a name/value pair to the JSON data set where the value is a string.

##### Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>name</i> | The name of the pair.                |
| <i>str</i>  | A pointer to the value for the pair. |

#### 16.140.2.3 Add() [3/12]

```
ParsedJsonDataSet * ParsedJsonDataSet::Add (
 const char * name,
 const IPADDR & i)
```

Add a name/value pair to the JSON data set where the value is an IP address.

##### Parameters

|             |                        |
|-------------|------------------------|
| <i>name</i> | The name of the pair.  |
| <i>i</i>    | The IP address to add. |

#### 16.140.2.4 Add() [4/12]

```
ParsedJsonDataSet * ParsedJsonDataSet::Add (
 const char * name,
 const MACADDR & ma)
```

Add a name/value pair to the JSON data set where the value is a MAC address.

##### Parameters

|             |                         |
|-------------|-------------------------|
| <i>name</i> | The name of the pair.   |
| <i>ma</i>   | The MAC address to add. |



**16.140.2.5 Add()** [5/12]

```
ParsedJsonDataSet * ParsedJsonDataSet::Add (
 const char * name,
 double d)
```

Add a name/value pair to the JSON data set where the value is a double.

**Parameters**

|             |                         |
|-------------|-------------------------|
| <i>name</i> | The name of the pair.   |
| <i>d</i>    | The value for the pair. |

**16.140.2.6 Add()** [6/12]

```
ParsedJsonDataSet * ParsedJsonDataSet::Add (
 const char * name,
 int i)
```

Add a name/value pair to the JSON data set where the value is an int.

**Parameters**

|             |                         |
|-------------|-------------------------|
| <i>name</i> | The name of the pair.   |
| <i>i</i>    | The value for the pair. |

**16.140.2.7 Add()** [7/12]

```
ParsedJsonDataSet * ParsedJsonDataSet::Add (
 const char * name,
 IPADDR4 i4)
```

Add a name/value pair to the JSON data set where the value is an IPv4 address.

**Parameters**

|             |                          |
|-------------|--------------------------|
| <i>name</i> | The name of the pair.    |
| <i>i4</i>   | The IPv4 address to add. |

**16.140.2.8 Add()** [8/12]

```
ParsedJsonDataSet * ParsedJsonDataSet::Add (
 const char * name,
 long i)
```

Add a name/value pair to the JSON data set where the value is a long.

**Parameters**

|             |                         |
|-------------|-------------------------|
| <i>name</i> | The name of the pair.   |
| <i>i</i>    | The value for the pair. |

**16.140.2.9 Add()** [9/12]

```
ParsedJsonDataSet * ParsedJsonDataSet::Add (
 const char * name,
 short i)
```

Add a name/value pair to the JSON data set where the value is a short.

## Parameters

|             |                         |
|-------------|-------------------------|
| <i>name</i> | The name of the pair.   |
| <i>i</i>    | The value for the pair. |

**16.140.2.10 Add()** [10/12]

```
ParsedJsonDataSet * ParsedJsonDataSet::Add (
 const char * name,
 unsigned int i)
```

Add a name/value pair to the JSON data set where the value is an unsigned int.

## Parameters

|             |                         |
|-------------|-------------------------|
| <i>name</i> | The name of the pair.   |
| <i>i</i>    | The value for the pair. |

**16.140.2.11 Add()** [11/12]

```
ParsedJsonDataSet * ParsedJsonDataSet::Add (
 const char * name,
 unsigned long i)
```

Add a name/value pair to the JSON data set where the value is a unsigned long.

## Parameters

|             |                         |
|-------------|-------------------------|
| <i>name</i> | The name of the pair.   |
| <i>i</i>    | The value for the pair. |

**16.140.2.12 Add()** [12/12]

```
ParsedJsonDataSet * ParsedJsonDataSet::Add (
 const char * name,
 unsigned short i)
```

Add a name/value pair to the JSON data set where the value is a unsigned short.

## Parameters

|             |                         |
|-------------|-------------------------|
| <i>name</i> | The name of the pair.   |
| <i>i</i>    | The value for the pair. |

**16.140.2.13 AddArrayElement()** [1/10]

```
ParsedJsonDataSet * ParsedJsonDataSet::AddArrayElement (
 bool b)
```

Add a bool to the current array.

## Parameters

|          |                        |
|----------|------------------------|
| <i>b</i> | The bool value to add. |
|----------|------------------------|

**16.140.2.14 AddArrayElement()** [2/10]

```
ParsedJsonDataSet * ParsedJsonDataSet::AddArrayElement (
 const char * str)
```

Add a string to the current array.

**Parameters**

|            |                                              |
|------------|----------------------------------------------|
| <i>str</i> | A char* pointing to the string value to add. |
|------------|----------------------------------------------|

**16.140.2.15 AddArrayElement()** [3/10]

```
ParsedJsonDataSet * ParsedJsonDataSet::AddArrayElement (
 const IPADDR & i)
```

Add an IP address to the current array.

**Parameters**

|          |                        |
|----------|------------------------|
| <i>i</i> | The IP address to add. |
|----------|------------------------|

**16.140.2.16 AddArrayElement()** [4/10]

```
ParsedJsonDataSet * ParsedJsonDataSet::AddArrayElement (
 double d)
```

Add a double to the current array.

**Parameters**

|          |                          |
|----------|--------------------------|
| <i>d</i> | The double value to add. |
|----------|--------------------------|

**16.140.2.17 AddArrayElement()** [5/10]

```
ParsedJsonDataSet * ParsedJsonDataSet::AddArrayElement (
 int i)
```

Add an integer value to the current array.

**Parameters**

|          |                           |
|----------|---------------------------|
| <i>i</i> | The integer value to add. |
|----------|---------------------------|

**16.140.2.18 AddArrayElement()** [6/10]

```
ParsedJsonDataSet * ParsedJsonDataSet::AddArrayElement (
 long i)
```

Add a long value to the current array.

**Parameters**

|          |                        |
|----------|------------------------|
| <i>i</i> | The long value to add. |
|----------|------------------------|

**16.140.2.19 AddArrayElement()** [7/10]

```
ParsedJsonDataSet * ParsedJsonDataSet::AddArrayElement (
 short i)
```

Add a short value to the current array.

## Parameters

|          |                         |
|----------|-------------------------|
| <i>i</i> | The short value to add. |
|----------|-------------------------|

**16.140.2.20 AddArrayElement()** [8/10]

```
ParsedJsonDataSet * ParsedJsonDataSet::AddArrayElement (
 unsigned int i)
```

Add an unsigned int value to the current array.

## Parameters

|          |                                |
|----------|--------------------------------|
| <i>i</i> | The unsigned int value to add. |
|----------|--------------------------------|

**16.140.2.21 AddArrayElement()** [9/10]

```
ParsedJsonDataSet * ParsedJsonDataSet::AddArrayElement (
 unsigned long i)
```

Add an unsigned long value to the current array.

## Parameters

|          |                                 |
|----------|---------------------------------|
| <i>i</i> | The unsigned long value to add. |
|----------|---------------------------------|

**16.140.2.22 AddArrayElement()** [10/10]

```
ParsedJsonDataSet * ParsedJsonDataSet::AddArrayElement (
 unsigned short i)
```

Add an unsigned short value to the current array.

## Parameters

|          |                                  |
|----------|----------------------------------|
| <i>i</i> | The unsigned short value to add. |
|----------|----------------------------------|

**16.140.2.23 AddArrayStart()**

```
ParsedJsonDataSet * ParsedJsonDataSet::AddArrayStart (
 const char * name)
```

Add an ARRAY start to the JSON data set.

## Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>name</i> | The name of the array to start. |
|-------------|---------------------------------|

**16.140.2.24 AddMyMac()**

```
ParsedJsonDataSet * ParsedJsonDataSet::AddMyMac (
 const char * name)
```

Add the device MAC address to the JSON data set.

## Parameters

|             |                                                   |
|-------------|---------------------------------------------------|
| <i>name</i> | The name of the field containing the MAC address. |
|-------------|---------------------------------------------------|

**16.140.2.25 AddNull()**

```
ParsedJsonDataSet * ParsedJsonDataSet::AddNull (
 const char * name)
```

Add a name/value pair to the JSON data set where the value is null.

**Parameters**

|             |                       |
|-------------|-----------------------|
| <i>name</i> | The name of the pair. |
|-------------|-----------------------|

**16.140.2.26 AddObjectStart()**

```
ParsedJsonDataSet * ParsedJsonDataSet::AddObjectStart (
 const char * name)
```

Use to start an object in the JSON data set.

**Parameters**

|             |                                  |
|-------------|----------------------------------|
| <i>name</i> | The name of the object to start. |
|-------------|----------------------------------|

**16.140.2.27 CopyCurrentString() [1/2]**

```
int ParsedJsonDataSet::CopyCurrentString (
 char * buf,
 int buflen) [inline]
```

Copy the string value of the current element.

**Returns**

The length of the current element if it is a STRING, ALLOC\_STRING, or STRING\_CHUNK. Otherwise it returns a 0.

**16.140.2.28 CopyCurrentString() [2/2]**

```
int ParsedJsonDataSet::CopyCurrentString (
 NBString & nbs) [inline]
```

Copy the string value of the current element.

**Returns**

The length of the current element if it is a STRING, ALLOC\_STRING, or STRING\_CHUNK. Otherwise it returns a 0.

**16.140.2.29 CopyObject()**

```
bool ParsedJsonDataSet::CopyObject (
 ParsedJsonDataSet & src_set)
```

Copies the provided JSON data set into the current one.

**Return values**

|              |                     |
|--------------|---------------------|
| <i>true</i>  | The copy succeeded. |
| <i>false</i> | The copy failed.    |

**16.140.2.30 CurrentBool()**

```
bool ParsedJsonDataSet::CurrentBool () [inline]
```

Returns true if the current element a TRUE\_EL primitive type.

## Return values

|              |                                                         |
|--------------|---------------------------------------------------------|
| <i>true</i>  | If the current element is a TRUE_EL primitive type.     |
| <i>false</i> | If the current element is not a TRUE_EL primitive type. |

**16.140.2.31 CurrentName()**

```
const char * ParsedJsonDataSet::CurrentName () [inline]
```

Get the name of the current element.

## Returns

A pointer to the name of the current element, if it has one. If it does not, it returns a 0.

**16.140.2.32 CurrentNumber()**

```
double ParsedJsonDataSet::CurrentNumber () [inline]
```

Get the number value of the current element.

## Returns

The number value if the current element's primitive type is NUMBER. Otherwise it returns a quite NAN.

**16.140.2.33 CurrentString()**

```
const char * ParsedJsonDataSet::CurrentString () [inline]
```

Get the string value of the current element.

## Returns

The a pointer to the string value of the current element if it is of primitive type STRING or ALLOC\_STRING. Otherwise it returns a 0.

**16.140.2.34 FindAndCopyFullNameString() [1/2]**

```
int ParsedJsonDataSet::FindAndCopyFullNameString (
 const char * name,
 char * buf,
 int buflen) [inline]
```

Find and copy the string value of the element with the given name in the current object. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.

## Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>name</i>   | The full name of the element to find.     |
| <i>buf</i>    | The buffer to copy the string value to.   |
| <i>buflen</i> | The length of the buffer being copied to. |
|               | •                                         |

## Returns

Full length of the string, regardless of truncation

**16.140.2.35 FindAndCopyFullNameString() [2/2]**

```
int ParsedJsonDataSet::FindAndCopyFullNameString (
 const char * name,
 NBString & nbs) [inline]
```

Find and copy the string value of the element with the given name in the current object. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.

## Parameters

|             |                                                                  |
|-------------|------------------------------------------------------------------|
| <i>name</i> | The full name of the element to find.                            |
| <i>nbs</i>  | The <a href="#">NBString</a> object to copy the string value to. |
|             | •                                                                |

## Returns

Full length of the string, regardless of truncation

**16.140.2.36 FindAndCopyGlobalString() [1/2]**

```
int ParsedJsonDataSet::FindAndCopyGlobalString (
 const char * name,
 char * buf,
 int buflen) [inline]
```

Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names. It then copies the found string to the provided buffer. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.

## Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>name</i>   | The full name of the element to find.     |
| <i>buf</i>    | The buffer to copy the string value to.   |
| <i>buflen</i> | The length of the buffer being copied to. |
|               | •                                         |

## Returns

Full length of the string, regardless of truncation

**16.140.2.37 FindAndCopyGlobalString() [2/2]**

```
int ParsedJsonDataSet::FindAndCopyGlobalString (
 const char * name,
 NBString & nbs) [inline]
```

Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names. It then copies the found string to the provided buffer. If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.

## Parameters

|             |                                                                  |
|-------------|------------------------------------------------------------------|
| <i>name</i> | The full name of the element to find.                            |
| <i>nbs</i>  | The <a href="#">NBString</a> object to copy the string value to. |
|             | •                                                                |

## Returns

Full length of the string, regardless of truncation



**16.140.2.38 FindAndCopyString()** [1/2]

```
int ParsedJsonDataSet::FindAndCopyString (
 const char * name,
 char * buf,
 int buflen) [inline]
```

Find and copy the string value of the element with the given name If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.

**Parameters**

|               |                                           |
|---------------|-------------------------------------------|
| <i>name</i>   | The name of the element to find.          |
| <i>buf</i>    | The buffer to copy the string value to.   |
| <i>buflen</i> | The length of the buffer being copied to. |
|               | •                                         |

**Returns**

Full length of the string, regardless of truncation

**16.140.2.39 FindAndCopyString()** [2/2]

```
int ParsedJsonDataSet::FindAndCopyString (
 const char * name,
 NBString & nbs) [inline]
```

Find and copy the string value of the element with the given name If not provided a buffer or the provided buffer is insufficient, the string will be truncated in the copy buffer.

**Parameters**

|             |                                                                  |
|-------------|------------------------------------------------------------------|
| <i>name</i> | The name of the element to find.                                 |
| <i>nbs</i>  | The <a href="#">NBString</a> object to copy the string value to. |
|             | •                                                                |

**Returns**

Full length of the string, regardless of truncation

**16.140.2.40 FindBoolean()**

```
bool ParsedJsonDataSet::FindBoolean (
 const char * name) [inline]
```

Find the boolean value of the element after the element with the given name.

**Parameters**

|             |                                  |
|-------------|----------------------------------|
| <i>name</i> | The name of the element to find. |
|-------------|----------------------------------|

**Return values**

|              |                                                      |
|--------------|------------------------------------------------------|
| <i>true</i>  | If the element is of the TRUE_EL primitive type.     |
| <i>false</i> | If the element is not of the TRUE_EL primitive type. |

#### 16.140.2.41 FindBooleanInCurentObject()

```
bool ParsedJsonDataSet::FindBooleanInCurentObject (
 const char * name) [inline]
```

Find the boolean value of the element within the current object. Does not search sub-objects.

##### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>name</i> | The name of the element to find. |
|-------------|----------------------------------|

##### Return values

|              |                                                      |
|--------------|------------------------------------------------------|
| <i>true</i>  | If the element is of the TRUE_EL primitive type.     |
| <i>false</i> | If the element is not of the TRUE_EL primitive type. |

#### 16.140.2.42 FindElementAfterName()

```
json_primitive_type ParsedJsonDataSet::FindElementAfterName (
 const char * name) [inline]
```

Finds name in current object points at element after name. This only supports simple, single element names. This searches from the current position to the end.

##### Parameters

|             |                       |
|-------------|-----------------------|
| <i>name</i> | The name to look for. |
|-------------|-----------------------|

##### Returns

The type that the parser is now currently set to.

##### Return values

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>NOTFOUND</i> | If the an element with the name wasn't found. |
|-----------------|-----------------------------------------------|

#### 16.140.2.43 FindElementAfterNameInCurrentArray()

```
json_primitive_type ParsedJsonDataSet::FindElementAfterNameInCurrentArray (
 const char * name) [inline]
```

Looks for elements with the current name in the current ARRAY only. Does not search sub arrays.

##### Parameters

|             |                       |
|-------------|-----------------------|
| <i>name</i> | The name to look for. |
|-------------|-----------------------|

##### Returns

The type that the parser is now currently set to.

##### Return values

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>NOTFOUND</i> | If the an element with the name wasn't found. |
|-----------------|-----------------------------------------------|

#### 16.140.2.44 FindElementAfterNameInCurrentObject()

```
json_primitive_type ParsedJsonDataSet::FindElementAfterNameInCurrentObject (
 const char * name) [inline]
```

Looks for elements with the current name in the current OBJECT only. Does not search sub objects.

## Parameters

|             |                       |
|-------------|-----------------------|
| <i>name</i> | The name to look for. |
|-------------|-----------------------|

## Returns

The type that the parser is now currently set to.

## Return values

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>NOTFOUND</i> | If the an element with the name wasn't found. |
|-----------------|-----------------------------------------------|

**16.140.2.45 FindFullAtName()**

```
json_primitive_type ParsedJsonDataSet::FindFullAtName (
 const char * name) [inline]
```

Find the element in the data set with the given name and move the parser to that element. This searches a full name path, such as ob1.ob2.ob3.ob4.

## Parameters

|             |                            |
|-------------|----------------------------|
| <i>name</i> | The full name to look for. |
|-------------|----------------------------|

## Returns

The type that the parser is now currently set to.

## Return values

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>NOTFOUND</i> | If the an element with the name wasn't found. |
|-----------------|-----------------------------------------------|

**16.140.2.46 FindFullName()**

```
json_primitive_type ParsedJsonDataSet::FindFullName (
 const char * name) [inline]
```

Find the element in the data set with the given name and move the parser to the next element. This searches a full name path, such as ob1.ob2.ob3.ob4.

## Parameters

|             |                            |
|-------------|----------------------------|
| <i>name</i> | The full name to look for. |
|-------------|----------------------------|

## Returns

The type that the parser is now currently set to.

## Return values

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <i>NOTFOUND</i> | If the an element with the name wasn't found, or there is no element after the named element. |
|-----------------|-----------------------------------------------------------------------------------------------|

**16.140.2.47 FindFullNameBoolean()**

```
bool ParsedJsonDataSet::FindFullNameBoolean (
 const char * name) [inline]
```

Find the boolean value of a given element.

## Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>name</i> | The full name of the element to find. |
|-------------|---------------------------------------|

## Return values

|              |                                                      |
|--------------|------------------------------------------------------|
| <i>true</i>  | If the element is of the TRUE_EL primitive type.     |
| <i>false</i> | If the element is not of the TRUE_EL primitive type. |

**16.140.2.48 FindFullNameNumber()**

```
double ParsedJsonDataSet::FindFullNameNumber (
 const char * name) [inline]
```

Find the number value of the element with the given name in the current object.

## Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>name</i> | The full name of the element to find. |
|             | .                                     |

## Returns

The number value of the element if found, otherwise a quiet nan.

**16.140.2.49 FindFullNamePermissiveBoolean()**

```
bool ParsedJsonDataSet::FindFullNamePermissiveBoolean (
 const char * name) [inline]
```

Find the permissive boolean value of a given element.

## Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>name</i> | The full name of the element to find. |
|-------------|---------------------------------------|

## Return values

|              |                                                                                              |
|--------------|----------------------------------------------------------------------------------------------|
| <i>true</i>  | If the current element a TRUE_EL primitive type, is "True", "true", or is a non-zero number. |
| <i>false</i> | If the current element is not a TRUE_EL primitive type, "True", "true", or is 0.             |

**16.140.2.50 FindFullNameString()**

```
const char * ParsedJsonDataSet::FindFullNameString (
 const char * name) [inline]
```

Find the string value of the element with the given name in the current object.

## Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>name</i> | The full name of the element to find. |
|             | .                                     |

## Returns

A pointer to the buffer if the string is found, or null otherwise.

### 16.140.2.51 FindGlobalBoolean()

```
bool ParsedJsonDataSet::FindGlobalBoolean (
 const char * name) [inline]
```

Find the boolean value of a given element. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.

#### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>name</i> | The name of the element to find. |
|-------------|----------------------------------|

#### Return values

|              |                                                      |
|--------------|------------------------------------------------------|
| <i>true</i>  | If the element is of the TRUE_EL primitive type.     |
| <i>false</i> | If the element is not of the TRUE_EL primitive type. |

### 16.140.2.52 FindGlobalElementAfterName()

```
json_primitive_type ParsedJsonDataSet::FindGlobalElementAfterName (
 const char * name) [inline]
```

Finds name in current object points at element after name. This only supports simple, single element names. It starts the search from the parser's current position, and if it doesn't find it, it will start the search over from the beginning.

#### Parameters

|             |                       |
|-------------|-----------------------|
| <i>name</i> | The name to look for. |
|-------------|-----------------------|

#### Returns

The type that the parser is now currently set to.

#### Return values

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>NOTFOUND</i> | If the an element with the name wasn't found. |
|-----------------|-----------------------------------------------|

### 16.140.2.53 FindGlobalNumber()

```
double ParsedJsonDataSet::FindGlobalNumber (
 const char * name) [inline]
```

Find the number value of a given element. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.

#### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>name</i> | The name of the element to find. |
|             | .                                |

#### Returns

The number value of the element if found, otherwise a quiet nan.

#### 16.140.2.54 FindGlobalObject()

```
bool ParsedJsonDataSet::FindGlobalObject (
 const char * name) [inline]
```

Find the OBJECT with the given name. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.

##### Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>name</i> | The name of the OBJECT to find. |
|             | .                               |

##### Return values

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| <i>true</i>  | If the OBJECT was found. The current position will be set to the start of the OBJECT. |
| <i>false</i> | If the OBJECT was not found.                                                          |

#### 16.140.2.55 FindGlobalPermissiveBoolean()

```
bool ParsedJsonDataSet::FindGlobalPermissiveBoolean (
 const char * name) [inline]
```

Find the permissive boolean value of a given element. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.

##### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>name</i> | The name of the element to find. |
|-------------|----------------------------------|

##### Return values

|              |                                                                                              |
|--------------|----------------------------------------------------------------------------------------------|
| <i>true</i>  | If the current element a TRUE_EL primitive type, is "True", "true", or is a non-zero number. |
| <i>false</i> | If the current element is not a TRUE_EL primitive type, "True", "true", or is 0.             |

#### 16.140.2.56 FindGlobalString()

```
const char * ParsedJsonDataSet::FindGlobalString (
 const char * name) [inline]
```

Find the string value of a given element. Starts at the current position and then starts again at the beginning of the data set if the element isn't found. This supports only simple, single element names.

##### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>name</i> | The name of the element to find. |
|-------------|----------------------------------|

##### Returns

A pointer to the buffer if the string is found, or null otherwise.

#### 16.140.2.57 FindNumber()

```
double ParsedJsonDataSet::FindNumber (
 const char * name) [inline]
```

Find the number value of the element after the element with the given name.

## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>name</i> | The name of the element to find. |
|-------------|----------------------------------|

## Returns

The number value of the element if found, otherwise a quiet nan.

**16.140.2.58 FindNumberInCurentObject()**

```
double ParsedJsonDataSet::FindNumberInCurentObject (
 const char * name) [inline]
```

Find the number value of the element with the given name in the current object. Does not search sub-objects.

## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>name</i> | The name of the element to find. |
|             | .                                |

## Returns

The number value of the element if found, otherwise a quiet nan.

**16.140.2.59 FindObject()**

```
bool ParsedJsonDataSet::FindObject (
 const char * name) [inline]
```

Find the OBJECT of the element after the element with the given name.

## Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>name</i> | The name of the OBJECT to find. |
|-------------|---------------------------------|

## Return values

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| <i>true</i>  | If the OBJECT was found. The current position will be set to the start of the OBJECT. |
| <i>false</i> | If the OBJECT was not found.                                                          |

**16.140.2.60 FindObjectInCurentObject()**

```
bool ParsedJsonDataSet::FindObjectInCurentObject (
 const char * name) [inline]
```

Find the OBJECT of the element with the given name in the current OBJECT. Does not search sub-objects.

## Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>name</i> | The name of the OBJECT to find. |
|-------------|---------------------------------|

## Return values

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| <i>true</i>  | If the OBJECT was found. The current position will be set to the start of the OBJECT. |
| <i>false</i> | If the OBJECT was not found.                                                          |

#### 16.140.2.61 FindPermissiveBoolean()

```
bool ParsedJsonDataSet::FindPermissiveBoolean (
 const char * name) [inline]
```

Find the permissive boolean value of the element after the element with the provided name.

##### Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>name</i> | The name of element to search on. |
|-------------|-----------------------------------|

##### Return values

|              |                                                                                              |
|--------------|----------------------------------------------------------------------------------------------|
| <i>true</i>  | If the current element a TRUE_EL primitive type, is "True", "true", or is a non-zero number. |
| <i>false</i> | If the current element is not a TRUE_EL primitive type, "True", "true", or is 0.             |

#### 16.140.2.62 FindPermissiveBooleanInCurentObject()

```
bool ParsedJsonDataSet::FindPermissiveBooleanInCurentObject (
 const char * name) [inline]
```

Find the permissive boolean value of the element with the given name in the current object. Does not search sub-objects.

##### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>name</i> | The name of the element to find. |
|             | .                                |

##### Return values

|              |                                                                                              |
|--------------|----------------------------------------------------------------------------------------------|
| <i>true</i>  | If the current element a TRUE_EL primitive type, is "True", "true", or is a non-zero number. |
| <i>false</i> | If the current element is not a TRUE_EL primitive type, "True", "true", or is 0.             |

#### 16.140.2.63 FindString()

```
const char * ParsedJsonDataSet::FindString (
 const char * name) [inline]
```

Find the string value of the element after the element with the given name.

##### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>name</i> | The name of the element to find. |
|-------------|----------------------------------|

##### Returns

A pointer to the buffer if the string is found, or null otherwise.

#### 16.140.2.64 FindStringInCurentObject()

```
const char * ParsedJsonDataSet::FindStringInCurentObject (
 const char * name) [inline]
```

Find the string value of the element with the given name in the current object. Does not search sub-objects.



## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>name</i> | The name of the element to find. |
|             | .                                |

## Returns

A pointer to the buffer if the string is found, or null otherwise.

**16.140.2.65 GetCurrent()**

```
json_primitive_type ParsedJsonDataSet::GetCurrent () [inline]
```

Get the element at the current position of the JSON data set.

## Returns

The current element of the JSON data set.

## Return values

|                  |                                                                        |
|------------------|------------------------------------------------------------------------|
| <i>UNDEFINED</i> | If there is an error in the parsing or if the parsing is not complete. |
| <i>EOF_EL</i>    | If it's the last element in the data set.                              |

## See also

[GetFirst\(\)](#)  
[GetNext\(\)](#)  
[GetRawCurrent\(\)](#)  
[SkipCurrentValue\(\)](#)  
[ResetPosition\(\)](#)

**16.140.2.66 GetFirst()**

```
json_primitive_type ParsedJsonDataSet::GetFirst () [inline]
```

Get the first element of the JSON data set.

## Returns

The first element of the JSON data set.

## Return values

|                  |                                                                        |
|------------------|------------------------------------------------------------------------|
| <i>UNDEFINED</i> | If there is an error in the parsing or if the parsing is not complete. |
| <i>EOF_EL</i>    | If it's the last element in the data set.                              |

## See also

[GetNext\(\)](#)  
[GetCurrent\(\)](#)  
[GetRawCurrent\(\)](#)  
[ResetPosition\(\)](#)  
[SkipCurrentValue\(\)](#)

### 16.140.2.67 GetNext()

`json_primitive_type` `ParsedJsonDataSet::GetNext ()` `[inline]`

Get the element at the next position of the JSON data set.

#### Returns

The next element of the JSON data set.

#### Return values

|                  |                                                                        |
|------------------|------------------------------------------------------------------------|
| <i>UNDEFINED</i> | If there is an error in the parsing or if the parsing is not complete. |
| <i>EOF_EL</i>    | If it's the last element in the data set.                              |

#### See also

[GetFirst\(\)](#)

[GetCurrent\(\)](#)

[GetRawCurrent\(\)](#)

[ResetPosition\(\)](#)

[SkipCurrentValue\(\)](#)

### 16.140.2.68 GetNextArray()

`json_primitive_type` `ParsedJsonDataSet::GetNextArray ()`

Iterates the current element of the JSON data set until it finds one of type `BEGIN_ARRAY`.

#### Returns

The next element with type `BEGIN_ARRAY` after the current position.

#### Return values

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| <i>NOTFOUND</i> | If there is no <code>BEGIN_ARRAY</code> element after the current position. |
|-----------------|-----------------------------------------------------------------------------|

#### See also

[GetNextNameInCurrentObject\(\)](#)

[GetNextNameInCurrentArray\(\)](#)

### 16.140.2.69 GetNextBoolInCurrentArray()

`json_primitive_type` `ParsedJsonDataSet::GetNextBoolInCurrentArray ()`

Get the next element of type `TRUE_EL` that exists in the current `ARRAY` after the current position.

#### Returns

The next element with type `TRUE_EL` in the current `ARRAY` after the current position.

#### Return values

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>NOTFOUND</i> | If there is no <code>NAME</code> element after the current position. |
|-----------------|----------------------------------------------------------------------|

#### See also

[CurrentBool\(\)](#)

[PermissiveCurrentBool\(\)](#)

### 16.140.2.70 GetNextName()

`json_primitive_type` ParsedJsonDataSet::GetNextName () [inline]

Iterates the current element of the JSON data set until it finds one of type NAME.

#### Returns

The next element with type NAME after the current position.

#### Return values

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>NOTFOUND</i> | If there is no NAME element after the current position. |
|-----------------|---------------------------------------------------------|

#### See also

[GetNextNameInCurrentObject\(\)](#)

[GetNextNameInCurrentArray\(\)](#)

### 16.140.2.71 GetNextNameInCurrentArray()

`json_primitive_type` ParsedJsonDataSet::GetNextNameInCurrentArray () [inline]

Get the next element of type NAME that exists in the current ARRAY after the current position.

#### Returns

The next element with type NAME in the current ARRAY after the current position.

#### Return values

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>NOTFOUND</i> | If there is no NAME element after the current position. |
|-----------------|---------------------------------------------------------|

#### See also

[GetNextName\(\)](#)

[GetNextNameInCurrentObject\(\)](#)

### 16.140.2.72 GetNextNameInCurrentObject()

`json_primitive_type` ParsedJsonDataSet::GetNextNameInCurrentObject () [inline]

Get the next element of type NAME that exists in the current OBJECT after the current position.

#### Returns

The next element with type NAME in the current OBJECT after the current position.

#### Return values

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>NOTFOUND</i> | If there is no NAME element after the current position. |
|-----------------|---------------------------------------------------------|

#### See also

[GetNextName\(\)](#)

[GetNextNameInCurrentArray\(\)](#)

### 16.140.2.73 GetNextNumberInCurrentArray()

`json_primitive_type` ParsedJsonDataSet::GetNextNumberInCurrentArray ()

Get the next element of type NUMBER that exists in the current ARRAY after the current position.

#### Returns

The next element with type NUMBER in the current ARRAY after the current position.

## Return values

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <b>NOTFOUND</b> | If there is no NAME element after the current position. |
|-----------------|---------------------------------------------------------|

## See also

[CurrentNumber\(\)](#)**16.140.2.74 GetNextObject()**`json_primitive_type` `ParsedJsonDataSet::GetNextObject ()`

Get the next element of type name that exists in the current OBJECT after the current position.

## Returns

The next element with type BEGIN\_OBJECT after the current position.

## Return values

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| <b>NOTFOUND</b> | If there is no BEGIN_OBJECT element after the current position. |
|-----------------|-----------------------------------------------------------------|

## See also

[GetNextNameInCurrentObject\(\)](#)[GetNextNameInCurrentArray\(\)](#)**16.140.2.75 GetNextObjectInCurrentArray()**`json_primitive_type` `ParsedJsonDataSet::GetNextObjectInCurrentArray ()`

Get the next element of type BEGIN\_OBJECT that exists in the current ARRAY after the current position.

## Returns

The next element with type BEGIN\_OBJECT in the current ARRAY after the current position.

## Return values

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| <b>NOTFOUND</b> | If there is no BEGIN_OBJECT element after the current position. |
|-----------------|-----------------------------------------------------------------|

## See also

[GetNextName\(\)](#)[GetNextNameInCurrentObject\(\)](#)**16.140.2.76 GetNextStringInCurrentArray()**`json_primitive_type` `ParsedJsonDataSet::GetNextStringInCurrentArray ()`

Get the next element of type STRING that exists in the current ARRAY after the current position.

## Returns

The next element with type STRING in the current ARRAY after the current position.

## Return values

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>NOTFOUND</i> | If there is no NAME element after the current position. |
|-----------------|---------------------------------------------------------|

## See also

[CurrentString\(\)](#)

**16.140.2.77 GetParsePosition()**

[JsonRef](#) ParsedJsonDataSet::GetParsePosition ()

Gets the current parse position object for the JSON data set.

## Returns

The current parse position object.

**16.140.2.78 GetPrintSize()**

```
int ParsedJsonDataSet::GetPrintSize (
 bool pretty = false) [inline]
```

Calculates how many characters the JSON data set would take to print.

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>pretty</i> | Whether indentation and new lines should be provided between elements. |
|---------------|------------------------------------------------------------------------|

## Returns

The number of characters needed to print the JSON data set.

**16.140.2.79 GetRawCurrent()**

[json\\_primitive\\_type](#) ParsedJsonDataSet::GetRawCurrent () [inline]

Get the element at the current position of the JSON data set, including non-public types.

## Returns

The current element of the JSON data set.

## Return values

|                  |                                                                        |
|------------------|------------------------------------------------------------------------|
| <i>UNDEFINED</i> | If there is an error in the parsing or if the parsing is not complete. |
| <i>EOF_EL</i>    | If it's the last element in the data set.                              |

## See also

[GetFirst\(\)](#)

[GetNext\(\)](#)

[GetCurrent\(\)](#)

[ResetPosition\(\)](#)

[SkipCurrentValue\(\)](#)

### 16.140.2.80 name()

```
JsonRef ParsedJsonDataSet::name (
 const char * name) [inline]
```

Get a [JsonRef](#) representing any entity from a parent object by key name.  
Useful if more than one JSON object/element has the same name.

Example

```
JsonRef userThree = myJsonRef.name("users")[3];
```

#### Parameters

|              |                                       |
|--------------|---------------------------------------|
| <i>*name</i> | Pointer to name of the entity to find |
|--------------|---------------------------------------|

#### Returns

The reference to the entity of type [JsonRef](#)

#### See also

[ParsedJsonDataSet::operator\(\)\(\)](#), [ParsedJsonDataSet::object\(\)](#)

### 16.140.2.81 object()

```
JsonRef ParsedJsonDataSet::object (
 const char * name) [inline]
```

Get a [JsonRef](#) representing an object from a parent object by key name.

Example

```
JsonRef john = myJsonRef.object("users").object("john");
```

#### Parameters

|              |                                       |
|--------------|---------------------------------------|
| <i>*name</i> | Pointer to name of the object to find |
|--------------|---------------------------------------|

#### Returns

The reference to the object of type [JsonRef](#)

### 16.140.2.82 operator()()

```
JsonRef ParsedJsonDataSet::operator() (
 const char * name) [inline]
```

Get a [JsonRef](#) representing any entity from a parent object by key name.  
Shortcut for [JsonRef::name\(\)](#)

Example

```
JsonRef allUsers = myJsonRef("users");
```

#### Parameters

|              |                                       |
|--------------|---------------------------------------|
| <i>*name</i> | Pointer to name of the entity to find |
|--------------|---------------------------------------|

#### Returns

The reference to the entity of type [JsonRef](#)

#### See also

[ParsedJsonDataSet::name\(\)](#), [ParsedJsonDataSet::object\(\)](#)

**16.140.2.83 operator[]()**

```
JsonRef ParsedJsonDataSet::operator[] (
 int i) [inline]
```

Get a [JsonRef](#) from an array by numerical index.

**Example**

```
JsonRef userThree = myJsonRef("users")[3];
```

**Returns**

A [JsonRef](#) if array element is found

**16.140.2.84 PermissiveCurrentBool()**

```
bool ParsedJsonDataSet::PermissiveCurrentBool () [inline]
```

Returns true if the current element a TRUE\_EL primitive type, is "True", "true", or is a non-zero number.

**Return values**

|              |                                                                                              |
|--------------|----------------------------------------------------------------------------------------------|
| <i>true</i>  | If the current element a TRUE_EL primitive type, is "True", "true", or is a non-zero number. |
| <i>false</i> | If the current element does not meet the true condition..                                    |

**16.140.2.85 PrintObject()**

```
int ParsedJsonDataSet::PrintObject (
 bool pretty = false) [inline]
```

Prints the JSON data set to stdout.

**Parameters**

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>pretty</i> | Whether to show indentation and new lines between JSON elements. |
|---------------|------------------------------------------------------------------|

**16.140.2.86 PrintObjectToBuffer()**

```
int ParsedJsonDataSet::PrintObjectToBuffer (
 char * buffer,
 int maxlen,
 bool pretty = false) [inline]
```

Prints the JSON data set to a provided buffer.

**Parameters**

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>buffer</i> | The buffer to print to.                                                |
| <i>maxlen</i> | The max length of the buffer.                                          |
| <i>pretty</i> | Whether indentation and new lines should be provided between elements. |

**Returns**

The number of characters printed.

**16.140.2.87 PrintObjectToFd()**

```
int ParsedJsonDataSet::PrintObjectToFd (
 int fd,
 bool pretty = false) [inline]
```

Prints the JSON data set to a specified file descriptor.

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>fd</i>     | The file descriptor to print to.                                       |
| <i>pretty</i> | Whether indentation and new lines should be provided between elements. |

**16.140.2.88 PrintObjectToString()**

```
int ParsedJsonDataSet::PrintObjectToString (
 NBString & s,
 bool pretty = false) [inline]
```

Prints the JSON data set to a [NBString](#) object.

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>s</i>      | The <a href="#">NBString</a> to print to.                              |
| <i>pretty</i> | Whether indentation and new lines should be provided between elements. |

**16.140.2.89 ReadFrom()**

```
virtual int ParsedJsonDataSet::ReadFrom (
 int fd) [virtual]
```

Reads in data from the specified file descriptor and parses it into the JSON data set.

## Parameters

|           |                                        |
|-----------|----------------------------------------|
| <i>fd</i> | The file descriptor to read data from. |
|-----------|----------------------------------------|

## Returns

The number of bytes read

Implements [buffer\\_object](#).

**16.140.2.90 ResetPosition()**

```
void ParsedJsonDataSet::ResetPosition () [inline]
```

Resets the parser position to the beginning of the JSON data set.

## See also

[GetFirst\(\)](#)  
[GetNext\(\)](#)  
[GetCurrent\(\)](#)  
[SkipCurrentValue\(\)](#)

**16.140.2.91 SetParsePosition()**

```
JsonRef ParsedJsonDataSet::SetParsePosition (
 JsonRef pos)
```

Sets the current parse position object for the JSON data set.

## Parameters

|            |                                                                           |
|------------|---------------------------------------------------------------------------|
| <i>pos</i> | The <a href="#">JsonRef</a> that you want to set the current position to. |
|------------|---------------------------------------------------------------------------|

## Returns

The current parse ref position for the JSON data set.



**16.140.2.92 SkipCurrentValue()**

```
json_primitive_type ParsedJsonDataSet::SkipCurrentValue () [inline]
```

Skips over the current value, and get the next element. If called inside an ARRAY or OBJECT, it will walk to the end of the ARRAY or OBJECT and return the next element it finds. If it reaches the end of the ARRAY or OBJECT before it finds a value, it will return NOTFOUND.

**Returns**

The next element.

**Return values**

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>NOTFOUND</i> | If there are no other elements after the current one. |
|-----------------|-------------------------------------------------------|

**See also**

[GetFirst\(\)](#)  
[GetCurrent\(\)](#)  
[GetNext\(\)](#)  
[ResetPosition\(\)](#)  
[SkipCurrentValue\(\)](#)

**16.140.2.93 WriteData()**

```
virtual int ParsedJsonDataSet::WriteData (
 const unsigned char * pCopyFrom,
 int numBytes) [virtual]
```

Writes the passed in data to the JSON data set.

**Parameters**

|                  |                         |
|------------------|-------------------------|
| <i>pCopyFrom</i> | What data to write.     |
| <i>numBytes</i>  | How many bytes to copy. |

**Returns**

The number of bytes written

Implements [buffer\\_object](#).

The documentation for this class was generated from the following file:

- `json_lexer.h`

**16.141 ParsedURI Class Reference**

Parsed Uniform Resource Identifier Class (URI)

```
#include <http_funcs.h>
```

**Public Member Functions**

- void [NewUri](#) (const char \*uri, uint16\_t timeout=20 \*[TICKS\\_PER\\_SECOND](#), bool skipLookup=false)  
*Replace the existing (URI) object with new information.*
- **ParsedURI** ()  
*Constructor to create a new blank URI object. For rare cases.*
- [ParsedURI](#) (const char \*uri, uint16\_t timeout=20 \*[TICKS\\_PER\\_SECOND](#), bool skipLookup=false)

- Constructor to create a new URI object.*
  - bool [valid](#) () const  
*Check to see if URI is valid.*
  - const char \* [GetPath](#) ()  
*Get the cached URI path.*
  - const char \* [GetHost](#) ()  
*Get the cached URI Host name.*
  - [IPADDR GetAddr](#) ()  
*Get the resolved host Address.*
  - uint16\_t [GetPort](#) ()  
*Get the cached URI network port number.*
  - bool [IsSecure](#) ()  
*Check the security state of the cached URI object. Requires WEB\_CLIENT\_SSL\_SUPPORT be defined in [predef.h](#).*
  - void [Invalidate](#) ()  
*Set the state of the cached URI object to invalid.*

### 16.141.1 Detailed Description

Parsed Uniform Resource Identifier Class (URI)

Web Client HTTP functions can pass a URL, which has to be parsed and use DNS to resolve if necessary. This class provides an object with the parsed and DNS results that can be used for future transactions. The intent of using this class is to cache the parsed results and dns lookup for communication to the same resource.

Enabling WEB\_CLIENT\_SSL\_SUPPORT in [predef.h](#) adds TLS support.

The maximum size of the cached URI, path and hose are 256 bytes each.

### 16.141.2 Constructor & Destructor Documentation

#### 16.141.2.1 ParsedURI()

```
ParsedURI::ParsedURI (
 const char * uri,
 uint16_t timeout = 20 * TICKS_PER_SECOND,
 bool skipLookup = false) [inline]
```

Constructor to create a new URI object.

Parameters

|                   |                                    |
|-------------------|------------------------------------|
| <i>uri</i>        | Pointer to a URI.                  |
| <i>timeout</i>    | Number of sysem time tick to wait. |
| <i>skipLookup</i> | Skip URI lookup. Default is false. |

### 16.141.3 Member Function Documentation

#### 16.141.3.1 GetAddr()

```
IPADDR ParsedURI::GetAddr () [inline]
```

Get the resolved host Address.

Returns

The resolved host address.

**16.141.3.2 GetHost()**

```
const char * ParsedURI::GetHost () [inline]
```

Get the cached URI Host name.

**Returns**

Pointer to the cached host name if the cached URI is valid, otherwise NULL.

**16.141.3.3 GetPath()**

```
const char * ParsedURI::GetPath () [inline]
```

Get the cached URI path.

**Returns**

Pointer to the cached path string if the cached URI is valid, otherwise NULL.

**16.141.3.4 GetPort()**

```
uint16_t ParsedURI::GetPort () [inline]
```

Get the cached URI network port number.

**Returns**

The cached URI port number if the cached URI is valid, otherwise 0.

**16.141.3.5 IsSecure()**

```
bool ParsedURI::IsSecure () [inline]
```

Check the security state of the cached URI object. Requires WEB\_CLIENT\_SSL\_SUPPORT be defined in [predef.h](#).

**Return values**

|              |                        |
|--------------|------------------------|
| <i>true</i>  | Secure.                |
| <i>false</i> | URI object is invalid. |

**16.141.3.6 NewUri()**

```
void ParsedURI::NewUri (
 const char * uri,
 uint16_t timeout = 20 *TICKS_PER_SECOND,
 bool skipLookup = false)
```

Replace the existing (URI) object with new information.

Update the existing URI with a new value.

**Parameters**

|                   |                                     |
|-------------------|-------------------------------------|
| <i>uri</i>        | Pointer to a URI string.            |
| <i>timeout</i>    | Number of sysem time ticks to wait. |
| <i>skipLookup</i> | Skip URI lookup. Default is false.  |

**16.141.3.7 valid()**

```
bool ParsedURI::valid () const [inline]
```

Check to see if URI is valid.

## Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | if valid     |
| <i>false</i> | if not valid |

The documentation for this class was generated from the following file:

- `http_funcs.h`

## 16.142 PartitionEntry Struct Reference

```
#include <effs_pt.h>
```

### Public Member Functions

- `uint64_t start ()`
- `uint64_t end ()`
- `uint64_t startAddr (uint32_t sectorSize)`
- `uint64_t endAddr (uint32_t sectorSize)`
- `uint64_t sectors ()`
- `uint64_t setStart (uint64_t startSector)`
- `uint64_t setSectors (uint64_t sCount)`

### Public Attributes

- `uint32_t flags: 8`  
*Any special flags for the partition.*
- `uint32_t sectorCount: 24`  
*How many sectors are in this partition.*
- `uint32_t partitionType: 8`  
*What type of filesystem is this partition.*
- `uint32_t firstSector: 24`  
*Index of first sector for this partition.*

### 16.142.1 Detailed Description

Note: `sectorCount` and `firstSector` have special meaning if the MSB is set. If MSB is set, the format is as follows:

```
realSectorCount = (2**(sectorCount[22:18]+4))*(sectorCount[17:0])
realFirstSector = (2**(firstSector[22:18]+4))*(firstSector[17:0])
```

The documentation for this struct was generated from the following file:

- `effs_pt.h`

## 16.143 PartitionTable Struct Reference

```
#include <effs_pt.h>
```

**Public Member Functions**

- uint64\_t **sectors** ()
- uint64\_t **setSectors** (uint64\_t sCount)
- uint64\_t **sectorLen** ()

**Public Attributes**

- union {
 struct {
 uint8\_t **bootExe** [422]
 *If direct execution is ever supported...*
 uint8\_t **tableLenExp**
*Table Length => 2\*\*tableLenExp: 2\*\*8 => 256 bytes.*
 uint8\_t **sectorLenExp**
*Sector Length, see Table Length.*
 uint32\_t **totalSectors**
*Total sectors available, see.*
 PartitionEntry **part** [8]
 *Partition entries.*
 uint8\_t **csums** [16]
 *Hamming code csum of 31 byte blocks comprising the Table.*
 }

 uint8\_t **pAsBytePtr** [1]
 };

**16.143.1 Detailed Description**

Note: totalSectors has special meaning if the MSB is set. If MSB is set, the format is as follows:

```
realSectorCount = (2** (sectorCount[30:27]+4)) * (sectorCount[26:0])
```

The documentation for this struct was generated from the following file:

- effs\_pt.h

**16.144 fifo\_buffer\_storage::PeekIterator Class Reference**

Iterator for non-destructive examination of buffered data.

```
#include <buffers.h>
```

**16.144.1 Detailed Description**

Iterator for non-destructive examination of buffered data.

Allows reading data from the FIFO without removing it, useful for protocol parsing and lookahead operations.

The documentation for this class was generated from the following file:

- buffers.h

**16.145 PinIO Class Reference**

GPIO Pin Class.

```
#include <cpu_pins.h>
```

## Public Types

- enum `pin_fn_t` {  
`PIN_FN_0` = 0, `PIN_FN_1` = 1, `PIN_FN_2` = 2, `PIN_FN_3` = 3,  
`PIN_FN_IN` = 4, `PIN_FN_OUT` = 8, `PIN_FN_IN` = 4, `PIN_FN_OUT` = 8,  
`PIN_FN_A`, `PIN_FN_B`, `PIN_FN_C`, `PIN_FN_D`,  
`PIN_FN_IN` = 4, `PIN_FN_OUT` = 8, `PIN_FN_0` = 0, `PIN_FN_1` = 1,  
`PIN_FN_2` = 2, `PIN_FN_3` = 3, `PIN_FN_4` = 6, `PIN_FN_5` = 7,  
`PIN_FN_6` = 8, `PIN_FN_7` = 9, `PIN_FN_8` = 10, `PIN_FN_9` = 11,  
`PIN_FN_10` = 12, `PIN_FN_11` = 13, `PIN_FN_12` = 14, `PIN_FN_13` = 15,  
`PIN_FN_14` = 16, `PIN_FN_15` = 17, `PIN_FN_16` = 18, `PIN_FN_17` = 19,  
`PIN_FN_18` = 20, `PIN_FN_19` = 21, `PIN_FN_20` = 22, `PIN_FN_21` = 23,  
`PIN_FN_22` = 24, `PIN_FN_23` = 25, `PIN_FN_24` = 26, `PIN_FN_25` = 27,  
`PIN_FN_26` = 28, `PIN_FN_27` = 29, `PIN_FN_28` = 30, `PIN_FN_29` = 31,  
`PIN_FN_30` = 32, `PIN_FN_31` = 33, `PIN_FN_32` = 34 }
- enum `pull_t`
- enum `pin_fn_t` {  
`PIN_FN_0` = 0, `PIN_FN_1` = 1, `PIN_FN_2` = 2, `PIN_FN_3` = 3,  
`PIN_FN_IN` = 4, `PIN_FN_OUT` = 8, `PIN_FN_IN` = 4, `PIN_FN_OUT` = 8,  
`PIN_FN_A`, `PIN_FN_B`, `PIN_FN_C`, `PIN_FN_D`,  
`PIN_FN_IN` = 4, `PIN_FN_OUT` = 8, `PIN_FN_0` = 0, `PIN_FN_1` = 1,  
`PIN_FN_2` = 2, `PIN_FN_3` = 3, `PIN_FN_4` = 6, `PIN_FN_5` = 7,  
`PIN_FN_6` = 8, `PIN_FN_7` = 9, `PIN_FN_8` = 10, `PIN_FN_9` = 11,  
`PIN_FN_10` = 12, `PIN_FN_11` = 13, `PIN_FN_12` = 14, `PIN_FN_13` = 15,  
`PIN_FN_14` = 16, `PIN_FN_15` = 17, `PIN_FN_16` = 18, `PIN_FN_17` = 19,  
`PIN_FN_18` = 20, `PIN_FN_19` = 21, `PIN_FN_20` = 22, `PIN_FN_21` = 23,  
`PIN_FN_22` = 24, `PIN_FN_23` = 25, `PIN_FN_24` = 26, `PIN_FN_25` = 27,  
`PIN_FN_26` = 28, `PIN_FN_27` = 29, `PIN_FN_28` = 30, `PIN_FN_29` = 31,  
`PIN_FN_30` = 32, `PIN_FN_31` = 33, `PIN_FN_32` = 34 }

*Pin Function modes to configure the managed pin(s).*

- enum `pull_t`  
*Pin pullup and pulldown modes to configure the managed pin(s).*
- enum `pin_fn_t` {  
`PIN_FN_0` = 0, `PIN_FN_1` = 1, `PIN_FN_2` = 2, `PIN_FN_3` = 3,  
`PIN_FN_IN` = 4, `PIN_FN_OUT` = 8, `PIN_FN_IN` = 4, `PIN_FN_OUT` = 8,  
`PIN_FN_A`, `PIN_FN_B`, `PIN_FN_C`, `PIN_FN_D`,  
`PIN_FN_IN` = 4, `PIN_FN_OUT` = 8, `PIN_FN_0` = 0, `PIN_FN_1` = 1,  
`PIN_FN_2` = 2, `PIN_FN_3` = 3, `PIN_FN_4` = 6, `PIN_FN_5` = 7,  
`PIN_FN_6` = 8, `PIN_FN_7` = 9, `PIN_FN_8` = 10, `PIN_FN_9` = 11,  
`PIN_FN_10` = 12, `PIN_FN_11` = 13, `PIN_FN_12` = 14, `PIN_FN_13` = 15,  
`PIN_FN_14` = 16, `PIN_FN_15` = 17, `PIN_FN_16` = 18, `PIN_FN_17` = 19,  
`PIN_FN_18` = 20, `PIN_FN_19` = 21, `PIN_FN_20` = 22, `PIN_FN_21` = 23,  
`PIN_FN_22` = 24, `PIN_FN_23` = 25, `PIN_FN_24` = 26, `PIN_FN_25` = 27,  
`PIN_FN_26` = 28, `PIN_FN_27` = 29, `PIN_FN_28` = 30, `PIN_FN_29` = 31,  
`PIN_FN_30` = 32, `PIN_FN_31` = 33, `PIN_FN_32` = 34 }

*Pin Function modes to configure the managed pin(s).*

- enum `pull_t`  
*Pin pullup and pulldown modes to configure the managed pin(s).*

## Public Member Functions

- `PinIO ()`  
Construct an empty `PinIO`. Exists for bootstrap compatibility; not intended for general use.
- `constexpr PinIO (uint32_t port, uint32_t pin)`  
Construct a `PinIO` for a specific `cpu pin`.
- `constexpr PinIO (const PinIO &rhs)`  
Construct a copy of another `PinIO`.

- void **setFn** (pin\_fn\_t fn) const  
*Set the pin function for the managed pin(s).*
- int8\_t **getFn** ()  
*Get the pin function for the managed pin(s).*
- void **function** (pin\_fn\_t fn) const  
*Set the pin function for the managed pin(s).*
- void **hiz** () const  
*Configure the pin(s) to Input.*
- void **drive** () const  
*Configure the pin(s) to Output, without modifying the driven value.*
- void **set** () const  
*Drive the pin(s) High.*
- void **clr** () const  
*Drive the pin(s) Low.*
- bool **tgl** () const  
*Toggle the driven value for the pin(s).*
- bool **toggle** () const  
*Toggle the driven value for the pin(s).*
- bool **readBack** () const  
*Read the state of the pin(s) line state without changing the pin function or direction.*
- bool **read** () const  
*Configure the pin as an input then return the line state.*
- bool **operator=** (bool val)  
*Assign a driven value to the pin(s).*
- **PinIO** & **operator=** (const **PinIO** &rhs)  
*Assign a driven value to the pin(s) based on the line state of another pin(s).*
- **operator bool** () const  
*Read the line state of the pin(s).*
- bool **operator!** () const  
*Return the opposite of the driven value of the pin(s).*
- void **multidrv** (bool enable) const  
*Configure the multidrive/open-drain driver for the pin(s).*
- void **setHighStrength** (bool bHighDrive)  
*Configure the drive strength of the output driver for the pin(s).*
- void **setPull** (pull\_t pull)  
*Configure the pad Pull resistor for the pin(s).*
- void **PullUp** (bool enable) const  
*Configure the pad Pull Up resistor for the pin(s).*
- void **PullDown** (bool enable) const  
*Configure the pad Pull Down resistor for the pin(s).*
- uint16\_t **analogRead** () const  
*Read an analog voltage on the given Pin. Only available for pins connected to the ADC.*

### Public Attributes

- volatile Pio & **pio**
- uint32\_t **mask**

### Friends

- class **\_PinVector**

## 16.145.1 Detailed Description

GPIO Pin Class.

## 16.145.2 Member Enumeration Documentation

### 16.145.2.1 pin\_fn\_t [1/3]

enum `PinIO::pin_fn_t`

**pin\_fn\_t**

Enumerator

|            |                |
|------------|----------------|
| PIN_FN_0   | Peripheral 0.  |
| PIN_FN_1   | Peripheral 1.  |
| PIN_FN_2   | Peripheral 2.  |
| PIN_FN_3   | Peripheral 3.  |
| PIN_FN_IN  | Input.         |
| PIN_FN_OUT | Output.        |
| PIN_FN_IN  | Input.         |
| PIN_FN_OUT | Output.        |
| PIN_FN_A   | Peripheral A.  |
| PIN_FN_B   | Peripheral B.  |
| PIN_FN_C   | Peripheral C.  |
| PIN_FN_D   | Peripheral D.  |
| PIN_FN_IN  | Input.         |
| PIN_FN_OUT | Output.        |
| PIN_FN_0   | Peripheral 0.  |
| PIN_FN_1   | Peripheral 1.  |
| PIN_FN_2   | Peripheral 2.  |
| PIN_FN_3   | Peripheral 3.  |
| PIN_FN_4   | Peripheral 4.  |
| PIN_FN_5   | Peripheral 5.  |
| PIN_FN_6   | Peripheral 6.  |
| PIN_FN_7   | Peripheral 7.  |
| PIN_FN_8   | Peripheral 8.  |
| PIN_FN_9   | Peripheral 9.  |
| PIN_FN_10  | Peripheral 10. |
| PIN_FN_11  | Peripheral 11. |
| PIN_FN_12  | Peripheral 12. |
| PIN_FN_13  | Peripheral 13. |
| PIN_FN_14  | Peripheral 14. |
| PIN_FN_15  | Peripheral 15. |
| PIN_FN_16  | Peripheral 16. |
| PIN_FN_17  | Peripheral 17. |
| PIN_FN_18  | Peripheral 18. |
| PIN_FN_19  | Peripheral 19. |
| PIN_FN_20  | Peripheral 20. |
| PIN_FN_21  | Peripheral 21. |
| PIN_FN_22  | Peripheral 22. |
| PIN_FN_23  | Peripheral 23. |
| PIN_FN_24  | Peripheral 24. |



## Enumerator

|           |                |
|-----------|----------------|
| PIN_FN_25 | Peripheral 25. |
| PIN_FN_26 | Peripheral 26. |
| PIN_FN_27 | Peripheral 27. |
| PIN_FN_28 | Peripheral 28. |
| PIN_FN_29 | Peripheral 29. |
| PIN_FN_30 | Peripheral 30. |
| PIN_FN_31 | Peripheral 31. |
| PIN_FN_32 | Peripheral 32. |

## 16.145.2.2 pin\_fn\_t [2/3]

enum `PinIO::pin_fn_t`

Pin Function modes to configure the managed pin(s).

## Enumerator

|            |                |
|------------|----------------|
| PIN_FN_0   | Peripheral 0.  |
| PIN_FN_1   | Peripheral 1.  |
| PIN_FN_2   | Peripheral 2.  |
| PIN_FN_3   | Peripheral 3.  |
| PIN_FN_IN  | Input.         |
| PIN_FN_OUT | Output.        |
| PIN_FN_IN  | Input.         |
| PIN_FN_OUT | Output.        |
| PIN_FN_A   | Peripheral A.  |
| PIN_FN_B   | Peripheral B.  |
| PIN_FN_C   | Peripheral C.  |
| PIN_FN_D   | Peripheral D.  |
| PIN_FN_IN  | Input.         |
| PIN_FN_OUT | Output.        |
| PIN_FN_0   | Peripheral 0.  |
| PIN_FN_1   | Peripheral 1.  |
| PIN_FN_2   | Peripheral 2.  |
| PIN_FN_3   | Peripheral 3.  |
| PIN_FN_4   | Peripheral 4.  |
| PIN_FN_5   | Peripheral 5.  |
| PIN_FN_6   | Peripheral 6.  |
| PIN_FN_7   | Peripheral 7.  |
| PIN_FN_8   | Peripheral 8.  |
| PIN_FN_9   | Peripheral 9.  |
| PIN_FN_10  | Peripheral 10. |
| PIN_FN_11  | Peripheral 11. |
| PIN_FN_12  | Peripheral 12. |
| PIN_FN_13  | Peripheral 13. |
| PIN_FN_14  | Peripheral 14. |
| PIN_FN_15  | Peripheral 15. |
| PIN_FN_16  | Peripheral 16. |
| PIN_FN_17  | Peripheral 17. |

## Enumerator

|           |                |
|-----------|----------------|
| PIN_FN_18 | Peripheral 18. |
| PIN_FN_19 | Peripheral 19. |
| PIN_FN_20 | Peripheral 20. |
| PIN_FN_21 | Peripheral 21. |
| PIN_FN_22 | Peripheral 22. |
| PIN_FN_23 | Peripheral 23. |
| PIN_FN_24 | Peripheral 24. |
| PIN_FN_25 | Peripheral 25. |
| PIN_FN_26 | Peripheral 26. |
| PIN_FN_27 | Peripheral 27. |
| PIN_FN_28 | Peripheral 28. |
| PIN_FN_29 | Peripheral 29. |
| PIN_FN_30 | Peripheral 30. |
| PIN_FN_31 | Peripheral 31. |
| PIN_FN_32 | Peripheral 32. |

**16.145.2.3 pin\_fn\_t [3/3]**

enum `PinIO::pin_fn_t`

Pin Function modes to configure the managed pin(s).

## Enumerator

|            |                |
|------------|----------------|
| PIN_FN_0   | Peripheral 0.  |
| PIN_FN_1   | Peripheral 1.  |
| PIN_FN_2   | Peripheral 2.  |
| PIN_FN_3   | Peripheral 3.  |
| PIN_FN_IN  | Input.         |
| PIN_FN_OUT | Output.        |
| PIN_FN_IN  | Input.         |
| PIN_FN_OUT | Output.        |
| PIN_FN_A   | Peripheral A.  |
| PIN_FN_B   | Peripheral B.  |
| PIN_FN_C   | Peripheral C.  |
| PIN_FN_D   | Peripheral D.  |
| PIN_FN_IN  | Input.         |
| PIN_FN_OUT | Output.        |
| PIN_FN_0   | Peripheral 0.  |
| PIN_FN_1   | Peripheral 1.  |
| PIN_FN_2   | Peripheral 2.  |
| PIN_FN_3   | Peripheral 3.  |
| PIN_FN_4   | Peripheral 4.  |
| PIN_FN_5   | Peripheral 5.  |
| PIN_FN_6   | Peripheral 6.  |
| PIN_FN_7   | Peripheral 7.  |
| PIN_FN_8   | Peripheral 8.  |
| PIN_FN_9   | Peripheral 9.  |
| PIN_FN_10  | Peripheral 10. |

## Enumerator

|           |                |
|-----------|----------------|
| PIN_FN_11 | Peripheral 11. |
| PIN_FN_12 | Peripheral 12. |
| PIN_FN_13 | Peripheral 13. |
| PIN_FN_14 | Peripheral 14. |
| PIN_FN_15 | Peripheral 15. |
| PIN_FN_16 | Peripheral 16. |
| PIN_FN_17 | Peripheral 17. |
| PIN_FN_18 | Peripheral 18. |
| PIN_FN_19 | Peripheral 19. |
| PIN_FN_20 | Peripheral 20. |
| PIN_FN_21 | Peripheral 21. |
| PIN_FN_22 | Peripheral 22. |
| PIN_FN_23 | Peripheral 23. |
| PIN_FN_24 | Peripheral 24. |
| PIN_FN_25 | Peripheral 25. |
| PIN_FN_26 | Peripheral 26. |
| PIN_FN_27 | Peripheral 27. |
| PIN_FN_28 | Peripheral 28. |
| PIN_FN_29 | Peripheral 29. |
| PIN_FN_30 | Peripheral 30. |
| PIN_FN_31 | Peripheral 31. |
| PIN_FN_32 | Peripheral 32. |

## 16.145.2.4 pull\_t

enum [PinIO::pull\\_t](#)  
**pull\_t**

## 16.145.3 Constructor &amp; Destructor Documentation

## 16.145.3.1 PinIO() [1/2]

```
PinIO::PinIO (
 uint32_t port,
 uint32_t pin) [inline], [constexpr]
```

Construct a [PinIO](#) for a specific cpu pin.

## Parameters

|             |                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------|
| <i>port</i> | The zero-indexed numeric instance number of the desired Pio module. (PIOA=0, PIOB=1, PIOC=2, etc.)     |
| <i>pin</i>  | The zero-indexed pin number within the requested port that the new <a href="#">PinIO</a> is to manage. |

Example usage for pin 'PIOD\_24'

```
PinIO LED1(3, 24);
```

## 16.145.3.2 PinIO() [2/2]

```
PinIO::PinIO (
 const PinIO & rhs) [inline], [constexpr]
```

Construct a copy of another [PinIO](#).

## Parameters

|            |                                                   |
|------------|---------------------------------------------------|
| <i>rhs</i> | The <a href="#">PinIO</a> to construct a copy of. |
|------------|---------------------------------------------------|

**16.145.4 Member Function Documentation****16.145.4.1 function()**

```
void PinIO::function (
 pin_fn_t fn) const [inline]
```

Set the pin function for the managed pin(s).

## Parameters

|           |                                                        |
|-----------|--------------------------------------------------------|
| <i>fn</i> | The pin function to be assigned to the managed pin(s). |
|-----------|--------------------------------------------------------|

**16.145.4.2 getFn()**

```
int8_t PinIO::getFn () [inline]
```

Get the pin function for the managed pin(s).

## Returns

The pin function as a [pin\\_fn\\_t](#) of the managed pin(s). Returns -1 if pin is not an I/O (ex: GND, VCC)

**16.145.4.3 multidrv()**

```
void PinIO::multidrv (
 bool enable) const [inline]
```

Configure the multidrive/open-drain driver for the pin(s).

## Parameters

|               |                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------|
| <i>enable</i> | True: configure the output driver as Open Drain. False: configure the output driver as Push-Pull. |
|---------------|---------------------------------------------------------------------------------------------------|

**16.145.4.4 operator bool()**

```
PinIO::operator bool () const [inline]
```

Read the line state of the pin(s).

## Returns

The line state of the pin(s): true = High, false = Low

**16.145.4.5 operator"!()**

```
bool PinIO::operator! () const [inline]
```

Return the opposite of the *driven* value of the pin(s).

## Returns

The opposite of the *driven* value of the pin(s).

**16.145.4.6 operator=() [1/2]**

```
bool PinIO::operator= (
 bool val) [inline]
```

Assign a driven value to the pin(s).

## Parameters

|            |                                                                |
|------------|----------------------------------------------------------------|
| <i>val</i> | The value to be driven on the pin(s): true = High, false = Low |
|------------|----------------------------------------------------------------|

## Returns

The driven value for the pin(s).

**16.145.4.7 operator=()** [2/2]

```
PinIO & PinIO::operator= (
 const PinIO & rhs) [inline]
```

Assign a driven value to the pin(s) based on the line state of another pin(s).

## Parameters

|            |                                                           |
|------------|-----------------------------------------------------------|
| <i>rhs</i> | The pin(s) who's line state will be assigned to this pin. |
|------------|-----------------------------------------------------------|

## Returns

A copy of the [PinIO](#) being assigned to.

**16.145.4.8 PullDown()**

```
void PinIO::PullDown (
 bool enable) const [inline]
```

Configure the pad Pull Down resistor for the pin(s).

## Parameters

|               |                                                                   |
|---------------|-------------------------------------------------------------------|
| <i>enable</i> | True: enable the pad Pull Down. False: disable the pad Pull Down. |
|---------------|-------------------------------------------------------------------|

**16.145.4.9 PullUp()**

```
void PinIO::PullUp (
 bool enable) const [inline]
```

Configure the pad Pull Up resistor for the pin(s).

## Parameters

|               |                                                               |
|---------------|---------------------------------------------------------------|
| <i>enable</i> | True: enable the pad Pull Up. False: disable the pad Pull Up. |
|---------------|---------------------------------------------------------------|

**16.145.4.10 read()**

```
bool PinIO::read () const [inline]
```

Configure the pin as an input then return the line state.

## Returns

The line state of the pin(s).

**16.145.4.11 readBack()**

```
bool PinIO::readBack () const [inline]
```

Read the state of the pin(s) line state *without* changing the pin function or direction.

**Returns**

The line state of the pin(s).

**16.145.4.12 setFn()**

```
void PinIO::setFn (
 pin_fn_t fn) const
```

Set the pin function for the managed pin(s).

**Parameters**

|           |                                                        |
|-----------|--------------------------------------------------------|
| <i>fn</i> | The pin function to be assigned to the managed pin(s). |
|-----------|--------------------------------------------------------|

**16.145.4.13 setHighStrength()**

```
void PinIO::setHighStrength (
 bool bHighDrive) [inline]
```

Configure the drive strength of the output driver for the pin(s).

**Parameters**

|                   |                                                                                                                       |
|-------------------|-----------------------------------------------------------------------------------------------------------------------|
| <i>bHighDrive</i> | True: configure the output driver for High Drive strength. False: configure the output driver for Low Drive strength. |
|-------------------|-----------------------------------------------------------------------------------------------------------------------|

**16.145.4.14 setPull()**

```
void PinIO::setPull (
 pull_t pull)
```

Configure the pad Pull resistor for the pin(s).

**Parameters**

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>pull</i> | direction and strength (if available) for the given pin |
|-------------|---------------------------------------------------------|

**16.145.4.15 tgl()**

```
bool PinIO::tgl () const [inline]
```

Toggle the driven value for the pin(s).

**Returns**

The previously driven value for the pin(s).

**16.145.4.16 toggle()**

```
bool PinIO::toggle () const [inline]
```

Toggle the driven value for the pin(s).

**Returns**

The previously driven value for the pin(s).

The documentation for this class was generated from the following files:

- coldfire/cpu/MCF5441X/include/cpu\_pins.h
- cortex-m7/cpu/SAME70/include/cpu\_pins.h
- cortex-m7/cpu/MIMXRT10xx/include/cpu\_pins.h

**16.146 PinIOArray2 Class Reference**

```
#include <pins.h>
```

**16.146.1 Detailed Description**

The following class is created to assist in processor pin configuration and operation for GPIO and special functions. Each pin is defined by the system in the "\nburn\<Platform>\system" and "\nburn\<Platform>\include" directories for the specific platform you are using.

Examples: P1[5] = 1; Set pin P1-5 high P2[30] = 0; Set pin P2-30 low P1[5].hiz(); Set pin P1-5 to be a high impedance input BOOL bpinstate = P2[30]; Set the pin to be an input and read if ( !P2[30] ) iprintf( "The pin is low" );

The documentation for this class was generated from the following file:

- MODM7AE70/include/pins.h

**16.147 PinIOArrayJ1 Class Reference**

```
#include <pins.h>
```

**16.147.1 Detailed Description**

The following class is created to assist in processor pin configuration and operation for GPIO and special functions. Each pin is defined by the system in the "\nburn\<Platform>\system" and "\nburn\<Platform>\include" directories for the specific platform you are using.

Examples: P1[5] = 1; Set pin P1-5 high P2[30] = 0; Set pin P2-30 low P1[5].hiz(); Set pin P1-5 to be a high impedance input BOOL bpinstate = P2[30]; Set the pin to be an input and read if ( !P2[30] ) iprintf( "The pin is low" );

The documentation for this class was generated from the following file:

- SBE70LC/include/pins.h

**16.148 PinIOJ1Array Class Reference**

```
#include <pins.h>
```

**16.148.1 Detailed Description**

The following class is created to assist in processor pin configuration and operation for GPIO and special functions. Each pin is defined by the system in the "\Nburn\<Platform>\system" and "\Nburn\<Platform>\include" directories for the specific platform you are using.

Examples: J1[5] = 1; Set pin J1-5 high J2[30] = 0; Set pin J2-30 low J1[5].hiz(); Set pin J1-5 to be a high impedance input BOOL bpinstate = J2[30]; Set the pin to be an input and read if ( !J2[30] ) iprintf( "The pin is low" );

The documentation for this class was generated from the following file:

- MOD5441X/include/pins.h

## 16.149 PinVector< n > Class Template Reference

GPIO Pin Vector Class [PinVector](#) is a template instantiation of the [\\_PinVector](#) class, allowing for minimal storage requirements for potentially large vectors, without heavy code duplication due to template copies.

```
#include <cpu_pins.h>
```

Inherits [\\_PinVector](#), and [\\_PinVector](#).

### Public Member Functions

- [PinVector](#) ()  
*Bare constructor for a [PinVector](#), where the bit configurations will be made later.*
- [PinVector](#) ([PinIO](#) \*initpins, uint32\_t pinCount)  
*[PinVector](#) Constructor, where pin configurations will be made at the time of construction.*
- uint32\_t [operator=](#) (uint32\_t val)  
*Assign a value to the [PinVector](#) Bus.*

### Public Member Functions inherited from [\\_PinVector](#)

- uint32\_t [operator=](#) (uint32\_t val)  
*Assign a value to the [\\_PinVector](#) Bus.*
- [PinIO](#) [operator\[\]](#) (int idx)  
*Access the [PinIO](#) for a specific bit position in the [\\_PinVector](#).*
- void [config](#) (uint32\_t idx, [PinIO](#) cfg)  
*Set the [PinIO](#) that will be used for a given bit position in the [\\_PinVector](#).*
- void [config](#) ([PinIO](#) \*pinCfgs, uint32\_t count)  
*Configure the [\\_PinVector](#) based on an array of [PinIO](#)s. The index of the [PinIO](#) in the configuration array will determine the bit position within the [\\_PinVector](#) that that [PinIO](#) represents.*
- [operator uint32\\_t](#) () const  
*Read the line state of the [\\_PinVector](#) bus.*

### 16.149.1 Detailed Description

```
template<uint8_t n>
```

```
class PinVector< n >
```

GPIO Pin Vector Class [PinVector](#) is a template instantiation of the [\\_PinVector](#) class, allowing for minimal storage requirements for potentially large vectors, without heavy code duplication due to template copies.

#### Parameters

|                                     |                                                                                                                        |
|-------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| <a href="#">PinVector::pinStore</a> | pinStore is the actual storage allocation for large <a href="#">PinVector</a> 's <a href="#">PinIO</a> configurations. |
|-------------------------------------|------------------------------------------------------------------------------------------------------------------------|

## 16.149.2 Constructor & Destructor Documentation

### 16.149.2.1 PinVector()

```
template<uint8_t n>
PinVector< n >::PinVector (
 PinIO * initpins,
 uint32_t pinCount) [inline]
```

[PinVector](#) Constructor, where pin configurations will be made at the time of construction.

#### Parameters

|                 |                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------|
| <i>initpins</i> | The array of <a href="#">PinIO</a> s to be used to configure the <a href="#">PinVector</a> . |
| <i>pinCount</i> | The number of <a href="#">PinIO</a> s in the configuration array to be used.                 |



### 16.149.3 Member Function Documentation

#### 16.149.3.1 operator=()

```
template<uint8_t n>
uint32_t PinVector< n >::operator= (
 uint32_t val) [inline]
```

Assign a value to the [PinVector](#) Bus.

##### Parameters

|            |                                 |
|------------|---------------------------------|
| <i>val</i> | The value to assign to the bus. |
|------------|---------------------------------|

##### Returns

The value driven on the bus.

The documentation for this class was generated from the following files:

- cortex-m7/cpu/SAME70/include/cpu\_pins.h
- cortex-m7/cpu/MIMXRT10xx/include/cpu\_pins.h

## 16.150 pool\_buffer Struct Reference

Main buffer structure for network and serial communication.

```
#include <buffers.h>
```

Inherits OS\_FIFO\_EL.

##### Public Attributes

- [VPoolPtr](#) **pPoolNext**  
*Pointer to next buffer in linked list.*
- [VPoolPtr](#) **pPoolPrev**  
*Pointer to previous buffer in linked list.*
- [vuint32\\_t](#) **dwTime**  
*Timestamp or checksum storage.*
- [vuint32\\_t](#) **dwTimeFraction**  
*Fractional timestamp component.*
- [vuint16\\_t](#) **usedsize**  
*Number of bytes currently used in pData.*
- [uint8\\_t](#) **bBuffer\_state**  
*Buffer ownership state (BO\_\* values)*
- [uint8\\_t](#) **bBufferFlag**  
*Buffer state flags (BS\_\* values)*
- [uint8\\_t](#) **bUsageCount**  
*Reference count for shared buffers.*
- [uint8\\_t](#) **bInterfaceNumber**  
*Network interface number.*
- [uint8\\_t](#) **pData** [[ETHER\\_BUFFER\\_SIZE](#)]  
*Buffer data payload (1548 bytes)*

### 16.150.1 Detailed Description

Main buffer structure for network and serial communication.

The [pool\\_buffer](#) structure is the fundamental building block for all network and serial data transfers in the Net↔Burner system. Each buffer contains data payload, ownership state, and linking information for efficient memory management.

The documentation for this struct was generated from the following file:

- buffers.h

## 16.151 PsudeoHeader Struct Reference

UDP/TCP IPv4 Pseudo-header Structure (bits 0 through 95 of the pseudo-header)

```
#include <ip.h>
```

### Public Attributes

- **beuint16\_t proto**  
*Protocol number (UDP=17, TCP=6) - note: only 8 bits used, padded to 16 bits.*
- **beuint16\_t len**  
*Length of UDP/TCP header plus data (not including pseudo-header itself)*
- **IPADDR4 srcip**  
*Source IP address (same as in IP header)*
- **IPADDR4 dstip**  
*Destination IP address (same as in IP header)*

### 16.151.1 Detailed Description

UDP/TCP IPv4 Pseudo-header Structure (bits 0 through 95 of the pseudo-header)

The pseudo-header is a conceptual construct used in calculating UDP and TCP checksums. It is NOT transmitted as part of the packet, but is included in the checksum calculation to provide protection against misrouted packets.

#### WARNING:

This structure has the elements in a different order than typically defined in RFCs. The IP checksum algorithm is order-invariant on 16-bit values, so this alternate ordering allows the structure to map directly over sections of UDP and TCP packets without requiring data copying or rearrangement.

#### Warning

Do not change the field order - it is specifically designed for efficient overlay onto packet structures.

The documentation for this struct was generated from the following file:

- ip.h

## 16.152 MQTT::PublishRequest Struct Reference

Structure used to specify details for a Publish call.

```
#include <mqtt_msg_reqs.h>
```

### 16.152.1 Detailed Description

Structure used to specify details for a Publish call.

The documentation for this struct was generated from the following file:

- mqtt\_msg\_reqs.h

## 16.153 QSPI\_Record Struct Reference

This struct contains the major variables/configurations used for a QSPI transfer.

```
#include <qspi.h>
```

**Public Attributes**

- volatile uint8\_t \* **pQSPIRxbuf**  
*This pointer is used to track the locations in memory where data will be read from the peripheral.*
- volatile uint8\_t \* **pQSPITxbuf**  
*This pointer is used to track the locations in memory where data will be written to the peripheral.*
- uint8\_t **BitsPerQueue**  
*This is the number of bits per transfer, (value = 8 - 32).*
- uint32\_t **QSPI\_SizeLeft**  
*This is the number of bytes left in the transfer.*
- uint16\_t **Command\_Mask**  
*This is a partial configuration for the queue's command reg.*
- OS\_SEM \* **QSPI\_Sem**  
*This is a pointer to an external semaphore provided by [QSPIStart\(\)](#).*

**16.153.1 Detailed Description**

This struct contains the major variables/configurations used for a QSPI transfer.  
The documentation for this struct was generated from the following file:

- qspi.h

**16.154 fifo\_buffer\_storage::ReadBufPtr Class Reference**

RAII accessor for reading from buffer storage.  
#include <buffers.h>

**16.154.1 Detailed Description**

RAII accessor for reading from buffer storage.  
Provides safe access to internal buffer read pointer with automatic locking and cleanup. Not IRQ safe.  
The documentation for this class was generated from the following file:

- buffers.h

**16.155 SAI\_cfg\_t Struct Reference**

Complete configuration structure for the SAI driver.  
#include <sai.h>

**Public Attributes**

- uint32\_t txClkFreq {12288000}
- uint32\_t rxClkFreq {12288000}
- rxTxSync\_t rxTxSync {RXTX\_ASYNC}
- I2S\_Type \* sai
- SAI\_rtx\_cfg\_t rx
- SAI\_rtx\_cfg\_t tx

**16.155.1 Detailed Description**

Complete configuration structure for the SAI driver.  
This structure encapsulates all configuration parameters for both the transmit and receive channels of the SAI peripheral, along with shared parameters like clock frequencies and synchronization mode.  
Clock Configuration  
Clock frequencies determine the sample rate:

- Sample Rate = Clock Frequency / (bits per word x words per frame)
- Common audio rates: 8kHz, 16kHz, 44.1kHz, 48kHz, 96kHz

Example for 48kHz stereo 24-bit:

- Clock = 48000 x 24 x 2 = 2,304,000 Hz

#### Usage

For most applications, use one of the predefined configurations:

- SAI\_I2S\_SLAVE\_24\_TXRX\_RK: I2S slave, 24-bit, full duplex, RX clock
- SAI\_LJUST\_SLAVE\_16\_TXRX\_TK: Left-justified slave, 16-bit, TX clock

Or customize from a base configuration:

```
SAI_cfg_t myConfig = SAI_I2S_SLAVE_24_TXRX_RK;
myConfig.txClkFreq = 3072000; // Adjust clock for 16kHz sample rate
myConfig.rx.enable = false; // Transmit only
```

## 16.155.2 Member Data Documentation

### 16.155.2.1 rx

```
SAI_rxtx_cfg_t SAI_cfg_t::rx
```

Receive channel configuration

### 16.155.2.2 rxClkFreq

```
uint32_t SAI_cfg_t::rxClkFreq {12288000}
```

Receive bit clock frequency in Hz. Default: 12.288 MHz

### 16.155.2.3 rxTxSync

```
rxTxSync_t SAI_cfg_t::rxTxSync {RXTX_ASYNC}
```

Synchronization mode between RX and TX channels

### 16.155.2.4 sai

```
I2S_Type* SAI_cfg_t::sai
```

Pointer to the SAI hardware instance (SAI1, SAI2, or SAI3)

### 16.155.2.5 tx

```
SAI_rxtx_cfg_t SAI_cfg_t::tx
```

Transmit channel configuration

### 16.155.2.6 txClkFreq

```
uint32_t SAI_cfg_t::txClkFreq {12288000}
```

Transmit bit clock frequency in Hz. Must match the actual clock rate for proper operation. Default: 12.288 MHz (512 x 48kHz for 24-bit stereo)

The documentation for this struct was generated from the following file:

- [sai.h](#)

## 16.156 SAI\_rxtx\_cfg\_t Struct Reference

Configuration structure for one direction (TX or RX) of the SAI module.

```
#include <sai.h>
```

## Public Attributes

- bool `enable` {false}
- uint8\_t `startDly` {1}
- startCond\_t `startCond` {START\_FRAME\_EDGE}
- clkGate\_t `clkGate` {CLK\_GATE\_CONTINUOUS}
- clkOut\_t `clkOut` {CLK\_OUT\_CONTINUOUS}
- clkSrc\_t `clkSrc` {CLK\_SRC\_MCK}
- clkCh\_t `clkCh` {CLK\_CH\_CH}
- clkPol\_t `clkPol` {CLK\_ACTIVE\_HIGH}
- uint8\_t `syncLen` {0}
- frameEdge\_t `syncEdge` {FRAME\_SYNC\_FALLING}
- frameSyncOut\_t `syncOut` {FRAME\_SYNC\_INPUT}
- frameSyncCh\_t `syncCh` {SYNC\_CH\_CH}
- uint8\_t `wordsPerFrame` {2}
- dataPacking\_t `dataPacking` {BUFFER\_WORD32\_RIGHT}
- bool `unifyDataStream` {false}
- bitOrder\_t `bitOrder` {MOST\_SIG\_FIRST}
- bool `lineIdleState` {1}
- uint8\_t `bitsPerWord` {24}
- uint32\_t `wordMask` {0}
- bufferDepletionBehavior\_t `depletionBehavior` {DEPLETED\_PAUSE}

### 16.156.1 Detailed Description

Configuration structure for one direction (TX or RX) of the SAI module.

This structure contains all parameters needed to configure either the transmit or receive channel of the SAI peripheral. The same structure type is used for both directions, with appropriate defaults provided by predefined configurations.

Timing Configuration

The SAI operates on a frame-based model where:

- A **frame** consists of multiple words
- Each **word** consists of multiple bits
- **Frame sync** marks the start of each frame
- **Bit clock** times each bit transfer

#### Example Configuration

```
SAI_rxtx_cfg_t txConfig;
txConfig.enable = true;
txConfig.startDly = 1; // Standard I2S: 1 bit delay
txConfig.startCond = START_FRAME_EDGE;
txConfig.clkOut = CLK_OUT_INPUT; // Slave mode
txConfig.clkSrc = CLK_SRC_EXT; // External clock
txConfig.wordsPerFrame = 2; // Stereo
txConfig.bitsPerWord = 24; // 24-bit samples
txConfig.dataPacking = BUFFER_WORD32_RIGHT;
```

### 16.156.2 Member Data Documentation

#### 16.156.2.1 bitOrder

```
bitOrder_t SAI_rxtx_cfg_t::bitOrder {MOST_SIG_FIRST}
```

Bit transmission order (MSB first is standard)

#### 16.156.2.2 bitsPerWord

```
uint8_t SAI_rxtx_cfg_t::bitsPerWord {24}
```

Number of bits per word (valid audio sample size). Common values: 8, 16, 24, 32

### 16.156.2.3 clkCh

```
clkCh_t SAI_rxtx_cfg_t::clkCh {CLK_CH_CH}
```

Clock channel selection (default/RX/TX)

### 16.156.2.4 clkGate

```
clkGate_t SAI_rxtx_cfg_t::clkGate {CLK_GATE_CONTINUOUS}
```

Clock gating mode for power management

### 16.156.2.5 clkOut

```
clkOut_t SAI_rxtx_cfg_t::clkOut {CLK_OUT_CONTINUOUS}
```

Clock output mode (input/continuous output/gated output)

### 16.156.2.6 clkPol

```
clkPol_t SAI_rxtx_cfg_t::clkPol {CLK_ACTIVE_HIGH}
```

Clock polarity (active high/low)

### 16.156.2.7 clkSrc

```
clkSrc_t SAI_rxtx_cfg_t::clkSrc {CLK_SRC_MCK}
```

Clock source selection (master clock or external)

### 16.156.2.8 dataPacking

```
dataPacking_t SAI_rxtx_cfg_t::dataPacking {BUFFER_WORD32_RIGHT}
```

Buffer word format and data justification

### 16.156.2.9 depletionBehavior

```
bufferDepletionBehavior_t SAI_rxtx_cfg_t::depletionBehavior {DEPLETED_PAUSE}
```

DMA behavior when buffer queue is exhausted

### 16.156.2.10 enable

```
bool SAI_rxtx_cfg_t::enable {false}
```

Enable this channel (true = enabled, false = disabled)

### 16.156.2.11 lineIdleState

```
bool SAI_rxtx_cfg_t::lineIdleState {1}
```

Logic level to drive on data line when idle (between words).

- true: Drive high when idle
- false: Drive low when idle

### 16.156.2.12 startCond

```
startCond_t SAI_rxtx_cfg_t::startCond {START_FRAME_EDGE}
```

Starting condition that triggers data transfer

### 16.156.2.13 startDly

```
uint8_t SAI_rxtx_cfg_t::startDly {1}
```

Data delay in bit clocks from frame sync to first data bit.

- I2S standard: 1 bit delay

- Left-justified: 0 bit delay
- Right-justified: varies based on word size

#### 16.156.2.14 syncCh

```
frameSyncCh_t SAI_rxtx_cfg_t::syncCh {SYNC_CH_CH}
```

Frame sync channel selection (default/RX/TX)

#### 16.156.2.15 syncEdge

```
frameEdge_t SAI_rxtx_cfg_t::syncEdge {FRAME_SYNC_FALLING}
```

Frame sync edge defining frame boundary

#### 16.156.2.16 syncLen

```
uint8_t SAI_rxtx_cfg_t::syncLen {0}
```

Frame sync pulse width in bit clocks.

- 0: Single bit clock wide (typical for I2S)
- >0: Specified number of bit clocks

#### 16.156.2.17 syncOut

```
frameSyncOut_t SAI_rxtx_cfg_t::syncOut {FRAME_SYNC_INPUT}
```

Frame sync output mode (input/negative pulse/positive pulse)

#### 16.156.2.18 unifyDataStream

```
bool SAI_rxtx_cfg_t::unifyDataStream {false}
```

Unify data stream across multiple SAI data lines. When true, treats multiple data lines as a single stream.

#### 16.156.2.19 wordMask

```
uint32_t SAI_rxtx_cfg_t::wordMask {0}
```

Word slot mask for frames with multiple words. Each bit represents a word slot:

- 0 = word slot active (transmitted/received)
- 1 = word slot masked (skipped)

Example for stereo with left channel only: 0x00000002

#### 16.156.2.20 wordsPerFrame

```
uint8_t SAI_rxtx_cfg_t::wordsPerFrame {2}
```

Number of words contained in each frame.

- Mono: 1 word per frame
- Stereo: 2 words per frame
- Multi-channel: N words per frame

The documentation for this struct was generated from the following file:

- [sai.h](#)

## 16.157 SAICtx\_t Class Reference

Main driver class for SAI peripheral operation.

```
#include <sai.h>
```

## Public Types

- enum [ctxState\\_t](#) {  
[CTX\\_STATE\\_STOPPED](#) , [CTX\\_STATE\\_INIT](#) , [CTX\\_STATE\\_TX](#) , [CTX\\_STATE\\_RX](#) ,  
[CTX\\_STATE\\_TXRX](#) }

*Current operational state of the SAI driver.*

## Public Member Functions

- int [Init](#) (const [SAI\\_cfg\\_t](#) &cfg)  
*Initialize the SAI hardware and driver context.*
- void [Shutdown](#) ()  
*Shut down the SAI hardware and release driver resources.*
- int [GetMaxTxBuffers](#) ()  
*Get the maximum number of transmit buffers that can be queued.*
- int [GetMaxRxBuffers](#) ()  
*Get the maximum number of receive buffers that can be queued.*
- int [getCurrentConfig](#) ([SAI\\_cfg\\_t](#) &cfg)  
*Retrieve the current active configuration.*
- [ctxState\\_t](#) [getState](#) () const  
*Get the current operational state of the driver.*
- int [TransmitBuffer](#) (void \*buffer, uint32\_t bufferLen, bool waitIfNeeded, bool enableHalfNotify=false)  
*Queue a buffer for transmission.*
- int [ReadyReceiveBuffer](#) (void \*buffer, uint32\_t bufferLen, bool waitIfNeeded, bool enableHalfNotify=false)  
*Queue a buffer for receiving data.*
- void [RegisterTxBufferCB](#) ([SAI\\_BufferDoneFn\\_t](#) cb)  
*Register a callback for transmit buffer completion.*
- void [RegisterRxBufferCB](#) ([SAI\\_BufferDoneFn\\_t](#) cb)  
*Register a callback for receive buffer completion.*
- void [clrStickyTxError](#) ()  
*Clear the sticky transmit error flag.*
- void [clrStickyRxError](#) ()  
*Clear the sticky receive error flag.*
- uint8\_t [getStickyTxError](#) ()  
*Get the sticky transmit error status.*
- uint8\_t [getStickyRxError](#) ()  
*Get the sticky receive error status.*
- void [dump](#) ()  
*Dump driver state and statistics to the console.*

### 16.157.1 Detailed Description

Main driver class for SAI peripheral operation.

[SAICtx\\_t](#) provides a high-level, buffer-oriented interface to the SAI peripheral. The driver manages DMA transfers, buffer chains, and provides callbacks for buffer completion events.

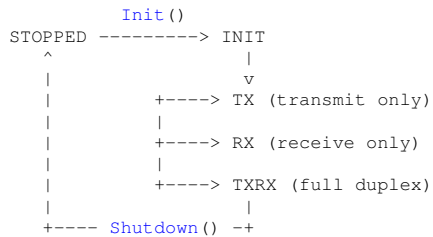
Driver Features

- **DMA-based transfers:** All data movement handled by eDMA for CPU efficiency
- **Buffer queuing:** Queue up to [SAI\\_TXBD\\_COUNT](#) transmit and [SAI\\_RXBD\\_COUNT](#) receive buffers
- **Circular buffering:** Automatic chaining of buffers for continuous streaming
- **Half-transfer notifications:** Optional callbacks at 50% buffer completion
- **Error handling:** Comprehensive error detection and reporting



- **Thread-safe:** Uses semaphores for safe operation from multiple tasks

#### Driver States



#### Thread Safety

The driver uses internal semaphores to synchronize buffer operations:

- Multiple tasks can queue buffers safely
- Callbacks execute in interrupt context
- Use RTOS primitives in callbacks to signal tasks

#### Performance Considerations

- Buffer size affects latency: larger buffers = more latency but less overhead
- Keep 2-3 buffers queued for gapless streaming
- Process completed buffers quickly to avoid starvation
- Use half-transfer callbacks for lower latency processing

#### Warning

Only one [SAICtx\\_t](#) instance should be used per SAI hardware peripheral. Multiple instances controlling the same hardware will conflict.

## 16.157.2 Member Enumeration Documentation

### 16.157.2.1 ctxState\_t

enum [SAICtx\\_t::ctxState\\_t](#)

Current operational state of the SAI driver.

#### Enumerator

|                   |                                                         |
|-------------------|---------------------------------------------------------|
| CTX_STATE_STOPPED | Driver is stopped; no transfers active                  |
| CTX_STATE_INIT    | Driver initialized but no channels active               |
| CTX_STATE_TX      | Transmit channel active only                            |
| CTX_STATE_RX      | Receive channel active only                             |
| CTX_STATE_TXRX    | Both transmit and receive channels active (full duplex) |

## 16.157.3 Member Function Documentation

### 16.157.3.1 clrStickyRxError()

void [SAICtx\\_t::clrStickyRxError\(\)](#) [inline]

Clear the sticky receive error flag.

Resets the internal error flag that latches when an RX error occurs. Call this after handling an error condition to re-enable error detection.

### 16.157.3.2 clrStickyTxError()

```
void SAICtx_t::clrStickyTxError () [inline]
```

Clear the sticky transmit error flag.

Resets the internal error flag that latches when a TX error occurs. Call this after handling an error condition to re-enable error detection.

### 16.157.3.3 dump()

```
void SAICtx_t::dump ()
```

Dump driver state and statistics to the console.

Prints detailed information about the driver's current state, including:

- Configuration parameters
- DMA status
- Buffer queue status
- Error counts
- Transfer statistics

Useful for debugging and performance analysis.

```
// Dump SAI driver state
sai.dump();
```

### 16.157.3.4 getCurrentConfig()

```
int SAICtx_t::getCurrentConfig (
 SAI_cfg_t & cfg)
```

Retrieve the current active configuration.

Reads back the current hardware configuration and populates the provided configuration structure. Useful for verifying settings or creating a modified configuration based on the current state.

#### Parameters

|     |     |                                                            |
|-----|-----|------------------------------------------------------------|
| out | cfg | Configuration structure to populate with current settings. |
|-----|-----|------------------------------------------------------------|

#### Returns

0 on success

Negative on failure (driver not initialized)

```
SAI_cfg_t currentConfig;
if (sai.getCurrentConfig(currentConfig) == 0) {
 iprintf("TX Clock: %lu Hz\n", currentConfig.txClkFreq);
}
```

### 16.157.3.5 GetMaxRxBuffers()

```
int SAICtx_t::GetMaxRxBuffers () [inline]
```

Get the maximum number of receive buffers that can be queued.

#### Returns

Maximum number of RX buffers (SAI\_RXBD\_COUNT)

### 16.157.3.6 GetMaxTxBuffers()

```
int SAICtx_t::GetMaxTxBuffers () [inline]
```

Get the maximum number of transmit buffers that can be queued.

#### Returns

Maximum number of TX buffers (SAI\_TXBD\_COUNT)

**16.157.3.7 getState()**

```
ctxState_t SAICtx_t::getState () const [inline]
```

Get the current operational state of the driver.

**Returns**

Current driver state (CTX\_STATE\_STOPPED, CTX\_STATE\_INIT, etc.)

```
if (sai.getState() == SAICtx_t::CTX_STATE_TXRX) {
 iprintf("Full duplex operation active\n");
}
```

**16.157.3.8 getStickyRxError()**

```
uint8_t SAICtx_t::getStickyRxError () [inline]
```

Get the sticky receive error status.

Returns the latched error code from the most recent RX error. The value persists until cleared with [clrStickyRxError\(\)](#).

**Returns**

Error code (non-zero indicates an error occurred)

**16.157.3.9 getStickyTxError()**

```
uint8_t SAICtx_t::getStickyTxError () [inline]
```

Get the sticky transmit error status.

Returns the latched error code from the most recent TX error. The value persists until cleared with [clrStickyTxError\(\)](#).

**Returns**

Error code (non-zero indicates an error occurred)

**16.157.3.10 Init()**

```
int SAICtx_t::Init (
 const SAI_cfg_t & cfg)
```

Initialize the SAI hardware and driver context.

Configures the SAI peripheral according to the provided configuration, sets up DMA channels, and prepares the driver for buffer operations.

**Parameters**

|    |     |                                                                                                                                                     |
|----|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| in | cfg | Configuration structure specifying all SAI parameters. Use one of the predefined configurations (SAI_I2S_SLAVE_*) or create a custom configuration. |
|----|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------|

**Returns**

0 on success

Negative error code on failure:

- Configuration validation errors
- Clock initialization errors
- DMA allocation errors
- Hardware initialization errors

**Note**

Must be called before any buffer operations.

Calling [Init\(\)](#) on an already initialized driver will fail unless [Shutdown\(\)](#) is called first.

```
SAI_cfg_t config = SAI_I2S_SLAVE_24_TXRX_RK;
if (sai.Init(config) < 0) {
 iprintf("SAI initialization failed\n");
 return;
}
```

### 16.157.3.11 ReadyReceiveBuffer()

```
int SAICtx_t::ReadyReceiveBuffer (
 void * buffer,
 uint32_t bufferLen,
 bool waitIfNeeded,
 bool enableHalfNotify = false)
```

Queue a buffer for receiving data.

Provides a buffer to the SAI driver to be filled with received data. The buffer will be added to the receive queue and automatically filled when data arrives. Up to SAI\_RXBD\_COUNT buffers can be queued simultaneously.

#### Parameters

|                         |                                                                                                                   |
|-------------------------|-------------------------------------------------------------------------------------------------------------------|
| <i>buffer</i>           | Pointer to the buffer to be filled with received data. <b>MUST be 32-byte aligned.</b>                            |
| <i>bufferLen</i>        | Length of the buffer in bytes. <b>MUST be a multiple of 32 bytes.</b>                                             |
| <i>waitIfNeeded</i>     | If true, blocks until space is available in the queue. If false, returns immediately with error if queue is full. |
| <i>enableHalfNotify</i> | If true, callback will be invoked at 50% buffer fill (with doneInvalid=0) in addition to full completion.         |

#### Returns

0 on success

Negative error code on failure:

- Buffer not aligned correctly
- Buffer size not valid
- Queue full (if waitIfNeeded is false)
- Driver not initialized or RX not enabled

#### Warning

The buffer must remain valid and unchanged until the completion callback indicates it is done (doneInvalid > 0).

#### Note

For continuous reception, queue the next buffer before the previous one completes to avoid losing data.

```
// Declare aligned buffer
alignas(32) uint8_t rxBuf[1024];

// Queue for reception
if (sai.ReadyReceiveBuffer(rxBuf, sizeof(rxBuf), true) < 0) {
 iprintf("Failed to queue RX buffer\n");
}
```

### 16.157.3.12 RegisterRxBufferCB()

```
void SAICtx_t::RegisterRxBufferCB (
 SAI_BufferDoneFn_t cb)
```

Register a callback for receive buffer completion.

Sets the function to be called when a receive buffer is filled or encounters an error. The callback executes in interrupt context.

#### Parameters

|           |                                                          |
|-----------|----------------------------------------------------------|
| <i>cb</i> | Callback function pointer, or NULL to disable callbacks. |
|-----------|----------------------------------------------------------|

**Note**

Only one callback can be registered at a time.

The callback executes in interrupt context - keep it brief!

```
void rxComplete(void *buffer, int status) {
 if (status > 0) {
 // Buffer filled successfully
 ProcessAudioData(buffer);
 }
}

sai.RegisterRxBufferCB(rxComplete);
```

**16.157.3.13 RegisterTxBufferCB()**

```
void SAICtx_t::RegisterTxBufferCB (
 SAI_BufferDoneFn_t cb)
```

Register a callback for transmit buffer completion.

Sets the function to be called when a transmit buffer completes or encounters an error. The callback executes in interrupt context.

**Parameters**

|           |                                                          |
|-----------|----------------------------------------------------------|
| <i>cb</i> | Callback function pointer, or NULL to disable callbacks. |
|-----------|----------------------------------------------------------|

**Note**

Only one callback can be registered at a time. Calling this function replaces any previously registered callback.

The callback executes in interrupt context - keep it brief!

```
void txComplete(void *buffer, int status) {
 if (status > 0) {
 // Buffer transmitted successfully
 OSSemPost (&txDoneSem);
 }
}

sai.RegisterTxBufferCB(txComplete);
```

**16.157.3.14 Shutdown()**

```
void SAICtx_t::Shutdown ()
```

Shut down the SAI hardware and release driver resources.

Stops all transfers, disables the SAI peripheral, releases DMA channels, and returns the driver to the stopped state.

**Warning**

Any buffers currently queued or in process will be abandoned. Ensure all important data is processed before calling [Shutdown\(\)](#).

**Note**

After shutdown, [Init\(\)](#) must be called again before using the driver.

```
sai.Shutdown();
iprintf("SAI driver stopped\n");
```

**16.157.3.15 TransmitBuffer()**

```
int SAICtx_t::TransmitBuffer (
 void * buffer,
 uint32_t bufferLen,
 bool waitIfNeeded,
 bool enableHalfNotify = false)
```

Queue a buffer for transmission.

Hands a buffer to the SAI driver for transmission. The buffer will be added to the transmit queue and automatically transmitted when its turn comes. Up to SAI\_TXBD\_COUNT buffers can be queued simultaneously.

## Parameters

|                         |                                                                                                                                                             |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>buffer</i>           | Pointer to the buffer containing data to transmit. <b>MUST be 32-byte aligned</b> (use <code>alignas(32)</code> or <code>attribute((aligned(32)))</code> ). |
| <i>bufferLen</i>        | Length of the buffer in bytes. <b>MUST be a multiple of 32 bytes</b> . Must also align with word width (1, 2, or 4 byte multiples).                         |
| <i>waitIfNeeded</i>     | If true, blocks until space is available in the queue. If false, returns immediately with error if queue is full.                                           |
| <i>enableHalfNotify</i> | If true, callback will be invoked at 50% buffer completion (with <code>doneInvalid=0</code> ) in addition to full completion.                               |

## Returns

0 on success

Negative error code on failure:

- Buffer not aligned correctly
- Buffer size not valid
- Queue full (if `waitIfNeeded` is false)
- Driver not initialized or TX not enabled

## Warning

The buffer must remain valid and unchanged until the completion callback indicates it is done (`doneInvalid > 0`).

Do not modify the buffer contents while it is queued or being transmitted.

## Note

For continuous streaming, queue the next buffer before the previous one completes to avoid gaps in transmission.

```
// Declare aligned buffer
alignas(32) uint8_t txBuf[1024];

// Fill buffer with audio data
FillAudioBuffer(txBuf, sizeof(txBuf));

// Queue for transmission
if (sai.TransmitBuffer(txBuf, sizeof(txBuf), true) < 0) {
 iprintf("Failed to queue TX buffer\n");
}
```

The documentation for this class was generated from the following file:

- [sai.h](#)

## 16.158 SEMC\_cfg\_t Struct Reference

Configuration structure for SEMC (Smart External Memory Controller) interface.

```
#include <semc.h>
```

## Public Attributes

- union {  
    uint32\_t [CRO](#)  
};

*Control Register 0 - Basic Interface Configuration.*

- union {  
    uint32\_t CR1  
};

*Control Register 1 - Primary Timing Configuration.*

- union {  
    uint32\_t CR2  
};

*Control Register 2 - Extended Timing Configuration.*

- PinIO clkPin
- PinIO csPin

### 16.158.1 Detailed Description

Configuration structure for SEMC (Smart External Memory Controller) interface.

This structure configures the timing and control parameters for SEMC chip selects, supporting both synchronous and asynchronous memory devices. The SEMC interface allows connection to external SRAM, PSRAM, NOR Flash, and other parallel memory devices.

The configuration is divided into three control registers (CR0, CR1, CR2) that can be accessed either as structured bit fields or as complete 32-bit values for direct register programming.

#### Note

- All timing parameters marked as "n+1" mean the actual bus timing will be the configured value plus one clock cycle. For example, cs\_setup = 2 results in 3 clock cycles on the bus.
- Parameters marked "async mode only" are ignored in synchronous/burst mode and vice versa. Refer to the clkMode field to determine which mode is active.

---

#### Expand for Example Usage

##### Examples

##### Asynchronous SRAM Configuration (8-bit)

```
SEMC_cfg_t asyncSRAM = {};

// CR0 - Basic configuration
asyncSRAM.busWidth = SEMC_BUS_WIDTH_8BIT;
asyncSRAM.clkMode = SEMC_CLK_ASYNC;
asyncSRAM.muxMode = SEMC_MUX_A8;

// CR1 - Async timing parameters
asyncSRAM.cs_setup = 1; // 2 cycles (1+1)
asyncSRAM.cs_hold_min = 1; // 2 cycles (1+1)
asyncSRAM.addr_setup = 1; // 2 cycles (1+1)
asyncSRAM.addr_hold = 1; // 2 cycles (1+1)
asyncSRAM.wr_en_lo = 2; // 3 cycles (2+1)
asyncSRAM.wr_en_hi = 2; // 3 cycles (2+1)
asyncSRAM.rd_en_lo = 2; // 3 cycles (2+1)
asyncSRAM.rd_en_hi = 2; // 3 cycles (2+1)

// CR2 - Additional async parameters
asyncSRAM.addr_wr_hold = 1; // 2 cycles (1+1)
asyncSRAM.turnaround = 2; // 2 cycles (n, no +1)
asyncSRAM.cs_min_intv = 1; // 2 cycles (1+1)

// Pin assignments
asyncSRAM.csPin = Pins[10];
```

##### Synchronous Burst PSRAM Configuration (16-bit)

```
SEMC_cfg_t syncPSRAM = {};

// CR0 - Synchronous burst configuration
syncPSRAM.busWidth = SEMC_BUS_WIDTH_16BIT;
syncPSRAM.clkMode = SEMC_CLK_SYNC;
syncPSRAM.burstLen = SEMC_BURST_8;
syncPSRAM.muxMode = SEMC_MUX_A8;
syncPSRAM.colAddrWidth = SEMC_COL_ADDR_10BIT;

// CR1 - Basic timing
syncPSRAM.cs_setup = 0; // 1 cycle (0+1)
syncPSRAM.cs_hold_min = 0; // 1 cycle (0+1)
syncPSRAM.addr_setup = 0; // 1 cycle (0+1)
```

```

syncPSRAM.addr_hold = 1; // 1 cycle (n, no +1 in sync mode)

// CR2 - Synchronous/burst timing
syncPSRAM.wr_dat_setup = 1; // 2 cycles (1+1)
syncPSRAM.wr_dat_hold = 1; // 1 cycle (n, no +1)
syncPSRAM.latency = 3; // 3 cycle latency (n, no +1)
syncPSRAM.rd_cycle = 3; // 4 cycles (3+1)
syncPSRAM.rd_hold = 1; // 1 cycle (n, no +1)
syncPSRAM.cs_min_intv = 1; // 2 cycles (1+1)

// Pin assignments
syncPSRAM.clkPin = Pins[15];
syncPSRAM.csPin = Pins[16];

```

### Fast NOR Flash Configuration (Async Mode)

```

SEMC_cfg_t norFlash = {};

// CR0 - 16-bit async configuration
norFlash.busWidth = SEMC_BUS_WIDTH_16BIT;
norFlash.clkMode = SEMC_CLK_ASYNC;
norFlash.muxMode = SEMC_MUX_A8;

// CR1 - Fast async timing for modern NOR Flash
norFlash.cs_setup = 0; // 1 cycle (0+1)
norFlash.cs_hold_min = 0; // 1 cycle (0+1)
norFlash.addr_setup = 0; // 1 cycle (0+1)
norFlash.addr_hold = 0; // 1 cycle (0+1)
norFlash.wr_en_lo = 3; // 4 cycles (3+1)
norFlash.wr_en_hi = 3; // 4 cycles (3+1)
norFlash.rd_en_lo = 4; // 5 cycles (4+1) - longer for read
norFlash.rd_en_hi = 4; // 5 cycles (4+1)

// CR2 - Turnaround time for bus release
norFlash.addr_wr_hold = 1; // 2 cycles (1+1)
norFlash.turnaround = 3; // 3 cycles (n, no +1)
norFlash.cs_min_intv = 2; // 3 cycles (2+1)

// Pin assignment
norFlash.csPin = Pins[20];

```

### Direct Register Programming

```

SEMC_cfg_t directConfig = {};

// Set entire control registers at once
directConfig.CR0 = 0x00000091; // Pre-calculated register value
directConfig.CR1 = 0x22222222; // Pre-calculated timing
directConfig.CR2 = 0x11111111; // Pre-calculated timing

// Pin assignments
directConfig.csPin = Pins[12];

```

## 16.158.2 Member Data Documentation

### 16.158.2.1 [union]

```
union { ... } SEMC_cfg_t
```

Control Register 0 - Basic Interface Configuration.

Contains fundamental interface parameters including bus width, clock mode, burst length, multiplexing mode, and column address width. Can be accessed as individual bit fields or as a complete 32-bit register value.

### 16.158.2.2 [union]

```
union { ... } SEMC_cfg_t
```

Control Register 1 - Primary Timing Configuration.

Defines setup, hold, and enable timing for chip select, address, write enable, and read enable signals. Parameters apply differently depending on clock mode (async vs sync). Most values are "n+1" meaning actual cycles = configured + 1.

### 16.158.2.3 [union]

```
union { ... } SEMC_cfg_t
```

Control Register 2 - Extended Timing Configuration.

Contains additional timing parameters for both synchronous and asynchronous modes. Includes write data timing, latency, read cycle length, turnaround, and minimum chip select interval. Parameters are mode-specific.



**16.158.2.4 addr\_hold**

`uint32_t SEMC_cfg_t::addr_hold`

Address hold time: Async=(n+1), Sync=(n) cycles

**16.158.2.5 addr\_setup**

`uint32_t SEMC_cfg_t::addr_setup`

Address setup time before control signal (n+1 cycles)

**16.158.2.6 addr\_wr\_hold**

`uint32_t SEMC_cfg_t::addr_wr_hold`

Address to write hold time (n+1 cycles, async only)

**16.158.2.7 advHoldLvl**

`bool SEMC_cfg_t::advHoldLvl`

ADV signal hold level

**16.158.2.8 advPol**

`bool SEMC_cfg_t::advPol`

Address valid (ADV) signal polarity

**16.158.2.9 burstLen**

`SEMC_Burst_t SEMC_cfg_t::burstLen`

Burst transfer length for synchronous mode

**16.158.2.10 busWidth**

`SEMC_BusWidth_t SEMC_cfg_t::busWidth`

Data bus width (8-bit or 16-bit)

**16.158.2.11 clkMode**

`SEMC_ClkMode_t SEMC_cfg_t::clkMode`

Clock mode (asynchronous or synchronous)

**16.158.2.12 clkPin**

`PinIO SEMC_cfg_t::clkPin`

Clock output pin assignment (synchronous mode only)

**16.158.2.13 colAddrWidth**

`SEMC_ColAddrWidth_t SEMC_cfg_t::colAddrWidth`

Column address bit width (synchronous mode)

**16.158.2.14 CR0**

`uint32_t SEMC_cfg_t::CR0`

Direct access to complete Control Register 0

**16.158.2.15 CR1**

`uint32_t SEMC_cfg_t::CR1`

Direct access to complete Control Register 1

**16.158.2.16 CR2**

```
uint32_t SEMC_cfg_t::CR2
```

Direct access to complete Control Register 2

**16.158.2.17 cs\_hold\_min**

```
uint32_t SEMC_cfg_t::cs_hold_min
```

Minimum chip select hold time after cycle end (n+1 cycles)

**16.158.2.18 cs\_min\_intv**

```
uint32_t SEMC_cfg_t::cs_min_intv
```

Minimum interval between chip select assertions (n+1 cycles)

**16.158.2.19 cs\_setup**

```
uint32_t SEMC_cfg_t::cs_setup
```

Chip select setup time before address valid (n+1 cycles)

**16.158.2.20 csPin**

```
PinIO SEMC_cfg_t::csPin
```

Chip select output pin assignment

**16.158.2.21 latency**

```
uint32_t SEMC_cfg_t::latency
```

Read data latency cycles (n cycles, sync/burst only)

**16.158.2.22 muxMode**

```
SEMC_AddrMux_t SEMC_cfg_t::muxMode
```

Address/data multiplexing mode

**16.158.2.23 rd\_cycle**

```
uint32_t SEMC_cfg_t::rd_cycle
```

Total read cycle length (n+1 cycles, sync/burst only)

**16.158.2.24 rd\_en\_hi**

```
uint32_t SEMC_cfg_t::rd_en_hi
```

Read enable high period (n+1 cycles, async only)

**16.158.2.25 rd\_en\_lo**

```
uint32_t SEMC_cfg_t::rd_en_lo
```

Read enable low pulse width (n+1 cycles, async only)

**16.158.2.26 rd\_hold**

```
uint32_t SEMC_cfg_t::rd_hold
```

Read data hold time (n cycles, sync/burst only)

**16.158.2.27 RESERVED0**

```
uint8_t SEMC_cfg_t::RESERVED0
```

Reserved bits, must be 0

**16.158.2.28 RESERVED1**

```
uint8_t SEMC_cfg_t::RESERVED1
```

Reserved bit, must be 0

**16.158.2.29 turnaround**

```
uint32_t SEMC_cfg_t::turnaround
```

Bus turnaround time between transfers (n cycles, async only)

**16.158.2.30 wr\_dat\_hold**

```
uint32_t SEMC_cfg_t::wr_dat_hold
```

Write data hold time (n cycles, sync/burst only)

**16.158.2.31 wr\_dat\_setup**

```
uint32_t SEMC_cfg_t::wr_dat_setup
```

Write data setup time (n+1 cycles, sync/burst only)

**16.158.2.32 wr\_en\_hi**

```
uint32_t SEMC_cfg_t::wr_en_hi
```

Write enable high period (n+1 cycles, async only)

**16.158.2.33 wr\_en\_lo**

```
uint32_t SEMC_cfg_t::wr_en_lo
```

Write enable low pulse width (n+1 cycles, async only)

The documentation for this struct was generated from the following file:

- semc.h

**16.159 SerialConfigServlet Class Reference**

Class to run serial configurator.

```
#include <config_server_serial.h>
```

Inherits servlet.

**16.159.1 Detailed Description**

Class to run serial configurator.

The documentation for this class was generated from the following file:

- config\_server\_serial.h

**16.160 SerialRecord Struct Reference**

```
#include <serialrecord.h>
```

**Public Member Functions**

- void [AssignUartNumber](#) (void)
- void [ProcessTCPReadSerialData](#) (void)
- void [ProcessSpecialFrameTCPReadSerialData](#) (void)
- void [ProcessSpecialFrameWriteNetworkData](#) (void)
- void [ProcessSpecialFrameWriteTimeout](#) (void)
- void [MakeTcpConnection](#) (void)

- void [CloseListenPort](#) (void)
- void [MakeUdpConnection](#) (void)
- bool [OkToListen](#) (void)
- void [ProcessTimeouts](#) (void)
- void [ProcessAccept](#) (void)
- void [OpenSerialPort](#) (void)
- void [ProcessReadNetworkData](#) (void)
- void [OpenListenPort](#) (void)
- void [ProcessWriteNetworkData](#) (void)
- void [ProcessWriteSerialData](#) (void)
- void [ProcessSerialError](#) (void)
- void [ProcessListenError](#) (void)
- void [ProcessNetworkError](#) (void)
- void [GetCurrentChannelStatus](#) (char \*buffer)

### 16.160.1 Detailed Description

Class Definition (struct is class with all members public)

### 16.160.2 Member Function Documentation

#### 16.160.2.1 AssignUartNumber()

```
void SerialRecord::AssignUartNumber (
 void)
```

Methods

Assign physical UART number to PortRecord array index member

#### 16.160.2.2 CloseListenPort()

```
void SerialRecord::CloseListenPort (
 void)
```

Close the listening port

#### 16.160.2.3 GetCurrentChannelStatus()

```
void SerialRecord::GetCurrentChannelStatus (
 char * buffer)
```

" Connected to IP xx.xx.xx.xx " Listening on port xx " UDP mode with learned send-to IP Address : xx.xx.xx.xx "  
UDP mode send to Address : xx.xx.xx.xx " Idle

#### 16.160.2.4 MakeTcpConnection()

```
void SerialRecord::MakeTcpConnection (
 void)
```

Initiate a TCP connection

#### 16.160.2.5 MakeUdpConnection()

```
void SerialRecord::MakeUdpConnection (
 void)
```

Initiate a UDP connection

#### 16.160.2.6 OkToListen()

```
bool SerialRecord::OkToListen (
 void)
```

Check if conditions are right to establish a listening port

**16.160.2.7 OpenListenPort()**

```
void SerialRecord::OpenListenPort (
 void)
```

Open the listening port

**16.160.2.8 OpenSerialPort()**

```
void SerialRecord::OpenSerialPort (
 void)
```

Open a serial port

**16.160.2.9 ProcessAccept()**

```
void SerialRecord::ProcessAccept (
 void)
```

Called when connections are waiting to be accepted

**16.160.2.10 ProcessListenError()**

```
void SerialRecord::ProcessListenError (
 void)
```

Process listening errors

**16.160.2.11 ProcessNetworkError()**

```
void SerialRecord::ProcessNetworkError (
 void)
```

Process network errors

**16.160.2.12 ProcessReadNetworkData()**

```
void SerialRecord::ProcessReadNetworkData (
 void)
```

Called when data is received on the network port side

**16.160.2.13 ProcessSerialError()**

```
void SerialRecord::ProcessSerialError (
 void)
```

Process serial errors

**16.160.2.14 ProcessSpecialFrameTCPReadSerialData()**

```
void SerialRecord::ProcessSpecialFrameTCPReadSerialData (
 void)
```

Called to initiate an outgoing TCP connection when data is received on the serial data port side (this version for TCP packet customization logic)

**16.160.2.15 ProcessSpecialFrameWriteNetworkData()**

```
void SerialRecord::ProcessSpecialFrameWriteNetworkData (
 void)
```

Called when write space has just opened up on network port (this version for TCP packet customization logic)

**16.160.2.16 ProcessSpecialFrameWriteTimeout()**

```
void SerialRecord::ProcessSpecialFrameWriteTimeout (
 void)
```

Time-out function used with TCP packet customization logic - waits for a given time to accumulate serial characters before flushing them out the network side

**16.160.2.17 ProcessTCPReadSerialData()**

```
void SerialRecord::ProcessTCPReadSerialData (
 void)
```

Called to initiate an outgoing TCP connection when data is received on the serial data port side

**16.160.2.18 ProcessTimeouts()**

```
void SerialRecord::ProcessTimeouts (
 void)
```

Test all of the communication time-outs Periodically check for live connection at a user-specified interval by sending an empty packet to the target device.

Wait at least two seconds for ACK to return after sending keep-alive packet. Target is lost if ACK was not the last packet received.

**16.160.2.19 ProcessWriteNetworkData()**

```
void SerialRecord::ProcessWriteNetworkData (
 void)
```

Called when write space has just opened up on network port

**16.160.2.20 ProcessWriteSerialData()**

```
void SerialRecord::ProcessWriteSerialData (
 void)
```

Called when write space has just opened up serial port

The documentation for this struct was generated from the following files:

- serialrecord.h
- serialrecord.cpp

**16.161 DHCP::Server Class Reference**

[DHCP Server](#) class Requires a lease allocator to be added in order to function.

```
#include <dhcpcd.h>
```

**16.161.1 Detailed Description**

[DHCP Server](#) class Requires a lease allocator to be added in order to function.

The documentation for this class was generated from the following file:

- dhcpcd.h

**16.162 SimpleBufferObject Class Reference**

Class for web client response buffers.

```
#include <web_buffers.h>
```

Inherits [buffer\\_object](#).

**16.162.1 Detailed Description**

Class for web client response buffers.

The documentation for this class was generated from the following file:

- web\_buffers.h

## 16.163 DHCP::SingleAllocator Class Reference

Stripped down allocator Fails to conform to many standard behaviors, but shows the basics of what needs to be implemented.

```
#include <dhcpd.h>
```

Inherits [DHCP::LeaseAllocator](#).

### 16.163.1 Detailed Description

Stripped down allocator Fails to conform to many standard behaviors, but shows the basics of what needs to be implemented.

The documentation for this class was generated from the following file:

- `dhcpd.h`

## 16.164 SMPoolPtr Class Reference

Smart pointer wrapper for automatic buffer management.

```
#include <buffers.h>
```

### Public Member Functions

- **SMPoolPtr** ()  
*Constructor - automatically allocates a buffer.*
- **SMPoolPtr** (const [PoolPtr](#) pp)  
*Constructor from existing buffer pointer.*
- **~SMPoolPtr** ()  
*Destructor - automatically frees the buffer.*

### 16.164.1 Detailed Description

Smart pointer wrapper for automatic buffer management.

RAII wrapper that automatically allocates a buffer on construction and frees it on destruction. Helps prevent buffer leaks in exception-safe code.

### 16.164.2 Constructor & Destructor Documentation

#### 16.164.2.1 SMPoolPtr()

```
SMPoolPtr::SMPoolPtr (
 const PoolPtr pp) [inline]
```

Constructor from existing buffer pointer.

#### Parameters

|           |                  |
|-----------|------------------|
| <i>pp</i> | Buffer to manage |
|-----------|------------------|

The documentation for this class was generated from the following file:

- `buffers.h`

## 16.165 SPI\_QSPI Class Reference

The Single-Bit SPI mode QSPI Peripheral Class.

```
#include <quadspi.h>
```

Inherits [SPIModule](#).

## Public Member Functions

- [SPI\\_QSPI](#) (uint8\_t QuadSPIModule=DEFAULT\_QUADSPI\_MODULE)  
*Create a SPI object.*
- [SPI\\_QSPI](#) (uint8\_t QuadSPIModule, uint32\_t baudRateInBps, uint8\_t transferSizeInBits=8, uint8\_t peripheralChipSelects=0x00, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)  
*Create a SPI object and Initialize with parameters.*
- virtual uint8\_t [Init](#) (uint32\_t baudRateInBps=2000000, uint8\_t transferSizeInBits=8, uint8\_t peripheralChipSelects=0x00, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0) override  
*Initialize an existing SPI object.*
- virtual uint32\_t [SetBusSpeed](#) (uint32\_t maxSpeed) override  
*Set the SPI bus speed Will attempt to set the desired bus speed. It may be different based on the available system characteristics.*
- virtual uint8\_t [Start](#) (uint8\_t \*transmitBufferPtr, volatile uint8\_t \*receiveBufferPtr, uint32\_t byteCount, int csReturnToInactive=DEASSERT\_AFTER\_LAST) override  
*Start a SPI transfer.*
- virtual uint8\_t [Tx](#) (uint8\_t \*transmitBufferPtr, uint32\_t byteCount, int csReturnToInactive=DEASSERT\_AFTER\_LAST) override  
*Convenience function for unidirectional transmit.*
- uint8\_t [Rx](#) (uint8\_t \*receiveBufferPtr, uint32\_t byteCount, int csReturnToInactive=DEASSERT\_AFTER\_LAST)  
*Convenience function for unidirectional receive.*
- virtual bool [SetCS](#) (uint8\_t CS) override  
*Set the chip select configuration for the SPI object's bus transactions.*

## Public Member Functions inherited from [SPIModule](#)

- [SPIModule](#) (uint8\_t [SPIModule](#))  
*Create a SPI object.*
- [SPIModule](#) (uint8\_t [SPIModule](#), uint32\_t baudRateInBps, uint8\_t transferSizeInBits=8, uint8\_t peripheralChipSelects=0x00, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0, uint8\_t delayBetweenTransfer=0)  
*Create a SPI object and Initialize with parameters.*
- virtual bool [RegisterSem](#) (OS\_SEM \*finishedSem)  
*Register a semaphore for the SPI module.*
- virtual bool [ClrSem](#) ()  
*Clear a semaphore registration.*
- virtual OS\_SEM \* [GetSem](#) ()  
*Obtain a pointer to the SPI finished semaphore.*
- virtual bool [Done](#) ()  
*Function to check SPI status.*
- virtual uint32\_t [GetActualBaudrate](#) ()  
*Returns the active baud rate.*
- [SPIModule](#) (uint8\_t [SPIModule](#))  
*Create a SPI object.*
- [SPIModule](#) (uint8\_t [SPIModule](#), uint32\_t baudRateInBps, uint8\_t transferSizeInBits=8, uint8\_t peripheralChipSelects=0x00, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)  
*Create a SPI object and Initialize with parameters.*
- [SPIModule](#) (uint8\_t [SPIModule](#), uint32\_t baudRateInBps, uint8\_t transferSizeInBits=8, [PinIO](#) gpioChipSelect=PinDef\_INVALID, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)



*Create a SPI object and Initialize with parameters.*

- virtual uint8\_t **Init** (uint32\_t baudRateInBps=2000000, uint8\_t transferSizeInBits=8, **PinIO** gpioChip↔ Select=PinDef\_INVALID, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)

*Initialize an existing SPI object.*

- virtual bool **RegisterSem** (OS\_SEM \*finishedSem)

*Register a semaphore for the SPI module.*

- virtual bool **ClrSem** ()

*Clear a semaphore registration.*

- virtual OS\_SEM \* **GetSem** ()

*Obtain a pointer to the SPI finished semaphore.*

- virtual bool **Done** ()

*Function to check SPI status.*

- virtual uint32\_t **GetActualBaudrate** ()

*Returns the active baud rate.*

### Additional Inherited Members

### Static Public Member Functions inherited from **SPIModule**

- static BOOL **Done** (uint8\_t **SPIModule**)

*Static function to check SPI status.*

- static BOOL **Done** (uint8\_t **SPIModule**)

*Static function to check SPI status.*

### 16.165.1 Detailed Description

The Single-Bit SPI mode QSPI Peripheral Class.

Class definition for SPI module objects

### 16.165.2 Constructor & Destructor Documentation

#### 16.165.2.1 **SPI\_QSPI()** [1/2]

```
SPI_QSPI::SPI_QSPI (
 uint8_t QuadSPIModule = DEFAULT_QUADSPI_MODULE)
```

Create a SPI object.

This constructor will create the object, but will *not* initialize the SPI module. You must call the **Init()** member function to initialize.

#### Parameters

|                      |                                                                 |
|----------------------|-----------------------------------------------------------------|
| <b>QuadSPIModule</b> | QSPI module number (0). The SAME70 only has one QuadSPI module. |
|----------------------|-----------------------------------------------------------------|

#### 16.165.2.2 **SPI\_QSPI()** [2/2]

```
SPI_QSPI::SPI_QSPI (
 uint8_t QuadSPIModule,
 uint32_t baudRateInBps,
 uint8_t transferSizeInBits = 8,
 uint8_t peripheralChipSelects = 0x00,
 uint8_t chipSelectPolarity = 0x0F,
 uint8_t clockPolarity = 0,
 uint8_t clockPhase = 1,
 BOOL doutHiz = TRUE,
```

```
uint8_t csToClockDelay = 0,
uint8_t delayAfterTransfer = 0)
```

Create a SPI object and Initialize with parameters.

Creates a SPI object and initializes the SPI module. You do not need to call the [Init\(\)](#) member function.

- If configured for 8 bits per transfer then the data must be uint8\_t aligned
- If configured for > than 8 bits per transfer then the data must be uint16\_t aligned

#### Parameters

|                              |                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>QuadSPIModule</i>         | QSPI module number (0). The SAME70 has only one QuadSPI module                                                              |
| <i>baudRateInBps</i>         | Maximum baud rate requested                                                                                                 |
| <i>transferSizeInBits</i>    | Number of bits per transfer: 8-bit up to 16-bit transfer sizes are valid                                                    |
| <i>peripheralChipSelects</i> | SPI chip selects to use for transfer. Only one chip select is available to the QuadSPI peripheral on pin P2[48].            |
| <i>chipSelectPolarity</i>    | Parameter not supported on the MODM7AE70. Chip select asserts with logic level low. <a href="#">spiChipSelectPolarity</a>   |
| <i>clockPolarity</i>         | 0 = inactive logic level low, 1 = high                                                                                      |
| <i>clockPhase</i>            | 0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge. |
| <i>doutHiz</i>               | Data output high impedance between transfers                                                                                |
| <i>csToClockDelay</i>        | Delay from chip select to valid clock (default is 0). When 0, the delay is half the clock period.                           |
| <i>delayAfterTransfer</i>    | When 0, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the character transfers. |

## 16.165.3 Member Function Documentation

### 16.165.3.1 Init()

```
virtual uint8_t SPI_QSPI::Init (
 uint32_t baudRateInBps = 2000000,
 uint8_t transferSizeInBits = 8,
 uint8_t peripheralChipSelects = 0x00,
 uint8_t chipSelectPolarity = 0x0F,
 uint8_t clockPolarity = 0,
 uint8_t clockPhase = 1,
 BOOL doutHiz = TRUE,
 uint8_t csToClockDelay = 0,
 uint8_t delayAfterTransfer = 0) [override], [virtual]
```

Initialize an existing SPI object.

- If configured for 8 bits per transfer then the data must be uint8\_t aligned
- If configured for > than 8 bits per transfer then the data must be uint16\_t aligned

#### Parameters

|                              |                                                                                                                           |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>baudRateInBps</i>         | Maximum baud rate requested                                                                                               |
| <i>transferSizeInBits</i>    | Number of bits per transfer: 8-bit up to 16-bit transfer sizes are valid                                                  |
| <i>peripheralChipSelects</i> | SPI chip selects to use for transfer. Only one chip select is available to the QuadSPI peripheral on pin P2[48].          |
| <i>chipSelectPolarity</i>    | Parameter not supported on the MODM7AE70. Chip select asserts with logic level low. <a href="#">spiChipSelectPolarity</a> |

## Parameters

|                           |                                                                                                                             |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>clockPolarity</i>      | 0 = inactive logic level low, 1 = high                                                                                      |
| <i>clockPhase</i>         | 0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge. |
| <i>doutHiz</i>            | Data output high impedance between transfers                                                                                |
| <i>csToClockDelay</i>     | Delay from chip select to valid clock (default is 0). When 0, the delay is half the clock period.                           |
| <i>delayAfterTransfer</i> | When 0, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the character transfers. |

## Returns

The current state of the SPI bus [dspState](#)

Reimplemented from [SPIModule](#).

**16.165.3.2 Rx()**

```
uint8_t SPI_QSPI::Rx (
 uint8_t * receiveBufferPtr,
 uint32_t byteCount,
 int csReturnToInactive = DEASSERT_AFTER_LAST) [inline], [virtual]
```

Convenience function for unidirectional receive.

## Parameters

|                           |                                                  |
|---------------------------|--------------------------------------------------|
| <i>receiveBufferPtr</i>   | Pointer to the buffer to write the received data |
| <i>byteCount</i>          | Number of bytes to transmit                      |
| <i>csReturnToInactive</i> | Chip select state                                |

## Returns

The current state of the SPI bus [dspState](#)

Reimplemented from [SPIModule](#).

**16.165.3.3 SetBusSpeed()**

```
virtual uint32_t SPI_QSPI::SetBusSpeed (
 uint32_t maxSpeed) [override], [virtual]
```

Set the SPI bus speed Will attempt to set the desired bus speed. It may be different based on the available system characteristics.

## Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>maxSpeed</i> | The maximum desired bus speed |
|-----------------|-------------------------------|

## Returns

The actual bus speed

Reimplemented from [SPIModule](#).

**16.165.3.4 SetCS()**

```
virtual bool SPI_QSPI::SetCS (
 uint8_t CS) [inline], [override], [virtual]
```

Set the chip select configuration for the SPI object's bus transactions.

Single-bit SPI mode for the QuadSPI peripheral only supports one chip select. This function is defined for compatibility between SPI drivers.

## Parameters

|    |                                              |
|----|----------------------------------------------|
| CS | The QuadSPI module has only one chip select. |
|----|----------------------------------------------|

## Returns

true if successful, false if SPI is currently active

Reimplemented from [SPIModule](#).

**16.165.3.5 Start()**

```
virtual uint8_t SPI_QSPI::Start (
 uint8_t * transmitBufferPtr,
 volatile uint8_t * receiveBufferPtr,
 uint32_t byteCount,
 int csReturnToInactive = DEASSERT_AFTER_LAST) [override], [virtual]
```

Start a SPI transfer.

- If configured for 8 bits per transfer then the data must be uint8\_t aligned
- If configured for > than 8 bits per transfer then the data must be uint16\_t aligned
- If either RX or TX pointer is assigned 'null' then that communication direction will not occur.

## Parameters

|                           |                                                       |
|---------------------------|-------------------------------------------------------|
| <i>transmitBufferPtr</i>  | Pointer to the buffer containing the data to transmit |
| <i>receiveBufferPtr</i>   | Pointer to the buffer to store the received data      |
| <i>byteCount</i>          | Number of bytes to transmit                           |
| <i>csReturnToInactive</i> | Chip select state <a href="#">dspIChipSelectMode</a>  |

## Returns

The current state of the SPI bus [dspIState](#)

Reimplemented from [SPIModule](#).

**16.165.3.6 Tx()**

```
virtual uint8_t SPI_QSPI::Tx (
 uint8_t * transmitBufferPtr,
 uint32_t byteCount,
 int csReturnToInactive = DEASSERT_AFTER_LAST) [inline], [override], [virtual]
```

Convenience function for unidirectional transmit.

## Parameters

|                           |                                                       |
|---------------------------|-------------------------------------------------------|
| <i>transmitBufferPtr</i>  | Pointer to the buffer containing the data to transmit |
| <i>byteCount</i>          | Number of bytes to transmit                           |
| <i>csReturnToInactive</i> | Chip select state                                     |

## Returns

The current state of the SPI bus [dspIState](#)

Reimplemented from [SPIModule](#).

The documentation for this class was generated from the following file:

- quadspi.h

## 16.166 SPI\_USART Class Reference

USART in SPI mode Peripheral Module Class.

#include <usart.h>

Inherits [SPIModule](#).

### Public Member Functions

- [SPI\\_USART](#) (uint8\_t USARTModule)  
*Create a SPI object.*
- [SPI\\_USART](#) (uint8\_t USARTModule, uint32\_t baudRateInBps, uint8\_t transferSizeInBits=8, uint8\_t peripheralChipSelects=0x00, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)  
*Create a SPI object and Initialize with parameters.*
- virtual uint8\_t [Init](#) (uint32\_t baudRateInBps=2000000, uint8\_t transferSizeInBits=8, uint8\_t peripheralChipSelects=0x00, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0) override  
*Initialize an existing SPI object.*
- virtual uint32\_t [SetBusSpeed](#) (uint32\_t maxSpeed) override  
*Set the SPI bus speed Will attempt to set the desired bus speed. It may be different based on the available system characteristics.*
- virtual uint8\_t [Start](#) (uint8\_t \*transmitBufferPtr, volatile uint8\_t \*receiveBufferPtr, uint32\_t byteCount, int csReturnToInactive=[DEASSERT\\_AFTER\\_LAST](#)) override  
*Start a SPI transfer.*
- virtual uint8\_t [Tx](#) (uint8\_t \*transmitBufferPtr, uint32\_t byteCount, int csReturnToInactive=[DEASSERT\\_AFTER\\_LAST](#)) override  
*Convenience function for unidirectional transmit.*
- uint8\_t [Rx](#) (uint8\_t \*receiveBufferPtr, uint32\_t byteCount, int csReturnToInactive=[DEASSERT\\_AFTER\\_LAST](#))  
*Convenience function for unidirectional receive.*
- virtual bool [SetCS](#) (uint8\_t CS) override  
*Set the chip select configuration for the SPI object's bus transactions.*

### Public Member Functions inherited from [SPIModule](#)

- [SPIModule](#) (uint8\_t SPIModule)  
*Create a SPI object.*
- [SPIModule](#) (uint8\_t SPIModule, uint32\_t baudRateInBps, uint8\_t transferSizeInBits=8, uint8\_t peripheralChipSelects=0x00, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0, uint8\_t delayBetweenTransfer=0)  
*Create a SPI object and Initialize with parameters.*
- virtual bool [RegisterSem](#) (OS\_SEM \*finishedSem)  
*Register a semaphore for the SPI module.*
- virtual bool [ClrSem](#) ()  
*Clear a semaphore registration.*
- virtual OS\_SEM \* [GetSem](#) ()  
*Obtain a pointer to the SPI finished semaphore.*
- virtual bool [Done](#) ()  
*Function to check SPI status.*
- virtual uint32\_t [GetActualBaudrate](#) ()  
*Returns the active baud rate.*
- [SPIModule](#) (uint8\_t SPIModule)  
*Create a SPI object.*

- [SPIModule](#) (uint8\_t [SPIModule](#), uint32\_t baudRateInBps, uint8\_t transferSizeInBits=8, uint8\_t peripheral↔ ChipSelects=0x00, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)

*Create a SPI object and Initialize with parameters.*

- [SPIModule](#) (uint8\_t [SPIModule](#), uint32\_t baudRateInBps, uint8\_t transferSizeInBits=8, [PinIO](#) gpioChip↔ Select=PinDef\_INVALID, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)

*Create a SPI object and Initialize with parameters.*

- virtual uint8\_t [Init](#) (uint32\_t baudRateInBps=2000000, uint8\_t transferSizeInBits=8, [PinIO](#) gpioChip↔ Select=PinDef\_INVALID, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)

*Initialize an existing SPI object.*

- virtual bool [RegisterSem](#) (OS\_SEM \*finishedSem)

*Register a semaphore for the SPI module.*

- virtual bool [ClrSem](#) ()

*Clear a semaphore registration.*

- virtual OS\_SEM \* [GetSem](#) ()

*Obtain a pointer to the SPI finished semaphore.*

- virtual bool [Done](#) ()

*Function to check SPI status.*

- virtual uint32\_t [GetActualBaudrate](#) ()

*Returns the active baud rate.*

## Additional Inherited Members

## Static Public Member Functions inherited from [SPIModule](#)

- static BOOL [Done](#) (uint8\_t [SPIModule](#))

*Static function to check SPI status.*

- static BOOL [Done](#) (uint8\_t [SPIModule](#))

*Static function to check SPI status.*

### 16.166.1 Detailed Description

USART in SPI mode Peripheral Module Class.

Class definition for SPI module objects

### 16.166.2 Constructor & Destructor Documentation

#### 16.166.2.1 [SPI\\_USART\(\)](#) [1/2]

```
SPI_USART::SPI_USART (
 uint8_t USARTModule)
```

Create a SPI object.

This constructor will create the object, but will *not* initialize the SPI module. You must call the [Init\(\)](#) member function to initialize.

#### Parameters

|                    |                            |
|--------------------|----------------------------|
| <i>USARTModule</i> | SPI module number (0 or 1) |
|--------------------|----------------------------|

### 16.166.2.2 SPI\_USART() [2/2]

```
SPI_USART::SPI_USART (
 uint8_t USARTModule,
 uint32_t baudRateInBps,
 uint8_t transferSizeInBits = 8,
 uint8_t peripheralChipSelects = 0x00,
 uint8_t chipSelectPolarity = 0x0F,
 uint8_t clockPolarity = 0,
 uint8_t clockPhase = 1,
 BOOL doutHiz = TRUE,
 uint8_t csToClockDelay = 0,
 uint8_t delayAfterTransfer = 0)
```

Create a SPI object and Initialize with parameters.

Creates a SPI object and initializes the SPI module. You do not need to call the [Init\(\)](#) member function.

#### Parameters

|                              |                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>USARTModule</i>           | USART module number (0 or 1)                                                                                                |
| <i>baudRateInBps</i>         | Maximum baud rate requested                                                                                                 |
| <i>transferSizeInBits</i>    | Number of bits per transfer: 8, 16 or 32                                                                                    |
| <i>peripheralChipSelects</i> | SPI chip selects to use for transfer                                                                                        |
| <i>chipSelectPolarity</i>    | 0 = inactive logic level low, 1 = high                                                                                      |
| <i>clockPolarity</i>         | 0 = inactive logic level low, 1 = high                                                                                      |
| <i>clockPhase</i>            | 0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge. |
| <i>doutHiz</i>               | Data output high impedance between transfers                                                                                |
| <i>csToClockDelay</i>        | This option is not supported by the USART peripheral in SPI mode.                                                           |
| <i>delayAfterTransfer</i>    | This option is not supported by the USART peripheral in SPI mode.                                                           |

## 16.166.3 Member Function Documentation

### 16.166.3.1 Init()

```
virtual uint8_t SPI_USART::Init (
 uint32_t baudRateInBps = 2000000,
 uint8_t transferSizeInBits = 8,
 uint8_t peripheralChipSelects = 0x00,
 uint8_t chipSelectPolarity = 0x0F,
 uint8_t clockPolarity = 0,
 uint8_t clockPhase = 1,
 BOOL doutHiz = TRUE,
 uint8_t csToClockDelay = 0,
 uint8_t delayAfterTransfer = 0) [override], [virtual]
```

Initialize an existing SPI object.

#### Parameters

|                              |                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>baudRateInBps</i>         | Maximum baud rate requested                                                                                                 |
| <i>transferSizeInBits</i>    | Number of bits per transfer: 8, 16 or 32                                                                                    |
| <i>peripheralChipSelects</i> | SPI chip selects to use for transfer                                                                                        |
| <i>chipSelectPolarity</i>    | 0 = inactive logic level low, 1 = high                                                                                      |
| <i>clockPolarity</i>         | 0 = inactive logic level low, 1 = high                                                                                      |
| <i>clockPhase</i>            | 0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge. |

## Parameters

|                           |                                                                   |
|---------------------------|-------------------------------------------------------------------|
| <i>doutHiz</i>            | Data output high impedance between transfers                      |
| <i>csToClockDelay</i>     | This option is not supported by the USART peripheral in SPI mode. |
| <i>delayAfterTransfer</i> | This option is not supported by the USART peripheral in SPI mode. |

## Returns

The current state of the SPI bus [dsSpiState](#)

Reimplemented from [SPIModule](#).

**16.166.3.2 Rx()**

```
uint8_t SPI_USART::Rx (
 uint8_t * receiveBufferPtr,
 uint32_t byteCount,
 int csReturnToInactive = DEASSERT_AFTER_LAST) [inline], [virtual]
```

Convenience function for unidirectional receive.

## Parameters

|                           |                                                  |
|---------------------------|--------------------------------------------------|
| <i>receiveBufferPtr</i>   | Pointer to the buffer to write the received data |
| <i>byteCount</i>          | Number of bytes to transmit                      |
| <i>csReturnToInactive</i> | Chip select state                                |

## Returns

The current state of the SPI bus [dsSpiState](#)

Reimplemented from [SPIModule](#).

**16.166.3.3 SetBusSpeed()**

```
virtual uint32_t SPI_USART::SetBusSpeed (
 uint32_t maxSpeed) [override], [virtual]
```

Set the SPI bus speed Will attempt to set the desired bus speed. It may be different based on the available system characteristics.

## Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>maxSpeed</i> | The maximum desired bus speed |
|-----------------|-------------------------------|

## Returns

The actual bus speed

Reimplemented from [SPIModule](#).

**16.166.3.4 SetCS()**

```
virtual bool SPI_USART::SetCS (
 uint8_t CS) [inline], [override], [virtual]
```

Set the chip select configuration for the SPI object's bus transactions.

## Parameters

|           |                            |
|-----------|----------------------------|
| <i>CS</i> | Chip select to set, 0 - 3. |
|-----------|----------------------------|



**Returns**

true if successful, false if SPI is currently active

Reimplemented from [SPIModule](#).

**16.166.3.5 Start()**

```
virtual uint8_t SPI_USART::Start (
 uint8_t * transmitBufferPtr,
 volatile uint8_t * receiveBufferPtr,
 uint32_t byteCount,
 int csReturnToInactive = DEASSERT_AFTER_LAST) [override], [virtual]
```

Start a SPI transfer.

- If configured for 8 bits per transfer then the data must be uint8\_t aligned
- If configured for > than 8 bits per transfer then the data must be uint16\_t aligned
- If configured for > than 16 bits per transfer then the data must be uint32\_t aligned
- If either RX or TX pointer is assigned 'null' then that communication direction will not occur.

**Parameters**

|                           |                                                       |
|---------------------------|-------------------------------------------------------|
| <i>transmitBufferPtr</i>  | Pointer to the buffer containing the data to transmit |
| <i>receiveBufferPtr</i>   | Pointer to the buffer to store the received data      |
| <i>byteCount</i>          | Number of bytes to transmit                           |
| <i>csReturnToInactive</i> | Chip select state                                     |

**Returns**

The current state of the SPI bus [dspState](#)

Reimplemented from [SPIModule](#).

**16.166.3.6 Tx()**

```
virtual uint8_t SPI_USART::Tx (
 uint8_t * transmitBufferPtr,
 uint32_t byteCount,
 int csReturnToInactive = DEASSERT_AFTER_LAST) [inline], [override], [virtual]
```

Convenience function for unidirectional transmit.

**Parameters**

|                           |                                                       |
|---------------------------|-------------------------------------------------------|
| <i>transmitBufferPtr</i>  | Pointer to the buffer containing the data to transmit |
| <i>byteCount</i>          | Number of bytes to transmit                           |
| <i>csReturnToInactive</i> | Chip select state                                     |

**Returns**

The current state of the SPI bus [dspState](#)

Reimplemented from [SPIModule](#).

The documentation for this class was generated from the following file:

- usart.h

## 16.167 spiDriverStruct Struct Reference

Variables/configurations for a DSPI transfer.

```
#include <dspi.h>
```

### Public Attributes

- [csReturnType](#) **csReturnToInactive**  
*The chip select assertion configuration.*
- volatile uint8\_t **SPI\_INT\_STATUS**  
*Status of the spi device.*
- volatile BOOL **SPIfinished**
  
- volatile uint8\_t \* **pRxBuf**  
*Pointer to memory where data will be read from the peripheral.*
- volatile uint8\_t \* **pTxBuf**  
*Pointer to memory where data will be written to the peripheral.*
- uint8\_t **BitsPerQueue**  
*The number of bits per transfer, (value = 8 - 32)*
- bool **isByteSwap**
  
- uint32\_t **SPI\_SizeLeft**  
*The number of bytes left in the transfer.*
- uint16\_t **Command\_Mask**  
*A partial configuration for the queue's command reg.*
- OS\_SEM \* **SPI\_Sem**  
*A pointer to an external semaphore provided by [DSPIStart\(\)](#)*
- uint32\_t **WordsToWrite**  
*Number of words to write to the command/TX Queues and Read from RX.*
- uint32\_t **LastWordsToWrite**  
*The amount from the previous ISR.*
- volatile [dspiDMAStruct](#) **dma**

### 16.167.1 Detailed Description

Variables/configurations for a DSPI transfer.

The documentation for this struct was generated from the following files:

- cortex-m7/cpu/SAME70/include/dspi.h
- cortex-m7/cpu/MIMXRT10xx/include/dspi.h

## 16.168 SPIModule Class Reference

SPI Peripheral Module Class.

```
#include <dspi.h>
```

Inherited by [SPI\\_QSPI](#), and [SPI\\_USART](#).

## Public Member Functions

- [SPIModule](#) (uint8\_t [SPIModule](#))  
*Create a SPI object.*
- [SPIModule](#) (uint8\_t [SPIModule](#), uint32\_t baudRateInBps, uint8\_t transferSizeInBits=8, uint8\_t peripheral↵ ChipSelects=0x00, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0, uint8\_t delayBetweenTransfer=0)  
*Create a SPI object and Initialize with parameters.*
- virtual uint8\_t [Init](#) (uint32\_t baudRateInBps=2000000, uint8\_t transferSizeInBits=8, uint8\_t peripheralChip↵ Selects=0x00, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL dout↵ Hiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)  
*Initialize an existing SPI object.*
- virtual uint32\_t [SetBusSpeed](#) (uint32\_t maxSpeed)  
*Set the SPI bus speed Will attempt to set the desired bus speed. It may be different based on the available system characteristics.*
- virtual uint8\_t [Start](#) (uint8\_t \*transmitBufferPtr, volatile uint8\_t \*receiveBufferPtr, uint32\_t byteCount, int cs↵ ReturnToInactive=[DEASSERT\\_AFTER\\_LAST](#))  
*Start a SPI transfer.*
- virtual uint8\_t [Tx](#) (uint8\_t \*transmitBufferPtr, uint32\_t byteCount, int csReturnToInactive=[DEASSERT\\_AFTER\\_LAST](#))  
*Convenience function for unidirectional transmit.*
- virtual uint8\_t [Rx](#) (uint8\_t \*receiveBufferPtr, uint32\_t byteCount, int csReturnToInactive=[DEASSERT\\_AFTER\\_LAST](#))  
*Convenience function for unidirectional receive.*
- virtual bool [RegisterSem](#) (OS\_SEM \*finishedSem)  
*Register a semaphore for the SPI module.*
- virtual bool [ClrSem](#) ()  
*Clear a semaphore registration.*
- virtual OS\_SEM \* [GetSem](#) ()  
*Obtain a pointer to the SPI finished semaphore.*
- virtual bool [Done](#) ()  
*Function to check SPI status.*
- virtual uint32\_t [GetActualBaudrate](#) ()  
*Returns the active baud rate.*
- virtual bool [SetCS](#) (uint8\_t CS)  
*Set the chip select configuration for the SPI object's bus transactions.*
- [SPIModule](#) (uint8\_t [SPIModule](#))  
*Create a SPI object.*
- [SPIModule](#) (uint8\_t [SPIModule](#), uint32\_t baudRateInBps, uint8\_t transferSizeInBits=8, uint8\_t peripheral↵ ChipSelects=0x00, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)  
*Create a SPI object and Initialize with parameters.*
- [SPIModule](#) (uint8\_t [SPIModule](#), uint32\_t baudRateInBps, uint8\_t transferSizeInBits=8, [PinIO](#) gpioChip↵ Select=PinDef\_INVALID, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)  
*Create a SPI object and Initialize with parameters.*
- virtual uint8\_t [Init](#) (uint32\_t baudRateInBps=2000000, uint8\_t transferSizeInBits=8, uint8\_t peripheralChip↵ Selects=0x00, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL dout↵ Hiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)  
*Initialize an existing SPI object.*
- virtual uint8\_t [Init](#) (uint32\_t baudRateInBps=2000000, uint8\_t transferSizeInBits=8, [PinIO](#) gpioChip↵ Select=PinDef\_INVALID, uint8\_t chipSelectPolarity=0x0F, uint8\_t clockPolarity=0, uint8\_t clockPhase=1, BOOL doutHiz=TRUE, uint8\_t csToClockDelay=0, uint8\_t delayAfterTransfer=0)  
*Initialize an existing SPI object.*
- virtual uint32\_t [SetBusSpeed](#) (uint32\_t maxSpeed)

*Set the SPI bus speed Will attempt to set the desired bus speed. It may be different based on the available system characteristics.*

- virtual uint8\_t [Start](#) (uint8\_t \*transmitBufferPtr, volatile uint8\_t \*receiveBufferPtr, uint32\_t byteCount, int cs↵ ReturnToInactive=[DEASSERT\\_AFTER\\_LAST](#))

*Start a SPI transfer.*

- virtual uint8\_t [Tx](#) (uint8\_t \*transmitBufferPtr, uint32\_t byteCount, int csReturnToInactive=[DEASSERT\\_AFTER\\_LAST](#))

*Convenience function for unidirectional transmit.*

- virtual uint8\_t [Rx](#) (uint8\_t \*receiveBufferPtr, uint32\_t byteCount, int csReturnToInactive=[DEASSERT\\_AFTER\\_LAST](#))

*Convenience function for unidirectional receive.*

- virtual bool [RegisterSem](#) (OS\_SEM \*finishedSem)

*Register a semaphore for the SPI module.*

- virtual bool [ClrSem](#) ()

*Clear a semaphore registration.*

- virtual OS\_SEM \* [GetSem](#) ()

*Obtain a pointer to the SPI finished semaphore.*

- virtual bool [Done](#) ()

*Function to check SPI status.*

- virtual uint32\_t [GetActualBaudrate](#) ()

*Returns the active baud rate.*

- virtual bool [SetCS](#) (uint8\_t CS)

*Set the chip select configuration for the SPI object's bus transactions.*

### Static Public Member Functions

- static BOOL [Done](#) (uint8\_t [SPIModule](#))

*Static function to check SPI status.*

- static BOOL [Done](#) (uint8\_t [SPIModule](#))

*Static function to check SPI status.*

## 16.168.1 Detailed Description

SPI Peripheral Module Class.

Class definition for SPI module objects

## 16.168.2 Constructor & Destructor Documentation

### 16.168.2.1 SPIModule() [1/5]

```
SPIModule::SPIModule (
 uint8_t SPIModule)
```

Create a SPI object.

This constructor will create the object, but will *not* initialize the SPI module. You must call the [Init\(\)](#) member function to initialize.

#### Parameters

|                           |                            |
|---------------------------|----------------------------|
| <a href="#">SPIModule</a> | SPI module number (0 or 1) |
|---------------------------|----------------------------|

**16.168.2.2 SPIModule() [2/5]**

```

SPIModule::SPIModule (
 uint8_t SPIModule,
 uint32_t baudRateInBps,
 uint8_t transferSizeInBits = 8,
 uint8_t peripheralChipSelects = 0x00,
 uint8_t chipSelectPolarity = 0x0F,
 uint8_t clockPolarity = 0,
 uint8_t clockPhase = 1,
 BOOL doutHiz = TRUE,
 uint8_t csToClockDelay = 0,
 uint8_t delayAfterTransfer = 0,
 uint8_t delayBetweenTransfer = 0)

```

Create a SPI object and Initialize with parameters.

Creates a SPI object and initializes the SPI module. You do not need to call the [Init\(\)](#) member function.

**Parameters**

|                              |                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">SPIModule</a>    | SPI module number (0 or 1)                                                                                                  |
| <i>baudRateInBps</i>         | Maximum baud rate requested                                                                                                 |
| <i>transferSizeInBits</i>    | Number of bits per transfer: 8, 16 or 32                                                                                    |
| <i>peripheralChipSelects</i> | SPI chip selects to use for transfer <a href="#">spiChipSelect</a>                                                          |
| <i>chipSelectPolarity</i>    | Parameter not supported on the MODM7AE70. Chip select asserts with logic level low. <a href="#">spiChipSelectPolarity</a>   |
| <i>clockPolarity</i>         | 0 = inactive logic level low, 1 = high                                                                                      |
| <i>clockPhase</i>            | 0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge. |
| <i>doutHiz</i>               | Data output high impedance between transfers                                                                                |
| <i>csToClockDelay</i>        | Delay from chip select to valid clock (default is 0)                                                                        |
| <i>delayAfterTransfer</i>    | Chip select mode <a href="#">dsPICChipSelectMode</a>                                                                        |
| <i>delayBetweenTransfer</i>  | Min Delay between transfers                                                                                                 |

**16.168.2.3 SPIModule() [3/5]**

```

SPIModule::SPIModule (
 uint8_t SPIModule)

```

Create a SPI object.

This constructor will create the object, but will *not* initialize the SPI module. You must call the [Init\(\)](#) member function to initialize.

**Parameters**

|                           |                   |
|---------------------------|-------------------|
| <a href="#">SPIModule</a> | SPI module number |
|---------------------------|-------------------|

**16.168.2.4 SPIModule() [4/5]**

```

SPIModule::SPIModule (
 uint8_t SPIModule,
 uint32_t baudRateInBps,
 uint8_t transferSizeInBits = 8,
 uint8_t peripheralChipSelects = 0x00,
 uint8_t chipSelectPolarity = 0x0F,
 uint8_t clockPolarity = 0,
 uint8_t clockPhase = 1,

```

```

 BOOL doutHiz = TRUE,
 uint8_t csToClockDelay = 0,
 uint8_t delayAfterTransfer = 0)

```

Create a SPI object and Initialize with parameters.

Creates a SPI object and initializes the SPI module. You do not need to call the [Init\(\)](#) member function.

#### Parameters

|                              |                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">SPIModule</a>    | SPI module number (0 or 1)                                                                                                  |
| <i>baudRateInBps</i>         | Maximum baud rate requested                                                                                                 |
| <i>transferSizeInBits</i>    | Number of bits per transfer: 8, 16 or 32                                                                                    |
| <i>peripheralChipSelects</i> | SPI chip selects to use for transfer <a href="#">spiChipSelect</a>                                                          |
| <i>chipSelectPolarity</i>    | Parameter not supported on the SOMRT1061. Chip select asserts with logic level low. <a href="#">spiChipSelectPolarity</a>   |
| <i>clockPolarity</i>         | 0 = inactive logic level low, 1 = high                                                                                      |
| <i>clockPhase</i>            | 0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge. |
| <i>doutHiz</i>               | Data output high impedance between transfers                                                                                |
| <i>csToClockDelay</i>        | Delay from chip select to valid clock (default is 0)                                                                        |
| <i>delayAfterTransfer</i>    | Chip select mode <a href="#">dspChipSelectMode</a>                                                                          |

#### 16.168.2.5 SPIModule() [5/5]

```

SPIModule::SPIModule (
 uint8_t SPIModule,
 uint32_t baudRateInBps,
 uint8_t transferSizeInBits = 8,
 PinIO gpioChipSelect = PinDef_INVALID,
 uint8_t chipSelectPolarity = 0x0F,
 uint8_t clockPolarity = 0,
 uint8_t clockPhase = 1,
 BOOL doutHiz = TRUE,
 uint8_t csToClockDelay = 0,
 uint8_t delayAfterTransfer = 0)

```

Create a SPI object and Initialize with parameters.

Creates a SPI object and initializes the SPI module. You do not need to call the [Init\(\)](#) member function.

#### Parameters

|                           |                                                                                                                                                                                            |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">SPIModule</a> | SPI module number (0 or 1)                                                                                                                                                                 |
| <i>baudRateInBps</i>      | Maximum baud rate requested                                                                                                                                                                |
| <i>transferSizeInBits</i> | Number of bits per transfer: 8, 16 or 32                                                                                                                                                   |
| <i>gpioChipSelect</i>     | <a href="#">PinIO</a> to be configured as GPIO and used as chip select. Only supports <a href="#">spiDriverStruct::csReturnToInactive</a> values of DEASSERT_AFTER_LAST and DEASSERT_NEVER |
| <i>chipSelectPolarity</i> | Parameter not supported on the SOMRT1061. Chip select asserts with logic level low. <a href="#">spiChipSelectPolarity</a>                                                                  |
| <i>clockPolarity</i>      | 0 = inactive logic level low, 1 = high                                                                                                                                                     |
| <i>clockPhase</i>         | 0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge.                                                                |
| <i>doutHiz</i>            | Data output high impedance between transfers                                                                                                                                               |
| <i>csToClockDelay</i>     | Delay from chip select to valid clock (default is 0)                                                                                                                                       |
| <i>delayAfterTransfer</i> | Chip select mode <a href="#">dspChipSelectMode</a>                                                                                                                                         |

### 16.168.3 Member Function Documentation

#### 16.168.3.1 ClrSem() [1/2]

```
virtual bool SPIModule::ClrSem () [inline], [virtual]
```

Clear a semaphore registration.

##### Returns

true if the clear was successful, false if a SPI transaction is in progress

#### 16.168.3.2 ClrSem() [2/2]

```
virtual bool SPIModule::ClrSem () [inline], [virtual]
```

Clear a semaphore registration.

##### Returns

true if the clear was successful, false if a SPI transaction is in progress

#### 16.168.3.3 Done() [1/4]

```
virtual bool SPIModule::Done () [inline], [virtual]
```

Function to check SPI status.

Called as a class method on a specific SPI object. For example: MySpi.Done()

##### Returns

true if SPI is finished, false if active

#### 16.168.3.4 Done() [2/4]

```
virtual bool SPIModule::Done () [inline], [virtual]
```

Function to check SPI status.

Called as a class method on a specific SPI object. For example: MySpi.Done()

##### Returns

true if SPI is finished, false if active

#### 16.168.3.5 Done() [3/4]

```
static BOOL SPIModule::Done (
 uint8_t SPIModule) [static]
```

Static function to check SPI status.

Is called as a class wide static method. For example: SPIModule::Done(0)

##### Parameters

|                  |                                    |
|------------------|------------------------------------|
| <i>SPIModule</i> | The SPI module to check, 0, 1 or 2 |
|------------------|------------------------------------|

##### Returns

true if SPI is finished, false if active

#### 16.168.3.6 Done() [4/4]

```
static BOOL SPIModule::Done (
 uint8_t SPIModule) [static]
```

Static function to check SPI status.

Is called as a class wide static method. For example: SPIModule::Done(0)

## Parameters

|                                  |                                    |
|----------------------------------|------------------------------------|
| <a href="#"><i>SPIModule</i></a> | The SPI module to check, 0, 1 or 2 |
|----------------------------------|------------------------------------|

## Returns

true if SPI is finished, false if active

**16.168.3.7 GetActualBaudrate()** [1/2]

```
virtual uint32_t SPIModule::GetActualBaudrate () [inline], [virtual]
```

Returns the active baud rate.

The baud rate will be set to the value specified when the SPI module is initialized. If that value is not possible, the next lowest baud rate will be chosen.

## Returns

The actual SPI module baud rate

**16.168.3.8 GetActualBaudrate()** [2/2]

```
virtual uint32_t SPIModule::GetActualBaudrate () [inline], [virtual]
```

Returns the active baud rate.

The baud rate will be set to the value specified when the SPI module is initialized. If that value is not possible, the next lowest baud rate will be chosen.

## Returns

The actual SPI module baud rate

**16.168.3.9 GetSem()** [1/2]

```
virtual OS_SEM * SPIModule::GetSem () [inline], [virtual]
```

Obtain a pointer to the SPI finished semaphore.

## Returns

A pointer to the semaphore

**16.168.3.10 GetSem()** [2/2]

```
virtual OS_SEM * SPIModule::GetSem () [inline], [virtual]
```

Obtain a pointer to the SPI finished semaphore.

## Returns

A pointer to the semaphore

**16.168.3.11 Init()** [1/3]

```
virtual uint8_t SPIModule::Init (
 uint32_t baudRateInBps = 2000000,
 uint8_t transferSizeInBits = 8,
 PinIO gpioChipSelect = PinDef_INVALID,
 uint8_t chipSelectPolarity = 0x0F,
 uint8_t clockPolarity = 0,
 uint8_t clockPhase = 1,
 BOOL doutHiz = TRUE,
 uint8_t csToClockDelay = 0,
 uint8_t delayAfterTransfer = 0) [virtual]
```

Initialize an existing SPI object.



## Parameters

|                           |                                                                                                                                                                                            |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baudRateInBps</i>      | Maximum baud rate requested                                                                                                                                                                |
| <i>transferSizeInBits</i> | Number of bits per transfer: 8, 16 or 32                                                                                                                                                   |
| <i>gpioChipSelect</i>     | <a href="#">PinIO</a> to be configured as GPIO and used as chip select. Only supports <a href="#">spiDriverStruct::csReturnToInactive</a> values of DEASSERT_AFTER_LAST and DEASSERT_NEVER |
| <i>chipSelectPolarity</i> | Parameter not supported on the SOMRT1061. Chip select asserts with logic level low. <a href="#">spiChipSelectPolarity</a>                                                                  |
| <i>clockPolarity</i>      | 0 = inactive logic level low, 1 = high                                                                                                                                                     |
| <i>clockPhase</i>         | 0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge.                                                                |
| <i>doutHiz</i>            | Data output high impedance between transfers                                                                                                                                               |
| <i>csToClockDelay</i>     | Delay from chip select to valid clock (default is 0)                                                                                                                                       |
| <i>delayAfterTransfer</i> | Chip select mode <a href="#">dspciChipSelectMode</a>                                                                                                                                       |

## Returns

The current state of the SPI bus [dspciState](#)

**16.168.3.12 Init() [2/3]**

```
virtual uint8_t SPIModule::Init (
 uint32_t baudRateInBps = 2000000,
 uint8_t transferSizeInBits = 8,
 uint8_t peripheralChipSelects = 0x00,
 uint8_t chipSelectPolarity = 0x0F,
 uint8_t clockPolarity = 0,
 uint8_t clockPhase = 1,
 BOOL doutHiz = TRUE,
 uint8_t csToClockDelay = 0,
 uint8_t delayAfterTransfer = 0) [virtual]
```

Initialize an existing SPI object.

## Parameters

|                              |                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>baudRateInBps</i>         | Maximum baud rate requested                                                                                                 |
| <i>transferSizeInBits</i>    | Number of bits per transfer: 8, 16 or 32                                                                                    |
| <i>peripheralChipSelects</i> | SPI chip selects to use for transfer <a href="#">spiChipSelect</a>                                                          |
| <i>chipSelectPolarity</i>    | Parameter not supported on the SOMRT1061. Chip select asserts with logic level low. <a href="#">spiChipSelectPolarity</a>   |
| <i>clockPolarity</i>         | 0 = inactive logic level low, 1 = high                                                                                      |
| <i>clockPhase</i>            | 0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge. |
| <i>doutHiz</i>               | Data output high impedance between transfers                                                                                |
| <i>csToClockDelay</i>        | Delay from chip select to valid clock (default is 0)                                                                        |
| <i>delayAfterTransfer</i>    | Chip select mode <a href="#">dspciChipSelectMode</a>                                                                        |

## Returns

The current state of the SPI bus [dspciState](#)

Reimplemented in [SPI\\_QSPI](#), and [SPI\\_USART](#).

**16.168.3.13 Init() [3/3]**

```
virtual uint8_t SPIModule::Init (
 uint32_t baudRateInBps = 2000000,
 uint8_t transferSizeInBits = 8,
 uint8_t peripheralChipSelects = 0x00,
 uint8_t chipSelectPolarity = 0x0F,
 uint8_t clockPolarity = 0,
 uint8_t clockPhase = 1,
 BOOL doutHiz = TRUE,
 uint8_t csToClockDelay = 0,
 uint8_t delayAfterTransfer = 0) [virtual]
```

Initialize an existing SPI object.

**Parameters**

|                              |                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>baudRateInBps</i>         | Maximum baud rate requested                                                                                                 |
| <i>transferSizeInBits</i>    | Number of bits per transfer: 8, 16 or 32                                                                                    |
| <i>peripheralChipSelects</i> | SPI chip selects to use for transfer <a href="#">spiChipSelect</a>                                                          |
| <i>chipSelectPolarity</i>    | Parameter not supported on the MODM7AE70. Chip select asserts with logic level low. <a href="#">spiChipSelectPolarity</a>   |
| <i>clockPolarity</i>         | 0 = inactive logic level low, 1 = high                                                                                      |
| <i>clockPhase</i>            | 0 = data captured leading edge clock, changed following edge. 1 = data changed leading edge clock, captured following edge. |
| <i>doutHiz</i>               | Data output high impedance between transfers                                                                                |
| <i>csToClockDelay</i>        | Delay from chip select to valid clock (default is 0)                                                                        |
| <i>delayAfterTransfer</i>    | Chip select mode <a href="#">dspChipSelectMode</a>                                                                          |

**Returns**

The current state of the SPI bus [dspState](#)

Reimplemented in [SPI\\_QSPI](#), and [SPI\\_USART](#).

**16.168.3.14 RegisterSem() [1/2]**

```
virtual bool SPIModule::RegisterSem (
 OS_SEM * finishedSem) [virtual]
```

Register a semaphore for the SPI module.

The SPI module will post to this semaphore when a transaction is complete

**Parameters**

|                    |                          |
|--------------------|--------------------------|
| <i>finishedSem</i> | Pointer to the semaphore |
|--------------------|--------------------------|

**Returns**

true if the registration was successful, false if a SPI transaction is in progress

**16.168.3.15 RegisterSem() [2/2]**

```
virtual bool SPIModule::RegisterSem (
 OS_SEM * finishedSem) [virtual]
```

Register a semaphore for the SPI module.

The SPI module will post to this semaphore when a transaction is complete

## Parameters

|                    |                          |
|--------------------|--------------------------|
| <i>finishedSem</i> | Pointer to the semaphore |
|--------------------|--------------------------|

## Returns

true if the registration was successful, false if a SPI transaction is in progress

**16.168.3.16 Rx()** [1/2]

```
virtual uint8_t SPIModule::Rx (
 uint8_t * receiveBufferPtr,
 uint32_t byteCount,
 int csReturnToInactive = DEASSERT_AFTER_LAST) [inline], [virtual]
```

Convenience function for unidirectional receive.

## Parameters

|                           |                                              |
|---------------------------|----------------------------------------------|
| <i>*receiveBufferPtr</i>  | Pointer to the buffer to store incoming data |
| <i>byteCount</i>          | Number of bytes to transmit                  |
| <i>csReturnToInactive</i> | Chip select state                            |

## Returns

The current state of the SPI bus [dspState](#)

Reimplemented in [SPI\\_QSPI](#), and [SPI\\_USART](#).

**16.168.3.17 Rx()** [2/2]

```
virtual uint8_t SPIModule::Rx (
 uint8_t * receiveBufferPtr,
 uint32_t byteCount,
 int csReturnToInactive = DEASSERT_AFTER_LAST) [inline], [virtual]
```

Convenience function for unidirectional receive.

## Parameters

|                           |                                              |
|---------------------------|----------------------------------------------|
| <i>*receiveBufferPtr</i>  | Pointer to the buffer to store incoming data |
| <i>byteCount</i>          | Number of bytes to transmit                  |
| <i>csReturnToInactive</i> | Chip select state                            |

## Returns

The current state of the SPI bus [dspState](#)

Reimplemented in [SPI\\_QSPI](#), and [SPI\\_USART](#).

**16.168.3.18 SetBusSpeed()** [1/2]

```
virtual uint32_t SPIModule::SetBusSpeed (
 uint32_t maxSpeed) [virtual]
```

Set the SPI bus speed Will attempt to set the desired bus speed. It may be different based on the available system characteristics.

## Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>maxSpeed</i> | The maximum desired bus speed |
|-----------------|-------------------------------|

**Returns**

The actual bus speed

Reimplemented in [SPI\\_QSPI](#), and [SPI\\_USART](#).

**16.168.3.19 SetBusSpeed() [2/2]**

```
virtual uint32_t SPIModule::SetBusSpeed (
 uint32_t maxSpeed) [virtual]
```

Set the SPI bus speed Will attempt to set the desired bus speed. It may be different based on the available system characteristics.

**Parameters**

|                 |                               |
|-----------------|-------------------------------|
| <i>maxSpeed</i> | The maximum desired bus speed |
|-----------------|-------------------------------|

**Returns**

The actual bus speed

Reimplemented in [SPI\\_QSPI](#), and [SPI\\_USART](#).

**16.168.3.20 SetCS() [1/2]**

```
virtual bool SPIModule::SetCS (
 uint8_t CS) [inline], [virtual]
```

Set the chip select configuration for the SPI object's bus transactions.  
If an invalid CS number is given, a default chip select of 0 is used.

**Parameters**

|           |                            |
|-----------|----------------------------|
| <i>CS</i> | Chip select to set, 0 - 3. |
|-----------|----------------------------|

**Returns**

true if successful, false if SPI is currently active

Reimplemented in [SPI\\_QSPI](#), and [SPI\\_USART](#).

**16.168.3.21 SetCS() [2/2]**

```
virtual bool SPIModule::SetCS (
 uint8_t CS) [inline], [virtual]
```

Set the chip select configuration for the SPI object's bus transactions.

**Parameters**

|           |                            |
|-----------|----------------------------|
| <i>CS</i> | Chip select to set, 0 - 3. |
|-----------|----------------------------|

**Returns**

true if successful, false if SPI is currently active

Reimplemented in [SPI\\_QSPI](#), and [SPI\\_USART](#).

**16.168.3.22 Start()** [1/2]

```
virtual uint8_t SPIModule::Start (
 uint8_t * transmitBufferPtr,
 volatile uint8_t * receiveBufferPtr,
 uint32_t byteCount,
 int csReturnToInactive = DEASSERT_AFTER_LAST) [virtual]
```

Start a SPI transfer.

- If configured for 8 bits per transfer then the data must be uint8\_t aligned
- If configured for > than 8 bits per transfer then the data must be uint16\_t aligned
- If configured for > than 16 bits per transfer then the data must be uint32\_t aligned
- If either RX or TX pointer is assigned 'null' then that communication direction will not occur.
- If DSPI\_Finished points to a semaphore, then the DSPI will POST to it when the transfer is complete.
- The semaphore is optional, but it can increase efficiency.

**Parameters**

|                           |                                                       |
|---------------------------|-------------------------------------------------------|
| <i>transmitBufferPtr</i>  | Pointer to the buffer containing the data to transmit |
| <i>receiveBufferPtr</i>   | Pointer to the buffer to store the received data      |
| <i>byteCount</i>          | Number of bytes to transmit                           |
| <i>csReturnToInactive</i> | Chip select state                                     |

**Returns**

The current state of the SPI bus [dspiState](#)

Reimplemented in [SPI\\_QSPI](#), and [SPI\\_USART](#).

**16.168.3.23 Start()** [2/2]

```
virtual uint8_t SPIModule::Start (
 uint8_t * transmitBufferPtr,
 volatile uint8_t * receiveBufferPtr,
 uint32_t byteCount,
 int csReturnToInactive = DEASSERT_AFTER_LAST) [virtual]
```

Start a SPI transfer.

- If configured for 8 bits per transfer then the data must be uint8\_t aligned
- If configured for > than 8 bits per transfer then the data must be uint16\_t aligned
- If configured for > than 16 bits per transfer then the data must be uint32\_t aligned
- If either RX or TX pointer is assigned 'null' then that communication direction will not occur.
- If DSPI\_Finished points to a semaphore, then the DSPI will POST to it when the transfer is complete.
- The semaphore is optional, but it can increase efficiency.

**Parameters**

|                           |                                                       |
|---------------------------|-------------------------------------------------------|
| <i>transmitBufferPtr</i>  | Pointer to the buffer containing the data to transmit |
| <i>receiveBufferPtr</i>   | Pointer to the buffer to store the received data      |
| <i>byteCount</i>          | Number of bytes to transmit                           |
| <i>csReturnToInactive</i> | Chip select state                                     |

**Returns**

The current state of the SPI bus [dspiState](#)

Reimplemented in [SPI\\_QSPI](#), and [SPI\\_USART](#).

**16.168.3.24 Tx() [1/2]**

```
virtual uint8_t SPIModule::Tx (
 uint8_t * transmitBufferPtr,
 uint32_t byteCount,
 int csReturnToInactive = DEASSERT_AFTER_LAST) [inline], [virtual]
```

Convenience function for unidirectional transmit.

**Parameters**

|                           |                                                       |
|---------------------------|-------------------------------------------------------|
| <i>*transmitBufferPtr</i> | Pointer to the buffer containing the data to transmit |
| <i>byteCount</i>          | Number of bytes to transmit                           |
| <i>csReturnToInactive</i> | Chip select state                                     |

**Returns**

The current state of the SPI bus [dspiState](#)

Reimplemented in [SPI\\_QSPI](#), and [SPI\\_USART](#).

**16.168.3.25 Tx() [2/2]**

```
virtual uint8_t SPIModule::Tx (
 uint8_t * transmitBufferPtr,
 uint32_t byteCount,
 int csReturnToInactive = DEASSERT_AFTER_LAST) [inline], [virtual]
```

Convenience function for unidirectional transmit.

**Parameters**

|                           |                                                       |
|---------------------------|-------------------------------------------------------|
| <i>*transmitBufferPtr</i> | Pointer to the buffer containing the data to transmit |
| <i>byteCount</i>          | Number of bytes to transmit                           |
| <i>csReturnToInactive</i> | Chip select state                                     |

**Returns**

The current state of the SPI bus [dspiState](#)

Reimplemented in [SPI\\_QSPI](#), and [SPI\\_USART](#).

The documentation for this class was generated from the following files:

- cortex-m7/cpu/SAME70/include/dspi.h
- cortex-m7/cpu/MIMXRT10xx/include/dspi.h

**16.169 SRAM\_Timing\_Spec\_t Struct Reference**

Timing specifications for common SRAM speeds.

```
#include <semc_utils.h>
```

**Public Attributes**

- uint32\_t **address\_access\_time\_ns**  
*tAA - Address to data valid*
- uint32\_t **chip\_enable\_access\_ns**  
*tACE - CE to data valid*
- uint32\_t **output\_enable\_access\_ns**  
*tOE - OE to data valid*
- uint32\_t **write\_cycle\_time\_ns**  
*tWC - Write cycle time*
- uint32\_t **read\_cycle\_time\_ns**  
*tRC - Read cycle time*
- uint32\_t **address\_setup\_time\_ns**  
*tAS - Address setup to WE*
- uint32\_t **address\_hold\_time\_ns**  
*tAH - Address hold from WE*
- uint32\_t **chip\_select\_setup\_ns**  
*tCS - CS setup time*
- uint32\_t **write\_pulse\_width\_ns**  
*tWP - WE pulse width*
- uint32\_t **output\_hold\_time\_ns**  
*tOH - Data hold after OE high*

**16.169.1 Detailed Description**

Timing specifications for common SRAM speeds.

These structures contain typical timing values for different speed grades of asynchronous SRAM devices. Use these as starting points and adjust based on your specific device datasheet.

The documentation for this struct was generated from the following file:

- semc\_utils.h

**16.170 SSC\_cfg\_t Struct Reference**

Configuration structure for the SSC driver. Passed to the initialize function to configure the hardware.

```
#include <ssc_i2s.h>
```

**Public Attributes**

- uint16\_t [clkDiv](#) {150}
- [SSC\\_rxtx\\_cfg\\_t rx](#)
- [SSC\\_rxtx\\_cfg\\_t tx](#)

**16.170.1 Detailed Description**

Configuration structure for the SSC driver. Passed to the initialize function to configure the hardware.

**16.170.2 Member Data Documentation****16.170.2.1 clkDiv**

```
uint16_t SSC_cfg_t::clkDiv {150}
```

The divider value for the peripheral clock used to generate the bit clock when CLK\_SRC\_MCK is selected.

### 16.170.2.2 rx

`SSC_rxtx_cfg_t` `SSC_cfg_t::rx`

The receive configuration.

### 16.170.2.3 tx

`SSC_rxtx_cfg_t` `SSC_cfg_t::tx`

The transmit configuration.

The documentation for this struct was generated from the following file:

- `ssc_i2s.h`

## 16.171 SSC\_rxtx\_cfg\_t Struct Reference

Configuration structure for a given direction (rx or tx) of the SSC module. Passed to the initialize function to configure the hardware.

```
#include <ssc_i2s.h>
```

### Public Attributes

- bool `enable` {false}
- uint8\_t `period` {64}
- uint8\_t `startDly` {1}
- startCond\_t `startCond` {START\_FRAME\_EDGE}
- clkGate\_t `clkGate` {CLK\_GATE\_CONTINUOUS}
- dataValid\_t `dataValid` {DATA\_VALID\_RISING}
- clkOut\_t `clkOut` {CLK\_OUT\_INPUT}
- clkSrc\_t `clkSrc` {CLK\_SRC\_RK}
- uint8\_t `syncLen` {0}
- frameEdge\_t `syncEdge` {FRAME\_SYNC\_FALLING}
- bool `syncDataEnabled` {false}
- frameSyncOut\_t `syncOut` {FRAME\_SYNC\_INPUT}
- uint8\_t `wordsPerFrame` {1}
- bitOrder\_t `bitOrder` {MOST\_SIG\_FIRST}
- bool `lineIdleState` {1}
- uint8\_t `bitsPerWord` {24}
- bufferDepletionBehavior\_t `depletionBehavior` {DEPLETED\_PAUSE}

### 16.171.1 Detailed Description

Configuration structure for a given direction (rx or tx) of the SSC module. Passed to the initialize function to configure the hardware.

### 16.171.2 Member Data Documentation

#### 16.171.2.1 bitOrder

`bitOrder_t` `SSC_rxtx_cfg_t::bitOrder` {MOST\_SIG\_FIRST}

The bit order of transitions.

#### 16.171.2.2 bitsPerWord

`uint8_t` `SSC_rxtx_cfg_t::bitsPerWord` {24}

The bit width of words to be transferred.



### 16.171.2.3 clkGate

```
clkGate_t SSC_rxtx_cfg_t::clkGate {CLK_GATE_CONTINUOUS}
```

The clock gating condition for this channel.

### 16.171.2.4 clkOut

```
clkOut_t SSC_rxtx_cfg_t::clkOut {CLK_OUT_INPUT}
```

The clock out condition for this channel.

### 16.171.2.5 clkSrc

```
clkSrc_t SSC_rxtx_cfg_t::clkSrc {CLK_SRC_RK}
```

The clock source for this channel.

### 16.171.2.6 dataValid

```
dataValid_t SSC_rxtx_cfg_t::dataValid {DATA_VALID_RISING}
```

The data valid condition for this channel.

### 16.171.2.7 depletionBehavior

```
bufferDepletionBehavior_t SSC_rxtx_cfg_t::depletionBehavior {DEPLETED_PAUSE}
```

The depletion behavior that should be used for driver DMA transfers.

### 16.171.2.8 enable

```
bool SSC_rxtx_cfg_t::enable {false}
```

Whether this direction should be enabled.

### 16.171.2.9 lineIdleState

```
bool SSC_rxtx_cfg_t::lineIdleState {1}
```

The state the line should be driven to when not transmitting a word.

### 16.171.2.10 period

```
uint8_t SSC_rxtx_cfg_t::period {64}
```

The period of each frame, in terms of the bit clock.

### 16.171.2.11 startCond

```
startCond_t SSC_rxtx_cfg_t::startCond {START_FRAME_EDGE}
```

The starting condition for this channel.

### 16.171.2.12 startDly

```
uint8_t SSC_rxtx_cfg_t::startDly {1}
```

How many bit periods should data be delayed by at the start of the frame.

### 16.171.2.13 syncDataEnabled

```
bool SSC_rxtx_cfg_t::syncDataEnabled {false}
```

Whether synchronizing data is used.

### 16.171.2.14 syncEdge

```
frameEdge_t SSC_rxtx_cfg_t::syncEdge {FRAME_SYNC_FALLING}
```

The frame sync edge defining frame start.

### 16.171.2.15 syncLen

```
uint8_t SSC_rxtx_cfg_t::syncLen {0}
```

The bit length of the synchronization data (when used).

### 16.171.2.16 syncOut

```
frameSyncOut_t SSC_rxtx_cfg_t::syncOut {FRAME_SYNC_INPUT}
```

The frame sync output configuration.

### 16.171.2.17 wordsPerFrame

```
uint8_t SSC_rxtx_cfg_t::wordsPerFrame {1}
```

Number of words contained in a frame.

The documentation for this struct was generated from the following file:

- `ssc_i2s.h`

## 16.172 SSCctx\_t Class Reference

```
#include <ssc_i2s.h>
```

### Public Member Functions

- `int Init (const SSC\_cfg\_t &cfg)`  
*Initializes the SSC hardware and driver context.*
- `void Shutdown ()`  
*Shuts down the SSC hardware and driver.*
- `int getCurrentConfig (SSC\_cfg\_t &cfg)`  
*Fills in the config object with the current active configuration.*
- `ctxState_t getState ()`  
*Returns the current driver state.*
- `int TransmitBuffer (void *buffer, uint32_t bufferLen, bool waitIfNeeded)`  
*Hands off a buffer to be transmitted by the SSC driver.*
- `int ReadyReceiveBuffer (void *buffer, uint32_t bufferLen, bool waitIfNeeded)`  
*Hands off a buffer to be written to by the SSC driver.*
- `void RegisterTxBufferDoneCB (SSC_BufferDoneFn_t cb)`  
*Registers a callback for when a transmit buffer is finished.*
- `void RegisterRxBufferDoneCB (SSC_BufferDoneFn_t cb)`  
*Registers a callback for when a receive buffer is finished.*

### 16.172.1 Detailed Description

`SSCctx_t` is a driver for the Synchronous Serial Controller. It operates on a buffer level, where buffers are handed to the system and either transmitted or filled with received data as appropriate.

### 16.172.2 Member Function Documentation

#### 16.172.2.1 getCurrentConfig()

```
int SSCctx_t::getCurrentConfig (
 SSC_cfg_t & cfg)
```

Fills in the config object with the current active configuration.

#### Parameters

|     |     |                                       |
|-----|-----|---------------------------------------|
| out | cfg | The configuration object to populate. |
|-----|-----|---------------------------------------|

**Returns**

Negative on failure.

**16.172.2.2 getState()**

```
ctxState_t SSCCtx_t::getState ()
```

Returns the current driver state.

**Returns**

The current driver state.

**16.172.2.3 Init()**

```
int SSCCtx_t::Init (
 const SSC_cfg_t & cfg)
```

Initializes the SSC hardware and driver context.

**Parameters**

|           |            |                           |
|-----------|------------|---------------------------|
| <i>in</i> | <i>cfg</i> | The configuration to use. |
|-----------|------------|---------------------------|

**Returns**

Negative on failure.

**16.172.2.4 ReadyReceiveBuffer()**

```
int SSCCtx_t::ReadyReceiveBuffer (
 void * buffer,
 uint32_t bufferLen,
 bool waitIfNeeded)
```

Hands off a buffer to be written to by the SSC driver.

**Parameters**

|                     |                                                                                                             |
|---------------------|-------------------------------------------------------------------------------------------------------------|
| <i>buffer</i>       | A pointer to the buffer to be written to.                                                                   |
| <i>bufferLen</i>    | The length of the buffer to be written. (Must be multiples of 1, 2, or 4 bytes depending on word bit width) |
| <i>waitIfNeeded</i> | Whether the driver should wait for space to receive or fail immediately upon exhausting the queue depth.    |

**Returns**

Negative on failure.

**16.172.2.5 TransmitBuffer()**

```
int SSCCtx_t::TransmitBuffer (
 void * buffer,
 uint32_t bufferLen,
 bool waitIfNeeded)
```

Hands off a buffer to be transmitted by the SSC driver.

## Parameters

|                     |                                                                                                              |
|---------------------|--------------------------------------------------------------------------------------------------------------|
| <i>buffer</i>       | A pointer to the buffer to be transmit.                                                                      |
| <i>bufferLen</i>    | The length of the buffer to be transmit. (Must be multiples of 1, 2, or 4 bytes depending on word bit width) |
| <i>waitIfNeeded</i> | Whether the driver should wait for space to transmit or fail immediately upon exhausting the queue depth.    |

## Returns

Negative on failure.

The documentation for this class was generated from the following files:

- ssc\_i2s.h
- ssc\_i2s.cpp

## 16.173 StopWatch Class Reference

High-resolution stopwatch timer for precise event timing and performance measurement.

```
#include <stopwatch.h>
```

## Public Member Functions

- [StopWatch](#) (int timer\_number=FIRST\_UNUSED\_TIMER)  
*Construct a stopwatch object with optional timer selection.*
- void [Start](#) (bool clear\_count=false)  
*Start or resume the stopwatch timer.*
- void [Clear](#) ()  
*Clear the accumulated elapsed time.*
- unsigned long long [Stop](#) ()  
*Stop the stopwatch and return elapsed time.*
- unsigned long long [GetTime](#) ()  
*Get the current elapsed time without stopping the timer.*
- double [CountResolution](#) ()  
*Get the resolution of a single timer tick in seconds.*
- double [Convert](#) (unsigned long long ticks)  
*Convert timer ticks to seconds.*

### 16.173.1 Detailed Description

High-resolution stopwatch timer for precise event timing and performance measurement.

The [StopWatch](#) class provides a hardware-based timer interface for measuring time intervals with resolution far exceeding the RTOS tick rate. It is ideal for performance profiling, timeout detection, and precise event timing.

The stopwatch operates in two states:

- **Running:** Timer is actively counting elapsed time
- **Stopped:** Timer preserves elapsed time but does not increment

Time accumulates across multiple Start/Stop cycles unless explicitly cleared, making it useful for measuring cumulative execution time across multiple code sections or function calls.

#### 16.173.1.1 Thread Safety

[StopWatch](#) objects are not inherently thread-safe. If multiple tasks need to access the same [StopWatch](#), appropriate synchronization (mutexes, semaphores) must be used to prevent race conditions.

### 16.173.1.2 Timer Allocation

On most platforms, hardware timers are automatically allocated. Some older platforms may require explicit timer number specification via the constructor.

#### Note

The internal time representation uses unsigned long long to prevent overflow for extended timing periods, supporting timing intervals of years without rollover on typical hardware.

## 16.173.2 Constructor & Destructor Documentation

### 16.173.2.1 Stopwatch()

```
StopWatch::StopWatch (
 int timer_number = FIRST_UNUSED_TIMER)
```

Construct a stopwatch object with optional timer selection.

Creates a new [StopWatch](#) instance and allocates a hardware timer for timing operations. On modern platforms with dedicated RTOS timers, the timer\_number parameter can be omitted and the system will automatically select an available timer resource.

The stopwatch is created in a stopped state with zero elapsed time. Call [Start\(\)](#) to begin timing.

#### Parameters

|                     |                                                                                                                                                                                                                                   |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>timer_number</i> | Optional timer resource identifier. Use FIRST_UNUSED_TIMER (default) for automatic allocation, or specify a specific timer number if your platform requires explicit timer selection. Valid timer numbers are platform-dependent. |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### Note

After construction, the stopwatch is in a stopped state with [GetTime\(\)](#) returning 0 until [Start\(\)](#) is called.

#### Warning

On platforms with limited timer resources, ensure you don't create more [StopWatch](#) instances than available hardware timers. Check your platform documentation for timer availability.

#### Example:

```
StopWatch timer1; // Automatic timer allocation
StopWatch timer2(FIRST_UNUSED_TIMER); // Explicit automatic allocation
StopWatch timer3(2); // Use specific timer #2 (platform-dependent)
```

## 16.173.3 Member Function Documentation

### 16.173.3.1 Clear()

```
void StopWatch::Clear ()
```

Clear the accumulated elapsed time.

Resets the stopwatch's elapsed time counter to zero. If the stopwatch is currently running, it continues running but from a zero starting point. If stopped, the elapsed time is simply cleared.

This is equivalent to calling [Start\(true\)](#) but works whether the stopwatch is running or stopped.

#### Note

After calling [Clear\(\)](#), [GetTime\(\)](#) will return 0 until time accumulates.

This function does not change the running/stopped state of the stopwatch.

#### Example:

```
StopWatch timer;

timer.Start();
DoWork();
```

```

timer.Stop();

printf("First measurement: %llu\n", timer.GetTime());

timer.Clear(); // Reset to zero

timer.Start();
DoMoreWork();
timer.Stop();

printf("Second measurement: %llu\n", timer.GetTime());

```

**See also**[Start\(\)](#)[Stop\(\)](#)**16.173.3.2 Convert()**

```

double Stopwatch::Convert (
 unsigned long long ticks)

```

Convert timer ticks to seconds.

Converts a time interval or elapsed time value from hardware timer ticks (as returned by [Stop\(\)](#) or [GetTime\(\)](#)) to seconds as a double precision floating point number. This provides a human-readable representation of the measured time.

The conversion accounts for the platform's timer frequency and provides accurate time representation suitable for display, logging, or mathematical operations requiring time in standard units.

**Parameters**

|              |                                                                                                                                          |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ticks</i> | The time value in hardware timer ticks to convert. This is typically obtained from <a href="#">Stop()</a> or <a href="#">GetTime()</a> . |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------|

**Returns**

The equivalent time in seconds as a double precision floating point number. The precision depends on the timer resolution (see [CountResolution\(\)](#)).

**Note**

The input value should be a tick count from this [StopWatch](#) instance. Tick values from other timers or sources may have different meanings.

The conversion is: seconds = ticks x [CountResolution\(\)](#)

For very long timing intervals (hours/days), be aware of floating point precision limitations in the double type, though this is rarely an issue in practice for timing applications.

**Example - Basic conversion:**

```

StopWatch timer;

timer.Start();
PerformOperation();
unsigned long long ticks = timer.Stop();

double seconds = timer.Convert(ticks);
printf("Operation took %f seconds\n", seconds);

```

**Example - Display in multiple units:**

```

StopWatch timer;

timer.Start();
DoWork();
unsigned long long ticks = timer.Stop();

double seconds = timer.Convert(ticks);
double milliseconds = seconds * 1000.0;
double microseconds = seconds * 1000000.0;

printf("Elapsed time:\n");
printf(" %f seconds\n", seconds);
printf(" %f milliseconds\n", milliseconds);
printf(" %f microseconds\n", microseconds);

```

```
printf(" %llu ticks\n", ticks);
```

**Example - Performance comparison:**

```
StopWatch timer;

timer.Start();
Method1();
unsigned long long time1 = timer.Stop();

timer.Start(true); // Clear and restart
Method2();
unsigned long long time2 = timer.Stop();

double seconds1 = timer.Convert(time1);
double seconds2 = timer.Convert(time2);

printf("Method1: %f seconds\n", seconds1);
printf("Method2: %f seconds\n", seconds2);

if (seconds1 < seconds2)
 printf("Method1 is %.2fx faster\n", seconds2 / seconds1);
else
 printf("Method2 is %.2fx faster\n", seconds1 / seconds2);
```

**Example - Throughput calculation:**

```
StopWatch timer;
int itemsProcessed = 1000000;

timer.Start();
for (int i = 0; i < itemsProcessed; i++)
 ProcessItem(i);
unsigned long long ticks = timer.Stop();

double seconds = timer.Convert(ticks);
double itemsPerSecond = itemsProcessed / seconds;
double secondsPerItem = seconds / itemsProcessed;

printf("Processed %d items in %f seconds\n", itemsProcessed, seconds);
printf("Throughput: %.2f items/second\n", itemsPerSecond);
printf("Average: %e seconds/item\n", secondsPerItem);
```

**See also**

[Stop\(\)](#)  
[GetTime\(\)](#)  
[CountResolution\(\)](#)

### 16.173.3.3 CountResolution()

```
double Stopwatch::CountResolution ()
```

Get the resolution of a single timer tick in seconds.

Returns the time value of one hardware timer tick, expressed as a double precision floating point number in seconds.

This represents the finest granularity at which the stopwatch can measure time.

The resolution depends on the hardware timer frequency and typically ranges from nanoseconds to microseconds on NetBurner platforms. Higher resolution (smaller tick duration) provides more precise timing measurements.

**Returns**

The duration of one timer tick in seconds as a double. For example, a return value of 0.000001 indicates 1 microsecond resolution, while 0.000000001 indicates 1 nanosecond resolution.

**Note**

This value is constant for a given platform and timer configuration.

The resolution represents the theoretical precision; actual measurement accuracy may be affected by interrupt latency and system load.

**Example - Display timer characteristics:**

```
StopWatch timer;
double resolution = timer.CountResolution();

printf("Stopwatch timer resolution:\n");
printf(" Per tick: %e seconds\n", resolution);
printf(" Per tick: %f microseconds\n", resolution * 1e6);
printf(" Per tick: %f nanoseconds\n", resolution * 1e9);
```

Example - Calculate measurement precision:

```
StopWatch timer;
double resolution = timer.CountResolution();

timer.Start();
QuickOperation();
unsigned long long ticks = timer.Stop();

double time = timer.Convert(ticks);
double precision = resolution;

printf("Measured time: %f +/- %f seconds\n", time, precision);
```

Example - Validate timing requirements:

```
StopWatch timer;
double resolution = timer.CountResolution();
double required_precision = 0.000001; // 1 microsecond

if (resolution > required_precision)
{
 printf("Warning: Timer resolution (%e s) is insufficient\n", resolution);
 printf("Required: %e s\n", required_precision);
}
else
{
 printf("Timer resolution (%e s) is adequate\n", resolution);
}
```

See also

[Convert\(\)](#)

[GetTime\(\)](#)

#### 16.173.3.4 GetTime()

```
unsigned long long Stopwatch::GetTime ()
```

Get the current elapsed time without stopping the timer.

Returns the current accumulated elapsed time in hardware timer ticks. Unlike [Stop\(\)](#), this function does not change the state of the stopwatch - if the timer is running, it continues running; if stopped, it remains stopped.

This is particularly useful for checking progress during long operations, implementing timeouts, or monitoring timing without interrupting the measurement.

##### Returns

The current accumulated elapsed time in hardware timer ticks. If the stopwatch is running, this includes time up to the moment of the call. If stopped, this returns the time at which [Stop\(\)](#) was called. Use [Convert\(\)](#) to convert this value to seconds.

##### Note

This function can be called while the timer is running or stopped.

The returned value is in hardware timer ticks. Use [Convert\(\)](#) to convert to seconds or other time units.

On running timers, consecutive calls to [GetTime\(\)](#) will return increasing values. On stopped timers, consecutive calls return the same value.

Example - Check progress without stopping:

```
StopWatch timer;
timer.Start();

while (ProcessingData())
{
 // Check time periodically without stopping
 double elapsed = timer.Convert(timer.GetTime());
 if (elapsed > MAX_ALLOWED_SECONDS)
 {
 printf("Warning: Operation exceeding time limit!\n");
 break;
 }
}

timer.Stop();
```

Example - Timeout implementation:

```
StopWatch timer;
```



```

timer.Start();

while (!OperationComplete())
{
 DoWork();

 if (timer.Convert(timer.GetTime()) > TIMEOUT_SECONDS)
 {
 printf("Operation timed out!\n");
 return ERROR_TIMEOUT;
 }
}

timer.Stop();
return SUCCESS;

```

**Example - Progress reporting:**

```

StopWatch timer;
timer.Start();

for (int i = 0; i < totalItems; i++)
{
 ProcessItem(i);

 if (i % 100 == 0)
 {
 double elapsed = timer.Convert(timer.GetTime());
 printf("Processed %d items in %.2f seconds\n", i, elapsed);
 }
}

timer.Stop();

```

**See also**

[Stop\(\)](#)  
[Start\(\)](#)  
[Convert\(\)](#)

**16.173.3.5 Start()**

```

void Stopwatch::Start (
 bool clear_count = false)

```

Start or resume the stopwatch timer.

Begins timing from the current moment. If the stopwatch was previously stopped, it will continue accumulating time from the last elapsed value unless `clear_count` is true. This allows for cumulative timing across multiple Start/Stop cycles.

If called when the stopwatch is already running, this function has no effect unless `clear_count` is true, in which case it restarts timing from zero.

**Parameters**

|                    |                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>clear_count</i> | If true, resets the elapsed time to zero before starting. If false (default), continues accumulating time from the previous elapsed value, enabling cumulative timing. |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Note**

Calling [Start\(\)](#) multiple times without stopping has no effect unless `clear_count` is true.

This function uses hardware timer reads and is very fast (typically < 1 microsecond overhead).

**Example - Basic timing:**

```

StopWatch timer;
timer.Start();
PerformOperation();
timer.Stop();

```

**Example - Cumulative timing:**

```

StopWatch timer;

timer.Start();
Operation1();
timer.Stop();

// Continue timing - adds to previous elapsed time

```

```
timer.Start();
Operation2();
timer.Stop();

// Total time for both operations
double total = timer.Convert(timer.GetTime());
```

**Example - Restart from zero:**

```
StopWatch timer;

timer.Start();
FirstMeasurement();
timer.Stop();

// Start fresh measurement
timer.Start(true); // Clear previous time
SecondMeasurement();
timer.Stop();
```

**See also**

[Stop\(\)](#)  
[Clear\(\)](#)  
[GetTime\(\)](#)

### 16.173.3.6 Stop()

```
unsigned long long Stopwatch::Stop ()
```

Stop the stopwatch and return elapsed time.

Stops the stopwatch timer and returns the total accumulated elapsed time in hardware timer ticks. The elapsed time is preserved and will continue to accumulate if [Start\(\)](#) is called again without clearing.

If the stopwatch is already stopped, this function simply returns the current elapsed time without modifying the stopwatch state.

**Returns**

The total accumulated elapsed time in hardware timer ticks since the stopwatch was started (or last cleared). Use [Convert\(\)](#) to convert this value to seconds.

**Note**

The returned value is in hardware timer ticks, not seconds. Use [Convert\(\)](#) to convert to human-readable time units.

Calling [Stop\(\)](#) multiple times returns the same value until [Start\(\)](#) is called again.

**Example - Basic stop and convert:**

```
StopWatch timer;
timer.Start();
PerformOperation();
unsigned long long ticks = timer.Stop();
double seconds = timer.Convert(ticks);
printf("Operation took %f seconds (%llu ticks)\n", seconds, ticks);
```

**Example - Multiple stop calls:**

```
StopWatch timer;
timer.Start();
DoWork();

unsigned long long time1 = timer.Stop(); // Stops and gets time
unsigned long long time2 = timer.Stop(); // Same value as time1
// time1 == time2
```

**See also**

[Start\(\)](#)  
[GetTime\(\)](#)  
[Convert\(\)](#)

The documentation for this class was generated from the following file:

- stopwatch.h

## 16.174 StringBufferObject Class Reference

Class for NBString-based web client response buffers.

#include <web\_buffers.h>

Inherits [buffer\\_object](#).

### 16.174.1 Detailed Description

Class for NBString-based web client response buffers.

The documentation for this class was generated from the following file:

- [web\\_buffers.h](#)

## 16.175 MQTT::SubHandler Class Reference

Easy class for registering a callback function that receives messages on an [MQTT](#) topic and handles them appropriately. Behaves similarly to [HttpResponseHandler](#).

#include <mqtt.h>

Inherits [MQTT::mqtt\\_leaf](#).

### Public Member Functions

- [SubHandler](#) (SubHandlerCb handler, const char \*topicFilter, [Client](#) \*mClient=NULL, uint16\_t flags=0)  
*SubHandler Constructor for a standalone SubHandler object.*
- virtual ValueTestResult [TestOrSetValue](#) ([ParsedJsonDataSet](#) &pjs, bool setVal) override  
*SubHandler::TestOrSetValue Test whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.*
- virtual int [SubscribeEvent](#) (int mqttFd, [PacketInfo::Msg](#) \*msg) override  
*SubHandler::SubscribeEvent Handle a message sent by the Broker triggered by this object's Subscription.*

### Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- [~mqtt\\_leaf](#) ()  
*~mqtt\_leaf Destructor for base mqtt\_leaf*
- void [RemoveFromTree](#) ()  
*Remove this object from it's parent tree.*
- bool [NameMatch](#) (const char \*cp)  
*Determine whether a given name matches this object.*
- virtual int [GetPubTopicName](#) ([NBString](#) &s)  
*Get the Topic Name this object Publishes to.*
- virtual int [GetSubTopicFilter](#) ([NBString](#) &s)  
*Get the Subscription Topic Filter this object Subscribes to.*
- virtual int [GetRenderLength](#) ([eObj\\_Serialize\\_t](#) form=[eObj\\_Serialize\\_CURRENT\\_FORM](#))  
*Get the number of bytes used to serialize this object in a particular format.*
- int [Publish](#) (int waitForCompletion=-1)  
*Publishes the current MQTT object, optionally waiting for completion.*
- void [SetPublishInterval](#) (uint32\_t intervalPeriodTicks)  
*Sets or clears an automatic publishing interval for this MQTT object.*
- virtual void [SetSerializationForm](#) ([eObj\\_Serialize\\_t](#) form)  
*Set the default serialization format for this object.*
- [eObj\\_Serialize\\_t](#) [GetSerializationForm](#) ()  
*Get the default serialization format for this object.*
- virtual void [SetFlags](#) (uint16\_t newFlags)  
*Set additional behavioral flags for this object.*

- virtual void [ClearFlags](#) (uint16\_t delFlags)  
*Clear behavioral flags from this object.*
- uint16\_t [GetFlags](#) ()  
*Get the behavioral flags set for this object.*
- int [getPriority](#) ()  
*Get the message priority for this object.*

### Protected Member Functions

- virtual int [ParseFd](#) (int mqttFd) override  
*[SubHandler::ParseFd](#) Parse the data from a file descriptor according to this object's current default serialization method and assign the value.*
- virtual int [RenderToBuffer](#) (eObj\_Serialize\_t form, uint8\_t \*buf, int buflen) override  
*[SubHandler::RenderToBuffer](#) Render this object to an output buffer.*
- virtual int [RenderToFd](#) (eObj\_Serialize\_t form, int fd) override  
*[SubHandler::RenderToFd](#) Render this object to an output socket.*

### Protected Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- [mqtt\\_leaf](#) (const char \*topic, [Client](#) \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|eObj\_Flag\_Sub\_N  
*Constructor for [mqtt\\_leaf](#) objects.*
- [mqtt\\_leaf](#) ([mqtt\\_leaf](#) &parent, const char \*topic, [Client](#) \*mClient, uint16\_t flags=(eObj\_Flag\_PubOnWrite|eObj\_Flag\_Subscribe|  
*Constructor for [mqtt\\_leaf](#) objects.*
- virtual [mqtt\\_leaf](#) \* [GetChildren](#) ()  
*Get the first child of this object.*
- virtual [mqtt\\_leaf](#) \* [AddChild](#) ([mqtt\\_leaf](#) \*child)  
*Add an [MqttObj](#) as a child to this object.*
- virtual [mqtt\\_leaf](#) \* [RemoveFirstChild](#) ()  
*Remove the first child from this object.*
- void [ForEachLeaf](#) (LeafCallBack \*p, void \*pd)  
*Call a function on each element of the [MqttObj](#) tree with this object as the root.*
- void [RootParse](#) ([ParsedJsonDataSet](#) &pjs)  
*Assign to this object tree from a JSON dataset.*
- bool [MatchId](#) ([ParsedJsonDataSet](#) &pjs)  
*Determine whether the JSON dataset's current index matches this object's name.*
- int [QueuePublish](#) ()  
*Queue this object to be Published via the registered [Client](#).*
- int [QueueSubscribe](#) ()  
*Queue this object to be Subscribed via the registered [Client](#).*
- int [QueueUnsubscribe](#) ()  
*Queue this object to be Unsubscribed via the registered [Client](#).*
- int [Assign](#) ()  
*Perform post assignemnt operations.*
- uint8\_t [PubState](#) ()  
*Get the Publish state.*
- uint8\_t [SubState](#) ()  
*Get the Subscribe state of this object.*
- virtual void [SetPubState](#) (uint8\_t newState)  
*Set the Publish state.*
- virtual void [SetSubState](#) (uint8\_t newState)  
*Set the Subscribe state.*
- int [HandleSubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)

*Handle a Subscribe Acknowledge message from the broker for our Subscribe request.*

- int [HandleUnsubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)

*Handle an Unsubscribe Acknowledge message from the broker for our Unsubscribe request.*

- int [HandlePubAck](#) (int mqttFd, [PacketInfo::Msg](#) \*msg)

*Handle an Publish Acknowledge message from the broker for our Publish request.*

### Additional Inherited Members

### Public Types inherited from [MQTT::mqtt\\_leaf](#)

- enum [eObj\\_State\\_t](#)

*Mqtt Object Publication/Subscription States.*

### Static Public Member Functions inherited from [MQTT::mqtt\\_leaf](#)

- static int [MqttRxPublish](#) ([Client](#) \*mClient, int mqttFd, [PacketInfo::Msg](#) \*msg, void \*ctx)

*Static message handler for Publish messages received by the registered [Client](#).*

### Protected Attributes inherited from [MQTT::mqtt\\_leaf](#)

- [NBInterpolatedString](#) **topicName**

*Topic name or additional branch.*

- [eObj\\_Serialize\\_t](#) **publishForm**

*Default serialization form.*

- [uint16\\_t](#) **mFlags**

*Behavioral flags.*

## 16.175.1 Detailed Description

Easy class for registering a callback function that receives messages on an [MQTT](#) topic and handles them appropriately. Behaves similarly to [HttpResponseHandler](#).

## 16.175.2 Constructor & Destructor Documentation

### 16.175.2.1 SubHandler()

```
MQTT::SubHandler::SubHandler (
 SubHandlerCb handler,
 const char * topicFilter,
 Client * mClient = NULL,
 uint16_t flags = 0)
```

[SubHandler](#) Constructor for a standalone [SubHandler](#) object.

#### Parameters

|                     |                                                                                                                                    |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>handler</i>      | Function that should be called when a message is received                                                                          |
| <i>*topicFilter</i> | Mqtt Topic Filter to Subscribe to                                                                                                  |
| <i>*mClient</i>     | Pointer to <a href="#">Client</a> to register with. If NULL, it will be registered with the first created <a href="#">Client</a> . |
| <i>flags</i>        | Behavioral flags to set on the created object                                                                                      |

## 16.175.3 Member Function Documentation

### 16.175.3.1 ParseFd()

```
virtual int MQTT::SubHandler::ParseFd (
 int mqttFd) [override], [protected], [virtual]
```

[SubHandler::ParseFd](#) Parse the data from a file descriptor according to this object's current default serialization method and assign the value.

#### Parameters

|               |                               |
|---------------|-------------------------------|
| <i>mqttFd</i> | File descriptor to read from. |
|---------------|-------------------------------|

#### Returns

eResult\_t return code

Implements [MQTT::mqtt\\_leaf](#).

### 16.175.3.2 RenderToBuffer()

```
virtual int MQTT::SubHandler::RenderToBuffer (
 eObj_Serialize_t form,
 uint8_t * buf,
 int buflen) [override], [protected], [virtual]
```

[SubHandler::RenderToBuffer](#) Render this object to an output buffer.

#### Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>form</i>   | Serialization format to render as |
| <i>*buf</i>   | Pointer to output buffer          |
| <i>buflen</i> | Size of the output buffer         |

#### Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

### 16.175.3.3 RenderToFd()

```
virtual int MQTT::SubHandler::RenderToFd (
 eObj_Serialize_t form,
 int fd) [override], [protected], [virtual]
```

[SubHandler::RenderToFd](#) Render this object to an output socket.

#### Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>form</i> | Serialization format to render as |
| <i>fd</i>   | File descriptor to render to      |

#### Returns

Length of the rendered data

Implements [MQTT::mqtt\\_leaf](#).

#### 16.175.3.4 SubscribeEvent()

```
virtual int MQTT::SubHandler::SubscribeEvent (
 int mqttFd,
 PacketInfo::Msg * msg) [override], [virtual]
```

[SubHandler::SubscribeEvent](#) Handle a message sent by the Broker triggered by this object's Subscription.

## Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>mqttFd</i> | File Descriptor to read any additional message data from |
| <i>*msg</i>   | Pointer to message received from the Broker              |

## Returns

eResult\_t return code

Reimplemented from [MQTT::mqtt\\_leaf](#).

**16.175.3.5 TestOrSetValue()**

```
virtual ValueTestResult MQTT::SubHandler::TestOrSetValue (
 ParsedJsonDataSet & pjs,
 bool setVal) [override], [virtual]
```

[SubHandler::TestOrSetValue](#) Test whether the current index of a JSON dataset is valid for this object, or assign the value at the current index of the dataset to this object.

## Parameters

|                 |                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------|
| <i>&amp;pjs</i> | The JSON dataset to use                                                                                 |
| <i>setVal</i>   | true: Assign from the value at the current index, false: test if current index is valid for this object |

## Returns

ValueTestResult indicating the validity of the current index

Implements [MQTT::mqtt\\_leaf](#).

The documentation for this class was generated from the following file:

- mqtt.h

**16.176 MQTT::SubscribeFilter Struct Reference**

Structure used to specify a topic filter for subscriptions.

```
#include <mqtt_msg_reqs.h>
```

**16.176.1 Detailed Description**

Structure used to specify a topic filter for subscriptions.

The documentation for this struct was generated from the following file:

- mqtt\_msg\_reqs.h

**16.177 MQTT::SubscribeFilter\_NbStr Struct Reference**

Structure used to specify a topic filter for subscriptions.

```
#include <mqtt_msg_reqs.h>
```

**16.177.1 Detailed Description**

Structure used to specify a topic filter for subscriptions.

The documentation for this struct was generated from the following file:

- mqtt\_msg\_reqs.h



## 16.178 MQTT::SubscribeRequest Struct Reference

Structure used to specify details for a Subscribe call.

```
#include <mqtt_msg_reqs.h>
```

### 16.178.1 Detailed Description

Structure used to specify details for a Subscribe call.

The documentation for this struct was generated from the following file:

- mqtt\_msg\_reqs.h

## 16.179 TFTP\_Server Class Reference

TFTP Server Class.

```
#include <tftp_server.h>
```

Inherits servlet, and TimeOutElement.

### Public Member Functions

- [TFTP\\_Server](#) (uint16\_t port=69)  
*TFTP server constructor.*
- void [Add](#) ([TFTP\\_Server\\_source\\_File](#) &sf)  
*Specify TFTP Server listen port number, default is 69.*

### 16.179.1 Detailed Description

TFTP Server Class.

Creates a TFTP Server Servlet

### 16.179.2 Constructor & Destructor Documentation

#### 16.179.2.1 TFTP\_Server()

```
TFTP_Server::TFTP_Server (
 uint16_t port = 69)
```

TFTP server constructor.

#### Parameters

|             |                                             |
|-------------|---------------------------------------------|
| <i>port</i> | Optional listen port number. Default is 69. |
|-------------|---------------------------------------------|

### 16.179.3 Member Function Documentation

#### 16.179.3.1 Add()

```
void TFTP_Server::Add (
 TFTP_Server_source_File & sf)
```

Specify TFTP Server listen port number, default is 69.

#### Parameters

|           |                             |
|-----------|-----------------------------|
| <i>sf</i> | Reference to file to share. |
|-----------|-----------------------------|

The documentation for this class was generated from the following file:

- tftp\_server.h

## 16.180 TFTP\_Server\_source\_File Class Reference

TFTP Server Source File.

```
#include <tftp_server.h>
```

### Friends

- class **TFTP\_Server**

### 16.180.1 Detailed Description

TFTP Server Source File.

Specify a source file to serve

The documentation for this class was generated from the following file:

- tftp\_server.h

## 16.181 TimeZoneRecord Struct Reference

Time zone record structure containing complete time zone information.

```
#include <timezones.h>
```

### Public Attributes

- const char \* [Posix](#)  
*POSIX-compliant time zone string.*
- const char \* [Description](#)  
*Human-readable time zone description.*
- const char \* [Name](#)  
*Formal time zone name.*
- const char \* [Short\\_name](#)  
*Abbreviated time zone name.*
- int [bUsCanada](#)  
*US/Canada time zone flag.*

### 16.181.1 Detailed Description

Time zone record structure containing complete time zone information.

This structure defines a time zone with all necessary information for proper time conversion and display. The NetBurner system provides an array of over 75 pre-configured time zones covering all major regions worldwide.

The POSIX timezone string format is:

```
std offset[dst[offset][,start[/time],end[/time]]]
```

Where:

- `std` = Standard time zone abbreviation (e.g., "PST", "CET")
- `offset` = Hours offset from UTC (e.g., "8" for PST = UTC-8)
- `dst` = Daylight saving time abbreviation (e.g., "PDT", "CEST")
- `start` = DST start rule (e.g., "M3.2.0" = 2nd Sunday in March)
- `end` = DST end rule (e.g., "M11.1.0" = 1st Sunday in November)

### Note

All time zone records are defined as const data in timezones.cpp

**Example Time Zone Records:**

```
// Pacific Time (US West Coast)
{"PST8PDT,M3.2.0,M11.1.0",
 "(GMT-08:00) Pacific Time (US and Canada); Tijuana",
 "Pacific Standard Time",
 "PST ",
 1}

// Central European Time
{"CET-1CEST,M3.5.0,M10.5.0/3",
 "(GMT+01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna",
 "W. Europe Standard Time",
 "CET ",
 0}

// Japan (no DST)
{"JST-9",
 "(GMT+09:00) Osaka, Sapporo, Tokyo",
 "Tokyo Standard Time",
 "JST ",
 0}
```

**Usage Example:**

```
#include <timezones.h>
#include <ntp.h>

void SetupTimeZone() {
 // Search for Pacific Time zone
 const TimeZoneRecord *tz = TZRecords;
 while (tz->Posix != NULL) {
 if (strstr(tz->Description, "Pacific Time") != NULL) {
 // Set the system time zone
 tzset(tz->Posix);
 iprintf("Set timezone to: %s\r\n", tz->Description);
 break;
 }
 tz++;
 }
}
```

**16.181.2 Member Data Documentation****16.181.2.1 bUsCanada**

```
int TimeZoneRecord::bUsCanada
```

US/Canada time zone flag.

Indicates whether this time zone is used in the United States and/or Canada. This flag can be used to filter time zones by region or to apply region-specific time zone selection logic.

**Returns**

- 1: Time zone is used in US and/or Canada
- 0: Time zone is used elsewhere (Europe, Asia, etc.)

**Note**

Some time zones span multiple regions. This flag indicates primary usage region.

**16.181.2.2 Description**

```
const char* TimeZoneRecord::Description
```

Human-readable time zone description.

A descriptive string that includes the GMT offset and major cities or regions in the time zone. This is useful for displaying time zone selections to users.

Format: "(GMT +/- HH:MM) City/Region descriptions"

Examples:

- "(GMT-08:00) Pacific Time (US and Canada); Tijuana"
- "(GMT+01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna"
- "(GMT+09:00) Osaka, Sapporo, Tokyo"

### 16.181.2.3 Name

```
const char* TimeZoneRecord::Name
```

Formal time zone name.

The official or formal name of the time zone, often matching Windows time zone names for compatibility. Used for documentation and display.

Examples:

- "Pacific Standard Time"
- "W. Europe Standard Time"
- "Tokyo Standard Time"

### 16.181.2.4 Posix

```
const char* TimeZoneRecord::Posix
```

POSIX-compliant time zone string.

The POSIX timezone string defines the time zone offset from UTC and optional daylight saving time rules. This string is used with the standard `tzset()` function to configure the system's time zone.

Format examples:

- "PST8PDT,M3.2.0,M11.1.0" - Pacific Time with DST
- "CET-1CEST,M3.5.0,M10.5.0/3" - Central European Time with DST
- "JST-9" - Japan Standard Time (no DST)
- "UTC0" - Coordinated Universal Time

#### Note

Negative offsets indicate zones west of UTC, positive east of UTC

### 16.181.2.5 Short\_name

```
const char* TimeZoneRecord::Short_name
```

Abbreviated time zone name.

A short 3-6 character abbreviation for the standard time zone name. This is typically the standard time abbreviation (not DST). Padded with spaces to 6 characters for alignment.

Examples:

- "PST " (Pacific Standard Time)
- "CET " (Central European Time)
- "JST " (Japan Standard Time)
- "UTC " (Coordinated Universal Time)

The documentation for this struct was generated from the following file:

- `timezones.h`

## 16.182 MQTT::TopicHandler Struct Reference

[TopicHandler](#) context definition.

```
#include <mqtt.h>
```

**Public Attributes**

- **NBString** **topicFilter**  
*Filter used to describe which topics this handler covers.*
- **MsgCallback\_t** **handleMsg**  
*Handler method to call.*
- void \* **ctx**  
*User provided extra data passed to the handler method.*

**16.182.1 Detailed Description**

**TopicHandler** context definition.

This struture holds the context information that a **Client** uses to dispatch recieved messages to the appropriate callback function to handle that message.

The documentation for this struct was generated from the following file:

- mqtt.h

**16.183 TwoWire Class Reference**

Wire interface class.

```
#include <Wire.h>
```

**Public Member Functions**

- **TwoWire** (void)  
*Wire Constructor with platform default I2C module.*
- **TwoWire** (int mod)  
*Wire Constructor with specified module.*
- **TwoWire** (PinIO scl, PinIO sda)  
*Constructor.*
- void **setClock** (uint32\_t clockbaud)  
*Set baud rate.*
- void **begin** (uint8\_t slave\_addr=0)  
*Inititalize the system.*
- void **end** ()  
*Turn system off.*
- void **beginTransaction** (uint8\_t address)  
*Set up a start for writing.*
- int **endTransmission** (bool bStop=true)  
*Do a stop with restart.*
- size\_t **write** (const uint8\_t \*pdata, int len)  
*Send data to the target device.*
- size\_t **write** (char v)  
*Send a char to the target device.*
- size\_t **write** (uint8\_t v)  
*Send an int to the target device.*
- size\_t **write** (const char \*pStr, int len=-1)  
*Send a string to the target device.*
- int **requestFrom** (uint8\_t address, size\_t quantity, bool stop=true)  
*Request bytes from the target device.*
- size\_t **available** ()  
*Check if any bytes are available to read.*

- `size_t read (uint8_t *pbuf, size_t maxlen)`  
*Read available data up to maxlen bytes (Not a standard Wire function)*
- `uint8_t read ()`  
*Read the next byte from the buffer.*

#### Target functions

- void `onReceive (slave_rx_handler *handler)`  
*Register a Receive callback (Does not work on the [BBI2C](#) software interface)*
- void `onRequest (slave_rq_handler *handler)`  
*Register a Request callback (Does not work on the [BBI2C](#) software interface)*
- void `setWireTimeout (int timeout_usec=0, bool reset_on_timeout=false)`  
*Set the completion timeout.*
- void `clearTimeout ()`  
*Clear the completion timeout.*
- bool `getWireTimeoutFlag ()`  
*Get whether there is a completion timeout set.*

#### Non-standard extensions

- void `reset ()`  
*Reset the [I2C](#) system and unhang the bus.*
- bool `ping (uint8_t addr)`  
*Check to see if addr exists on this bus.*
- int `writeRegN (uint8_t dAddr, uint32_t reg, const uint8_t *buf, uint32_t blen)`  
*Write a number of 8-bit values to an [I2C](#) slave to the specified register address.*
- int `readRegN (uint8_t dAddr, uint32_t reg, uint8_t *buf, uint32_t blen)`  
*Read a number of 8-bit values from an [I2C](#) slave at the specified register address.*
- int `writeReg8 (uint8_t dAddr, uint32_t reg, uint8_t data)`  
*Write an 8-bit value to a [I2C](#) target device register.*
- int `readReg8 (uint8_t dAddr, uint32_t reg, uint8_t &data)`  
*Read an 8-bit value from an [I2C](#) target device register.*

### 16.183.1 Detailed Description

Wire interface class.

### 16.183.2 Constructor & Destructor Documentation

#### 16.183.2.1 TwoWire() [1/2]

```
TwoWire::TwoWire (
 int mod)
```

Wire Constructor with specified module.

##### Parameters

|            |                                   |
|------------|-----------------------------------|
| <i>mod</i> | <a href="#">I2C</a> module number |
|------------|-----------------------------------|

#### 16.183.2.2 TwoWire() [2/2]

```
TwoWire::TwoWire (
 PinIO scl,
 PinIO sda)
```

Constructor.

## Parameters

|            |         |
|------------|---------|
| <i>scl</i> | SCL pin |
| <i>sda</i> | SDA pin |

If [BBI2C](#) is defined, this will automatically use software [I2C](#) if the specified pins don't map to a hardware [I2C](#) device

### 16.183.3 Member Function Documentation

#### 16.183.3.1 available()

```
size_t TwoWire::available ()
```

Check if any bytes are available to read.

## Return values

|               |                                  |
|---------------|----------------------------------|
| <i>Number</i> | of bytes available in the buffer |
|---------------|----------------------------------|

#### 16.183.3.2 begin()

```
void TwoWire::begin (
 uint8_t slave_addr = 0)
```

Initialize the system.

## Parameters

|                   |                              |
|-------------------|------------------------------|
| <i>slave_addr</i> | Address of the target device |
|-------------------|------------------------------|

#### 16.183.3.3 beginTransmission()

```
void TwoWire::beginTransmission (
 uint8_t address)
```

Set up a start for writing.

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>address</i> | Address of the target device |
|----------------|------------------------------|

#### 16.183.3.4 endTransmission()

```
int TwoWire::endTransmission (
 bool bStop = true)
```

Do a stop with restart.

## Parameters

|              |                                                    |
|--------------|----------------------------------------------------|
| <i>bStop</i> | If true, sends a stop message and releases the bus |
|--------------|----------------------------------------------------|

## Return values

|   |                 |
|---|-----------------|
| 0 | Success         |
| 1 | Data too big    |
| 2 | NACK on address |

|   |              |
|---|--------------|
| 3 | NACK on data |
| 4 | Other error  |
| 5 | Timeout      |

#### 16.183.3.5 onReceive()

```
void TwoWire::onReceive (
 slave_rx_handler * handler)
```

Register a Receive callback (Does not work on the [BBI2C](#) software interface)

##### Parameters

|                |                                                         |
|----------------|---------------------------------------------------------|
| <i>handler</i> | Pointer to the function to call when bytes are received |
|----------------|---------------------------------------------------------|

#### 16.183.3.6 onRequest()

```
void TwoWire::onRequest (
 slave_rq_handler * handler)
```

Register a Request callback (Does not work on the [BBI2C](#) software interface)

##### Parameters

|                |                                                          |
|----------------|----------------------------------------------------------|
| <i>handler</i> | Pointer to the function to call when bytes are requested |
|----------------|----------------------------------------------------------|

#### 16.183.3.7 ping()

```
bool TwoWire::ping (
 uint8_t addr)
```

Check to see if addr exists on this bus.

##### Return values

|             |                       |
|-------------|-----------------------|
| <i>true</i> | if the address exists |
|-------------|-----------------------|

#### 16.183.3.8 read() [1/2]

```
uint8_t TwoWire::read () [inline]
```

Read the next byte from the buffer.

##### Return values

|            |                              |
|------------|------------------------------|
| <i>One</i> | byte of data from the buffer |
|------------|------------------------------|

#### 16.183.3.9 read() [2/2]

```
size_t TwoWire::read (
 uint8_t * pbuf,
 size_t maxlen)
```

Read available data up to maxlen bytes (Not a standard Wire function)



## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>pbuf</i>   | Pointer to a buffer to read bytes into |
| <i>maxlen</i> | Maximum number of bytes to read        |

## Return values

|   |                              |
|---|------------------------------|
| 0 | if no read data is available |
|---|------------------------------|

**16.183.3.10 readReg8()**

```
int TwoWire::readReg8 (
 uint8_t dAddr,
 uint32_t reg,
 uint8_t & data) [inline]
```

Read an 8-bit value from an [I2C](#) target device register.

## Parameters

|              |                                                           |
|--------------|-----------------------------------------------------------|
| <i>dAddr</i> | Address of <a href="#">I2C</a> device                     |
| <i>reg</i>   | Register address to read                                  |
| <i>data</i>  | Reference to variable in which to store the register data |

## Returns

result code.

**16.183.3.11 readRegN()**

```
int TwoWire::readRegN (
 uint8_t dAddr,
 uint32_t reg,
 uint8_t * buf,
 uint32_t blen)
```

Read a number of 8-bit values from an [I2C](#) slave at the specified register address.

## Parameters

|              |                                          |
|--------------|------------------------------------------|
| <i>dAddr</i> | Address of <a href="#">I2C</a> device    |
| <i>reg</i>   | Register address to read                 |
| <i>*buf</i>  | Pointer to buffer to store received data |
| <i>blen</i>  | Number of bytes to read                  |

## Returns

result code.

**16.183.3.12 requestFrom()**

```
int TwoWire::requestFrom (
 uint8_t address,
 size_t quantity,
 bool stop = true)
```

Request bytes from the target device.

## Parameters

|                 |                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------|
| <i>address</i>  | Address of the target device                                                              |
| <i>quantity</i> | How many bytes to request                                                                 |
| <i>stop</i>     | Set true to send a stop message and releases the bus when done, otherwise sends a restart |

## Return values

|               |               |
|---------------|---------------|
| <i>number</i> | of bytes read |
|---------------|---------------|

**16.183.3.13 setClock()**

```
void TwoWire::setClock (
 uint32_t clockbaud) [inline]
```

Set baud rate.

## Parameters

|                  |                            |
|------------------|----------------------------|
| <i>clockbaud</i> | Baud rate (default 200Khz) |
|------------------|----------------------------|

**16.183.3.14 setWireTimeout()**

```
void TwoWire::setWireTimeout (
 int timeout_usec = 0,
 bool reset_on_timeout = false)
```

Set the completion timeout.

## Parameters

|                         |                                                                                  |
|-------------------------|----------------------------------------------------------------------------------|
| <i>timeout_usec</i>     | Timeout in microseconds                                                          |
| <i>reset_on_timeout</i> | Set true to reset the <a href="#">I2C</a> system and unhang the bus upon timeout |

**16.183.3.15 write() [1/4]**

```
size_t TwoWire::write (
 char v) [inline]
```

Send a char to the target device.

## Parameters

|          |                    |
|----------|--------------------|
| <i>v</i> | Character to write |
|----------|--------------------|

## Return values

|               |                  |
|---------------|------------------|
| <i>number</i> | of bytes written |
|---------------|------------------|

**16.183.3.16 write() [2/4]**

```
size_t TwoWire::write (
 const char * pStr,
 int len = -1) [inline]
```

Send a string to the target device.

## Parameters

|             |                        |
|-------------|------------------------|
| <i>pStr</i> | C-style string to send |
| <i>len</i>  | Length of data         |

## Return values

|               |                  |
|---------------|------------------|
| <i>number</i> | of bytes written |
|---------------|------------------|

**16.183.3.17 write() [3/4]**

```
size_t TwoWire::write (
 const uint8_t * pdata,
 int len)
```

Send data to the target device.

## Parameters

|              |                         |
|--------------|-------------------------|
| <i>pdata</i> | Pointer to data to send |
| <i>len</i>   | Length of data          |

## Return values

|               |                  |
|---------------|------------------|
| <i>number</i> | of bytes written |
|---------------|------------------|

**16.183.3.18 write() [4/4]**

```
size_t TwoWire::write (
 uint8_t v) [inline]
```

Send an int to the target device.

## Parameters

|          |                       |
|----------|-----------------------|
| <i>v</i> | Integer data to write |
|----------|-----------------------|

## Return values

|               |                  |
|---------------|------------------|
| <i>number</i> | of bytes written |
|---------------|------------------|

**16.183.3.19 writeReg8()**

```
int TwoWire::writeReg8 (
 uint8_t dAddr,
 uint32_t reg,
 uint8_t data) [inline]
```

Write an 8-bit value to a [I2C](#) target device register.

## Parameters

|              |                                       |
|--------------|---------------------------------------|
| <i>dAddr</i> | Address of <a href="#">I2C</a> device |
| <i>reg</i>   | Register address to write             |
| <i>data</i>  | Data to write to register             |

**Returns**

result code

**16.183.3.20 writeRegN()**

```
int TwoWire::writeRegN (
 uint8_t dAddr,
 uint32_t reg,
 const uint8_t * buf,
 uint32_t blen)
```

Write a number of 8-bit values to an [I2C](#) slave to the specified register address.

Executing this function will send the following: a start bit, the target device address and read/write bit, the 8-bit data bytes, and finally a stop bit. Note that the address is not incremented with each 8-bit data value.

**Parameters**

|              |                                            |
|--------------|--------------------------------------------|
| <i>dAddr</i> | Address of <a href="#">I2C</a> device      |
| <i>reg</i>   | Register address to write                  |
| <i>*buf</i>  | Pointer to buffer containing data to write |
| <i>blen</i>  | Number of bytes to write                   |

**Returns**

result code.

The documentation for this class was generated from the following file:

- Wire.h

**16.184 UDPPacket Class Reference**

UDP Packet Class - Complete UDP packet management.

```
#include <udp.h>
```

**Public Member Functions**

- [UDPPacket](#) (OS\_FIFO \*pFifo, TickTimeout timeout)  
*Create UDP packet from FIFO with timeout.*
- [UDPPacket](#) (int sock)  
*Create UDP packet from socket (blocks until data available)*
- [UDPPacket](#) (PoolPtr p)  
*Create from raw pool buffer (ADVANCED - rarely needed)*
- [UDPPacket](#) ([UDPPacket](#) &pkt)  
*Move constructor - transfers ownership.*
- [UDPPacket](#) (bool blslpv6=false)  
*Construct a new UDP packet object.*
- [~UDPPacket](#) ()  
*Destructor - automatically frees pool buffer.*
- bool [Receive](#) (OS\_FIFO \*pFifo, uint32\_t wait\_ticks)  
*Reuse packet object for multiple receives (efficiency optimization)*
- void [SetSourcePort](#) (uint16\_t port)  
*Set source port (0=auto-assign ephemeral port, 1024-65535=specific port)*
- uint16\_t [GetSourcePort](#) (void) const  
*Get source port from received packet or previously set value.*

- [MACADDR GetMacSource](#) ()  
*Get sender's MAC address (link-layer identification)*
- [IPADDR GetSourceAddress](#) (void)  
*Get source IP address of the packet.*
- [IPADDR GetDestinationAddress](#) (void)  
*Get destination IP address of the packet.*
- bool [bIsIPv6](#) () const  
*Check if packet uses IPv6 addressing.*
- void [SetDestinationPort](#) (uint16\_t port)  
*Set destination port (standard services: 53=DNS, 80=HTTP, 123=NTP, 161=SNMP)*
- uint16\_t [GetDestinationPort](#) (void) const  
*Get destination port (for received packets: local port that received it)*
- uint16\_t [GetPacketId](#) (void)  
*Get IP packet ID (used for fragment reassembly and packet tracking)*
- void [SetDSCP](#) (uint8\_t dscp)  
*Set DSCP for QoS (0=best effort, 46=VoIP/EF, 34=video/AF41, 8=bulk/CS1)*
- uint8\_t [GetDSCP](#) (void) const  
*Get DSCP value (0-63)*
- uint8\_t\* [GetDataBuffer](#) (bool bReAllocatelfNeeded=false)  
*Get data buffer pointer for reading or writing.*
- void [SetDataSize](#) (uint16\_t numBytes)  
*Set data size (REQUIRED after [GetDataBuffer\(\)](#) writes)*
- uint16\_t [GetDataSize](#) (void) const  
*Get current data size in bytes.*
- void [AddData](#) (uint8\_t\* pData, uint16\_t len)  
*Add binary data (auto-updates size - no [SetDataSize\(\)](#) needed)*
- void [AddData](#) (PCSTR pData)  
*Add null-terminated string (includes NULL in packet)*
- void [AddDataWord](#) (uint16\_t w)  
*Add 16-bit value.*
- void [AddDataByte](#) (uint8\_t b)  
*Add 8-bit value.*
- bool [Validate](#) (void)  
*Validate received packet (checks checksum and data presence)*
- void [ResetData](#) (void)  
*Reset data size to 0 (clears packet)*
- void [SendAndKeep](#) (const IPADDR &to, uint8\_t ttl=0)  
*Send copy, keep original (for sending to multiple destinations)*
- void [Send](#) (const IPADDR &to, uint8\_t ttl=0)  
*Send and free buffer (recommended - most efficient)*
- void [SendAndKeepViaInterfaceNum](#) (const IPADDR &to, int interface, uint8\_t ttl=0)  
*Send copy via specific interface, keep original.*
- void [SendViaInterfaceNum](#) (const IPADDR &to, int interface, uint8\_t ttl=0)  
*Send via specific interface and free buffer.*
- void [SendAndKeepViaIfAddr](#) (const IPADDR &to, const IPADDR &from\_ip, uint8\_t ttl=0)  
*Send a packet and retain a copy via a specific interface address.*
- void [SendViaIfAddr](#) (const IPADDR &to, const IPADDR &from\_ip, uint8\_t ttl=0)  
*Send a packet via a specific interface address.*

## 16.184.1 Detailed Description

UDP Packet Class - Complete UDP packet management.

Provides comprehensive UDP packet handling with automatic memory management, flexible data insertion methods, and advanced sending options.

### 16.184.1.1 Transmission Patterns

#### Method 1: `sprintf` with `GetDataBuffer()` - Best for formatted text

```
UDPPacket pkt;
pkt.SetSourcePort(123);
pkt.SetDestinationPort(456);
uint16_t len = sprintf(pkt.GetDataBuffer(), "Temp: %d", temp);
pkt.SetDataSize(len); // IMPORTANT: Must set size!
pkt.Send(destIP);
```

#### Method 2: `AddData()` - Best for binary/incremental data

```
UDPPacket pkt;
pkt.SetSourcePort(123);
pkt.SetDestinationPort(456);
pkt.AddData(header, sizeof(header));
pkt.AddDataWord(sensor);
pkt.Send(destIP);
```

### 16.184.1.2 Reception Pattern

```
OS_FIFO udpFifo;
OSFifoInit(&udpFifo);
RegisterUDPFifo(PORT, &udpFifo);

while (1) {
 UDPPacket pkt(&udpFifo, TICKS_PER_SECOND * 5);
 if (pkt.Validate()) {
 // Process pkt.GetDataBuffer() and pkt.GetDataSize()
 }
}
```

See also

`RegisterUDPFifo()`, `OS_FIFO`

## 16.184.2 Constructor & Destructor Documentation

### 16.184.2.1 UDPPacket() [1/5]

```
UDPPacket::UDPPacket (
 OS_FIFO * pFifo,
 TickTimeout timeout)
```

Create UDP packet from FIFO with timeout.

Constructs a `UDPPacket` by receiving data from a registered FIFO queue with a specified timeout. This constructor blocks until either a packet arrives or the timeout expires. The packet may be invalid even if the constructor returns, so **ALWAYS** call `Validate()` immediately after construction.

Parameters

|                |                                                                                                                                                                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>pFifo</i>   | Pointer to <code>OS_FIFO</code> queue registered with <code>RegisterUDPFifo()</code>                                                                                                                                                                                               |
| <i>timeout</i> | Timeout duration in system ticks: <ul style="list-style-type: none"> <li>0 = Non-blocking (return immediately if no packet)</li> <li>TICKS_PER_SECOND = 1 second timeout</li> <li>TICKS_PER_SECOND * N = N second timeout</li> <li>0xFFFFFFFF = Wait forever (blocking)</li> </ul> |

**Note**

**CRITICAL:** ALWAYS call [Validate\(\)](#) after construction before using packet data  
 Timeout expiration does not guarantee an invalid packet - always check with [Validate\(\)](#)  
 The FIFO must be created with [OSFifoInit\(\)](#) and registered with [RegisterUDPFifo\(\)](#)  
 This constructor blocks the calling task until packet arrives or timeout  
 For non-blocking behavior, use timeout=0 and check [Validate\(\)](#)

**Warning**

**NEVER** access packet data without calling [Validate\(\)](#) first!  
 A timeout does NOT mean the packet is invalid - must still validate  
 The FIFO must be registered before using this constructor

**Constructor vs Receive() Method:**

This constructor is equivalent to:

```
UDPPacket pkt;
pkt.Receive(pFifo, timeout);
```

However, using the [Receive\(\)](#) method in a loop is more efficient than constructing new packets repeatedly.

**FIFO Setup Requirements:**

Before using this constructor:

```
OS_FIFO udpFifo;
OSFifoInit(&udpFifo); // Initialize FIFO
RegisterUDPFifo(5000, &udpFifo); // Register on port 5000
```

**Expand for Example Usage****Examples****Example Usage - Basic Receive with Timeout:**

```
OS_FIFO udpFifo;
OSFifoInit(&udpFifo);
RegisterUDPFifo(5000, &udpFifo);

// Receive with 5 second timeout
UDPPacket pkt(&udpFifo, TICKS_PER_SECOND * 5);
if (pkt.Validate()) {
 printf("Got %d bytes from %s:%d\r\n",
 pkt.GetDataSize(),
 pkt.GetSourceAddress().ToString(),
 pkt.GetSourcePort());
 ProcessPacket(pkt);
} else {
 printf("No valid packet (timeout or corrupt)\r\n");
}
```

**Example Usage - Non-Blocking Check:**

```
// Check for packet without blocking
UDPPacket pkt(&udpFifo, 0); // timeout=0, immediate return
if (pkt.Validate()) {
 printf("Packet available!\r\n");
 ProcessPacket(pkt);
} else {
 // No packet available right now - continue other work
 DoOtherWork();
}
```

**Example Usage - Server Loop (INEFFICIENT):**

```
// INEFFICIENT - Creates new packet each iteration
while (running) {
 UDPPacket pkt(&udpFifo, TICKS_PER_SECOND); // Allocation overhead!
 if (pkt.Validate()) {
 ProcessPacket(pkt);
 }
}

// EFFICIENT - Reuse same packet object
UDPPacket pkt;
while (running) {
```

```

 if (pkt.Receive(&udpFifo, TICKS_PER_SECOND)) { // Better approach
 if (pkt.Validate()) {
 ProcessPacket(pkt);
 }
 }
 }
}

```

#### Example Usage - Heartbeat Monitoring:

```

OS_FIFO heartbeatFifo;
OSFifoInit(&heartbeatFifo);
RegisterUDPFifo(9000, &heartbeatFifo);

// Wait for heartbeat with 10 second timeout
UDPPacket hb(&heartbeatFifo, TICKS_PER_SECOND * 10);
if (hb.Validate()) {
 printf("Heartbeat received - connection alive\r\n");
 lastHeartbeat = Secs;
} else {
 printf("Heartbeat timeout - connection may be lost\r\n");
 HandleConnectionLost();
}

```

#### Example Usage - Request with Timeout:

```

// Send request
UDPPacket request;
request.SetDestinationPort(8080);
request.AddData("GET_STATUS", 10);
request.Send(serverIP);

// Wait for response with 3 second timeout
OS_FIFO responseFifo;
OSFifoInit(&responseFifo);
RegisterUDPFifo(0, &responseFifo); // Listen on any available port

UDPPacket response(&responseFifo, TICKS_PER_SECOND * 3);
if (response.Validate()) {
 printf("Response received within timeout\r\n");
 ProcessResponse(response);
} else {
 printf("Request timeout - server not responding\r\n");
 HandleTimeout();
}

```

| Scenario                | Use This Constructor? | Better Alternative                   |
|-------------------------|-----------------------|--------------------------------------|
| One-time receive        | Yes                   | N/A                                  |
| Loop with many receives | No                    | Use <a href="#">Receive()</a> method |
| Quick test/debug        | Yes                   | Convenient for testing               |
| Performance critical    | No                    | Reuse packet object                  |
| Multiple ports          | Maybe                 | Consider <a href="#">Receive()</a>   |

#### When to Use This Constructor:

#### Timeout Behavior:

- Constructor returns after timeout expires, even without packet
- [Validate\(\)](#) will return false if no packet received
- [Validate\(\)](#) may return false even if packet received (corrupted)
- Always check [Validate\(\)](#) - never assume timeout means invalid

#### See also

[Validate\(\)](#), [Receive\(\)](#), [RegisterUDPFifo\(\)](#), [OSFifoInit\(\)](#), [UDPPacket\(\)](#)



**16.184.2.2 UDPPacket()** [2/5]

```
UDPPacket::UDPPacket (
 int sock)
```

Create UDP packet from socket (blocks until data available)

Constructs a [UDPPacket](#) by receiving data from a UDP socket. This constructor blocks until data is available on the socket. **Best practice:** Use after [select\(\)](#) or [poll\(\)](#) to ensure data is ready before calling this constructor to avoid unexpected blocking.

**Parameters**

|             |                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------|
| <i>sock</i> | UDP socket descriptor from <a href="#">CreateRxUdpSocket()</a> or <a href="#">CreateRxTxUdpSocket()</a> |
|-------------|---------------------------------------------------------------------------------------------------------|

**Note**

**CRITICAL:** This constructor BLOCKS if no data is available

Use [select\(\)](#) or [poll\(\)](#) first to check if data is ready

Always call [Validate\(\)](#) after construction to verify packet integrity

The socket must be a valid UDP socket (SOCK\_DGRAM)

For non-blocking behavior, set socket to non-blocking mode first

**Warning**

Will BLOCK indefinitely if no data arrives and socket is blocking

Always use with [select\(\)/poll\(\)](#) or set socket to non-blocking mode

The socket must remain valid - don't close it while packet exists

**Recommended Pattern:**

Always use [select\(\)](#) before this constructor to avoid blocking:

1. Use [select\(\)](#) to check if data is ready
2. If data ready, construct [UDPPacket](#) (won't block)
3. Call [Validate\(\)](#) to verify packet
4. Process packet data

**Socket Creation:**

Before using this constructor, create a UDP socket:

```
// Receive-only socket
int sock = CreateRxUdpSocket(1234); // Listen on port 1234

// Or receive/transmit socket
int sock = CreateRxTxUdpSocket(1234); // Port 1234
```

**Expand for Example Usage****Examples****Example Usage - Basic with select():**

```
int sock = CreateRxUdpSocket(1234);

// Setup select() to wait for data
fd_set readfds;
FD_ZERO(&readfds);
FD_SET(sock, &readfds);

if (select(sock + 1, &readfds, NULL, NULL, NULL) > 0) {
 // Data is ready - constructor won't block
 UDPPacket pkt(sock);
 if (pkt.Validate()) {
 printf("Received packet from %s:%u\r\n",
 pkt.GetSourceAddress().ToString(),
 pkt.GetSourcePort());
 ProcessPacket(pkt);
 }
}
```

**Example Usage - With Timeout:**

```

int sock = CreateRxUdpSocket(5000);

fd_set readfds;
FD_ZERO(&readfds);
FD_SET(sock, &readfds);

// Wait up to 5 seconds for data
struct timeval tv;
tv.tv_sec = 5;
tv.tv_usec = 0;

int result = select(sock + 1, &readfds, NULL, NULL, &tv);
if (result > 0) {
 UDPPacket pkt(sock); // Data ready - won't block
 if (pkt.Validate()) {
 ProcessPacket(pkt);
 }
} else if (result == 0) {
 printf("Timeout - no data received\r\n");
} else {
 printf("Select error\r\n");
}

```

**Example Usage - Multiple Sockets:**

```

int controlSock = CreateRxUdpSocket(5000);
int dataSock = CreateRxUdpSocket(5001);

fd_set readfds;
FD_ZERO(&readfds);
FD_SET(controlSock, &readfds);
FD_SET(dataSock, &readfds);

int maxfd = (controlSock > dataSock) ? controlSock : dataSock;

if (select(maxfd + 1, &readfds, NULL, NULL, NULL) > 0) {
 if (FD_ISSET(controlSock, &readfds)) {
 UDPPacket ctrlPkt(controlSock);
 if (ctrlPkt.Validate()) {
 ProcessControlPacket(ctrlPkt);
 }
 }

 if (FD_ISSET(dataSock, &readfds)) {
 UDPPacket dataPkt(dataSock);
 if (dataPkt.Validate()) {
 ProcessDataPacket(dataPkt);
 }
 }
}

```

**Example Usage - Server Loop:**

```

int sock = CreateRxUdpSocket(8080);

while (running) {
 fd_set fds;
 FD_ZERO(&fds);
 FD_SET(sock, &fds);

 struct timeval tv = {1, 0}; // 1 second timeout

 if (select(sock + 1, &fds, NULL, NULL, &tv) > 0) {
 UDPPacket pkt(sock);
 if (pkt.Validate()) {
 IPADDR clientAddr = pkt.GetSourceAddress();
 uint16_t clientPort = pkt.GetSourcePort();

 printf("Request from %s:%u\r\n",
 clientAddr.ToString(), clientPort);

 // Send response
 UDPPacket response;
 response.SetDestinationPort(clientPort);
 response.AddData("ACK", 3);
 response.Send(clientAddr);
 }
 }

 // Timeout - check for shutdown, do housekeeping, etc.
}

```

**Example Usage - Non-Blocking Socket:**

```

int sock = CreateRxUdpSocket(6000);

// Set socket to non-blocking mode
int flags = fcntl(sock, F_GETFL, 0);
fcntl(sock, F_SETFL, flags | O_NONBLOCK);

// Now constructor won't block - returns immediately
UDPPacket pkt(sock);
if (pkt.Validate()) {
 printf("Packet received\r\n");
 ProcessPacket(pkt);
} else {
 // Either no data or invalid packet
 printf("No valid packet available\r\n");
}

```

| Approach                         | Pros                                                       | Cons                                    |
|----------------------------------|------------------------------------------------------------|-----------------------------------------|
| <b>Socket</b> (this)             | Works with <a href="#">select()</a> , standard BSD sockets | Requires <a href="#">select()</a> setup |
| <b>FIFO</b> (other constructors) | Simple, RTOS-native, efficient                             | NetBurner-specific API                  |

**Socket vs FIFO Approach:****When to Use Sockets:**

- Integrating with [select\(\)](#)-based event loops
- Porting BSD socket code to NetBurner
- Multiplexing multiple UDP ports
- Need timeout with standard timeval
- Compatibility with POSIX socket APIs

**When to Use FIFOs:**

- Pure RTOS design (no select needed)
- Higher performance (fewer layers)
- NetBurner-native applications
- Integration with other OS\_FIFO queues

**Common Mistakes:**

```

// WRONG - Will block forever if no data!
int sock = CreateRxUdpSocket(1234);
UDPPacket pkt(sock); // BLOCKS until packet arrives

// CORRECT - Check with select() first
int sock = CreateRxUdpSocket(1234);
fd_set fds;
FD_ZERO(&fds);
FD_SET(sock, &fds);
if (select(sock + 1, &fds, NULL, NULL, NULL) > 0) {
 UDPPacket pkt(sock); // Won't block - data ready
}

// ALSO CORRECT - Non-blocking socket
int sock = CreateRxUdpSocket(1234);
fcntl(sock, F_SETFL, O_NONBLOCK);
UDPPacket pkt(sock); // Returns immediately

```

**See also**

[Validate\(\)](#), [CreateRxUdpSocket\(\)](#), [CreateRxTxUdpSocket\(\)](#), [select\(\)](#), [UDPPacket\(OS\\_FIFO\\*, uint32\\_t\)](#)

**16.184.2.3 UDPPacket()** [3/5]

```
UDPPacket::UDPPacket (
 PoolPtr p)
```

Create from raw pool buffer (ADVANCED - rarely needed)

Constructs a [UDPPacket](#) by wrapping an existing pool buffer pointer. This is an advanced constructor for custom protocol implementations that work directly with the NetBurner pool buffer system. The [UDPPacket](#) takes ownership of the buffer and will automatically free it when destroyed.

**Parameters**

|          |                                                      |
|----------|------------------------------------------------------|
| <i>p</i> | Pool buffer pointer (PoolPtr) containing packet data |
|----------|------------------------------------------------------|

**Warning**

**DANGEROUS** - Incorrect use causes memory corruption, double-frees, or crashes

Only use if you fully understand the NetBurner pool buffer allocation system

The pool buffer must be properly allocated and formatted

[UDPPacket](#) assumes FULL OWNERSHIP - do not free the buffer elsewhere

After passing to [UDPPacket](#), the original PoolPtr should not be used

**Note**

This constructor is for EXPERT USE ONLY

Most applications should use [UDPPacket\(\)](#) or [UDPPacket\(bool\)](#) instead

Required for custom protocol stacks or low-level network code

The buffer must contain valid UDP/IP headers in the correct format

**When This Constructor Is Needed:**

- Implementing custom network protocols below UDP level
- Interfacing with raw network drivers
- Zero-copy optimizations in performance-critical code
- Custom packet inspection/modification tools
- Working with network hardware at buffer level

**Prerequisites:**

Before using this constructor, you must understand:

- NetBurner pool buffer allocation (GetBuffer/FreeBuffer)
- UDP/IP packet header structures
- Buffer ownership and lifecycle management
- Memory alignment requirements
- Reference counting semantics

**Buffer Ownership Rules:**

```
PoolPtr buf = AllocatePoolBuffer(); // You own buffer
// ... format buffer with UDP/IP headers ...

UDPPacket pkt(buf); // UDPPacket now owns buffer
// DO NOT use 'buf' anymore - ownership transferred!
// DO NOT call FreeBuffer(buf) - UDPPacket will do it

// When pkt goes out of scope or is destroyed, buffer is freed
```

**Expand for Example Usage****Examples**

**Example Usage - Custom Protocol (Expert Level):**

```
// ADVANCED EXAMPLE - Only for experts!
PoolPtr CreateCustomUDPPacket(const uint8_t* data, uint16_t len) {
 // Allocate buffer from pool
 PoolPtr buf = GetBuffer();
 if (!buf) {
 return nullptr;
 }

 // Manually construct UDP/IP headers
 uint8_t* pData = (uint8_t*)buf->pData;

 // IP header setup (simplified - actual code more complex)
 struct IPHeader* ipHdr = (struct IPHeader*)pData;
 ipHdr->version = 4;
 ipHdr->protocol = UDP_PROTOCOL;
 // ... more header fields ...

 // UDP header setup
 struct UDPHeader* udpHdr = (struct UDPHeader*)(pData + IP_HEADER_SIZE);
 udpHdr->srcPort = htons(5000);
 udpHdr->dstPort = htons(6000);
 udpHdr->length = htons(len + UDP_HEADER_SIZE);
 // ... checksum calculation ...

 // Copy payload
 memcpy(pData + IP_HEADER_SIZE + UDP_HEADER_SIZE, data, len);

 return buf; // Caller takes ownership
}

void SendCustomPacket() {
 uint8_t payload[] = "Custom data";
 PoolPtr buf = CreateCustomUDPPacket(payload, sizeof(payload));

 if (buf) {
 UDPPacket pkt(buf); // Wrap buffer - takes ownership
 // pkt will auto-free buffer when destroyed

 // Can still use normal UDPPacket methods
 printf("Packet size: %u\r\n", pkt.GetDataSize());
 }
}
```

**Example Usage - Raw Packet Inspection:**

```
// Intercept packets at low level for inspection
void InspectRawPacket(PoolPtr rawBuffer) {
 // Create temporary packet wrapper for inspection
 // Note: This takes ownership of rawBuffer!
 UDPPacket pkt(rawBuffer);

 printf("Source: %s:%u\r\n",
 pkt.GetSourceAddress().ToString(),
 pkt.GetSourcePort());
 printf("Destination: %s:%u\r\n",
 pkt.GetDestinationAddress().ToString(),
 pkt.GetDestinationPort());
 printf("Data size: %u bytes\r\n", pkt.GetDataSize());

 // Buffer will be freed when pkt goes out of scope
}
```

**Common Mistakes to Avoid:**

```
// WRONG - Double free!
PoolPtr buf = GetBuffer();
UDPPacket pkt(buf);
FreeBuffer(buf); // CRASH - pkt already owns it!

// WRONG - Using buffer after transfer
PoolPtr buf = GetBuffer();
UDPPacket pkt(buf);
buf->pData[0] = 0xFF; // UNDEFINED - pkt owns buffer now!

// WRONG - Uninitialized buffer
PoolPtr buf = GetBuffer();
UDPPacket pkt(buf); // CRASH - no valid UDP/IP headers!

// CORRECT - Proper ownership transfer
PoolPtr buf = CreateValidPacketBuffer();
UDPPacket pkt(buf); // Clean transfer
// Don't touch buf anymore
```

For Most Applications:

**You should NOT use this constructor.** Use these instead:

- `UDPPacket()` - Standard packet creation
- `UDPPacket(bool)` - IPv4/IPv6 packet creation

See also

`UDPPacket()`, `UDPPacket(bool)`, `PoolPtr`, `GetBuffer()`, `FreeBuffer()`

#### 16.184.2.4 UDPPacket() [4/5]

```
UDPPacket::UDPPacket (
 UDPPacket & pkt)
```

Move constructor - transfers ownership.

Constructs a new `UDPPacket` by moving resources from an existing packet. The source packet transfers ownership of its internal buffers and state to the new packet. After the move, the source packet becomes invalid and should not be used.

Parameters

|            |                                                  |
|------------|--------------------------------------------------|
| <i>pkt</i> | Source packet to move from (will be invalidated) |
|------------|--------------------------------------------------|

Note

Source packet becomes INVALID after move - do not use it!

This is a move operation, not a copy - no data duplication occurs

Useful for returning packets from functions efficiently

No memory allocation or copying happens - just ownership transfer

The source packet should be considered destroyed after the move

Move Semantics:

After construction:

- **New packet:** Has all data and state from source
- **Source packet:** Empty/invalid state, should not be accessed

Typical Use Cases:

- Returning `UDPPacket` from a function without copying
- Transferring packet ownership between objects
- Building packets in helper functions
- Factory pattern implementations
- Avoiding unnecessary buffer copies

---

#### Expand for Example Usage

##### Examples

Example Usage - Return from Function:

```
// Build a packet and return it efficiently (no copy)
UDPPacket CreateStatusPacket(uint32_t statusCode) {
 UDPPacket pkt;
 pkt.SetDestinationPort(8080);
 pkt.AddDataWord(statusCode);
 pkt.AddData("Status update", 13);

 return pkt; // Move constructor called automatically
}

void SendStatus() {
 UDPPacket statusPkt = CreateStatusPacket(200); // Moved, not copied
 statusPkt.Send(serverIP);
}
```

**Example Usage - Packet Factory Pattern:**

```

class PacketBuilder {
public:
 // Returns packet by move - efficient
 UDPPacket BuildHeartbeat(uint32_t sequence) {
 UDPPacket pkt;
 pkt.SetDestinationPort(5000);
 pkt.AddDataByte(0x01); // Heartbeat type
 pkt.AddDataWord(sequence);
 pkt.AddDataWord(Secs); // Timestamp
 return pkt; // Move constructor invoked
 }

 UDPPacket BuildDataPacket(const uint8_t* data, uint16_t len) {
 UDPPacket pkt;
 pkt.SetDestinationPort(5001);
 pkt.AddDataByte(0x02); // Data type
 pkt.AddData((uint8_t*)data, len);
 return pkt; // Move constructor invoked
 }
};

void SendPackets() {
 PacketBuilder builder;

 UDPPacket hb = builder.BuildHeartbeat(42); // Moved
 hb.Send(serverIP);

 uint8_t data[] = {1, 2, 3, 4};
 UDPPacket dp = builder.BuildDataPacket(data, 4); // Moved
 dp.Send(serverIP);
}

```

**Example Usage - Conditional Packet Building:**

```

UDPPacket PreparePacket(bool urgent) {
 UDPPacket pkt;

 if (urgent) {
 pkt.SetDSCP(46); // Expedited Forwarding
 pkt.SetDestinationPort(9000);
 pkt.AddData("URGENT", 6);
 } else {
 pkt.SetDSCP(0); // Best effort
 pkt.SetDestinationPort(9001);
 pkt.AddData("NORMAL", 6);
 }

 return pkt; // Move constructor - no copy overhead
}

void ProcessAlert(bool isUrgent) {
 UDPPacket alertPkt = PreparePacket(isUrgent); // Efficient move
 alertPkt.Send(monitorIP);
}

```

**Example Usage - Packet Queue Transfer:**

```

// Transfer packets between queues efficiently
void MovePacketToQueue(UDPPacket& sourcePkt, std::vector<UDPPacket>& queue) {
 UDPPacket movedPkt(sourcePkt); // Move constructor
 queue.push_back(movedPkt);

 // sourcePkt is now invalid - don't use it!
}

```

**Performance Benefits:**

```

// WITHOUT move constructor (hypothetical copy)
UDPPacket CreatePacket() {
 UDPPacket pkt;
 pkt.AddData(largeBuffer, 1024);
 return pkt; // Would copy 1024+ bytes!
}

// WITH move constructor (actual behavior)
UDPPacket CreatePacket() {
 UDPPacket pkt;
 pkt.AddData(largeBuffer, 1024);
 return pkt; // Just transfers pointer - fast!
}

```

**What Happens During Move:**

1. New packet receives all internal pointers from source
2. New packet receives all state (ports, addresses, data size)
3. Source packet's pointers are set to NULL/invalid
4. Source packet should not be accessed after move
5. No memory copying or allocation occurs

**Warning**

**CRITICAL:** After move, the source packet is INVALID

```
UDPPacket pkt1;
pkt1.AddData("Hello", 5);

UDPPacket pkt2(pkt1); // Move from pkt1 to pkt2

// SAFE - pkt2 has the data
pkt2.Send(serverIP);

// UNSAFE - pkt1 is now invalid!
pkt1.Send(serverIP); // UNDEFINED BEHAVIOR!
```

**Compiler Optimization:**

Modern compilers often use Return Value Optimization (RVO) or Named Return Value Optimization (NRVO) to eliminate even the move constructor call, constructing the return value directly in the caller's location. The move constructor exists as a fallback when such optimizations can't be applied.

**See also**

UDPPacket(), UDPPacket(bool), Send(), AddData()

**16.184.2.5 UDPPacket() [5/5]**

```
UDPPacket::UDPPacket (
 bool bIsIpv6 = false)
```

Construct a new UDP packet object.

Creates a UDP packet object that can be used for sending or receiving UDP data. The packet is allocated with appropriate buffer space and initialized for either IPv4 or IPv6 operation.

**Parameters**

|                |                                                                                                                                                                       |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>bIsIpv6</i> | IP version selector: <ul style="list-style-type: none"> <li>• false = IPv4 packet (default, most common)</li> <li>• true = IPv6 packet (128-bit addresses)</li> </ul> |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Note**

The default is IPv4 (bIsIpv6=false) which is appropriate for most applications

For received packets, the IP version is automatically detected regardless of constructor parameter

The same UDPPacket object can be reused for multiple send/receive operations

Memory is allocated during construction - consider reusing packets in loops



**When to Use IPv6:**

- Your network infrastructure requires IPv6 addressing
- Communicating with IPv6-only hosts
- Future-proofing for IPv6 transition
- Link-local communication using fe80:: addresses

| Feature        | IPv4 (false)        | IPv6 (true)               |
|----------------|---------------------|---------------------------|
| Address Size   | 32-bit (4 bytes)    | 128-bit (16 bytes)        |
| Address Format | 192.168.1.100       | fe80::1234:5678:abcd:ef01 |
| Header Size    | 20 bytes minimum    | 40 bytes minimum          |
| Typical Use    | Most networks today | Modern/future networks    |

**IPv4 vs IPv6 Considerations:****Expand for Example Usage****Examples****Example Usage - Basic IPv4 Packet (Default):**

```
// Create IPv4 packet (most common case)
UDPPacket pkt; // IPv4 by default
pkt.SetDestinationPort(8080);
pkt.AddData("Hello", 5);
pkt.Send(serverIP);
```

**Example Usage - Explicit IPv4:**

```
// Explicitly specify IPv4
UDPPacket pkt(false);
pkt.SetSourcePort(5000);

while (running) {
 if (pkt.Receive(&fifo, TICKS_PER_SECOND)) {
 if (pkt.Validate()) {
 ProcessIPv4Packet(pkt);
 }
 }
}
```

**Example Usage - IPv6 Packet:**

```
// Create IPv6 packet for modern networks
UDPPacket pkt(true); // IPv6 mode

// IPv6 address example
IPADDR ipv6Addr;
ipv6Addr.SetFromAscii("fe80::1234:5678:abcd:ef01");

pkt.SetDestinationPort(9000);
pkt.AddData("IPv6 Data", 9);
pkt.Send(ipv6Addr);
```

**Example Usage - Dual-Stack Application:**

```
// Support both IPv4 and IPv6
void ProcessPackets(OS_FIFO *fifo, bool useIPv6) {
 UDPPacket pkt(useIPv6); // Construct with appropriate version

 while (running) {
 if (pkt.Receive(fifo, TICKS_PER_SECOND)) {
 if (pkt.Validate()) {
 // Check actual packet type
 if (pkt.bIsIPv6()) {
 printf("IPv6 packet received\r\n");
 } else {
 printf("IPv4 packet received\r\n");
 }
 ProcessPacket(pkt);
 }
 }
 }
}
```

```

 }

 // In main application
 if (networkSupportsIPv6()) {
 ProcessPackets(&fifo, true); // IPv6
 } else {
 ProcessPackets(&fifo, false); // IPv4
 }
}

```

#### Example Usage - Efficient Packet Reuse:

```

// Create packet once, reuse many times (recommended pattern)
UDPPacket pkt; // IPv4 packet

// Send multiple messages efficiently
for (int i = 0; i < 100; i++) {
 pkt.ResetData(); // Clear previous data
 pkt.AddDataWord(i);
 pkt.AddData("Message", 7);
 pkt.Send(destIP, destPort);
 OSTimeDly(TICKS_PER_SECOND);
}

// Same packet object can also receive
while (running) {
 if (pkt.Receive(&fifo, TICKS_PER_SECOND * 5)) {
 if (pkt.Validate()) {
 ProcessResponse(pkt);
 }
 }
}

```

#### Performance Tip:

For high-performance applications, create the [UDPPacket](#) object once outside your main loop and reuse it:

```

// INEFFICIENT - Creates/destroys packet each iteration
while (running) {
 UDPPacket pkt; // Allocation overhead!
 if (pkt.Receive(&fifo, timeout)) {
 // process
 }
}

// EFFICIENT - Create once, reuse many times
UDPPacket pkt; // Allocated once
while (running) {
 if (pkt.Receive(&fifo, timeout)) {
 // process (much faster!)
 }
}

```

#### Warning

For received packets, the IP version is determined by the incoming packet, not by the constructor parameter. Use [blsIPv6\(\)](#) to check received packet type.

#### See also

[blsIPv6\(\)](#), [Receive\(\)](#), [Send\(\)](#), [ResetData\(\)](#)

#### 16.184.2.6 ~UDPPacket()

```
UDPPacket::~UDPPacket ()
```

Destructor - automatically frees pool buffer.

Destructor that automatically releases the UDP packet's pool buffer back to the system memory pool. This implements RAII (Resource Acquisition Is Initialization) principles, ensuring proper cleanup without manual memory management.

#### Note

**Automatic cleanup** - Called automatically when packet goes out of scope

**No manual calls needed** - Never call this destructor directly

**Exception-safe** - Cleanup happens even if exceptions occur

**Pool buffer release** - Returns buffer to pool for reuse

**Warning**

If `Send()` was already called, the buffer is already freed - destructor handles this  
 Do not access packet data after the `UDPPacket` object is destroyed  
 Buffer is NOT freed if `SendAndKeep()` was the last send operation

**RAII Benefits:**

Resource Acquisition Is Initialization provides automatic memory management:

- **Allocation:** Constructor allocates pool buffer
- **Usage:** Packet methods use the buffer
- **Cleanup:** Destructor automatically frees buffer
- **Safety:** No memory leaks even with exceptions

**When Destructor Runs:**

The destructor is automatically called when:

- `UDPPacket` goes out of scope
- `UDPPacket` is explicitly deleted
- Function returns (for local variables)
- Exception is thrown (stack unwinding)
- Container is destroyed (if packet in vector, etc.)

**Memory Management Cases:**

The destructor handles different memory states correctly:

**Case 1: Buffer never sent**

```
{
 UDPPacket pkt;
 pkt.AddData("Hello", 5);
 // Destructor frees buffer when pkt goes out of scope
}
```

**Case 2: Buffer sent with `Send()`**

```
{
 UDPPacket pkt;
 pkt.AddData("Hello", 5);
 pkt.Send(destIP); // Buffer already freed here
 // Destructor sees buffer is NULL, does nothing
}
```

**Case 3: Buffer sent with `SendAndKeep()`**

```
{
 UDPPacket pkt;
 pkt.AddData("Hello", 5);
 pkt.SendAndKeep(dest1); // Buffer retained
 pkt.SendAndKeep(dest2); // Buffer still retained
 // Destructor frees buffer when pkt goes out of scope
}
```

**Case 4: Exception safety**

```
try {
 UDPPacket pkt;
 pkt.AddData("Hello", 5);
 ThrowException(); // Exception thrown
 // Destructor still called during stack unwinding - no leak
} catch (...) {
 // Buffer already cleaned up by destructor
}
```

**Performance Considerations:**

- **Fast operation:** Simply returns buffer to pool
- **No system calls:** Pure memory pool operation
- **O(1) complexity:** Constant time release
- **Thread-safe:** Pool operations are synchronized

**Expand for Example Usage****Examples**

**Common Patterns:**

```

// Pattern 1: Local scope (most common)
void SendMessage() {
 UDPPacket pkt; // Constructor allocates
 pkt.AddData("Message", 7);
 pkt.Send(serverIP);
} // Destructor called here - automatic cleanup

// Pattern 2: Early cleanup with explicit scope
void ProcessData() {
 {
 UDPPacket pkt;
 pkt.Receive(&fifo, timeout);
 ProcessPacket(pkt);
 } // Buffer freed here, before other work

 DoOtherWork(); // More memory available
}

// Pattern 3: Container of packets
void ManagePackets() {
 std::vector<UDPPacket> packets;
 packets.emplace_back(); // Constructor called
 packets.clear(); // Destructors called for all
}

// Pattern 4: Dynamic allocation (less common)
void DynamicPacket() {
 UDPPacket* pkt = new UDPPacket();
 pkt->AddData("Hello", 5);
 delete pkt; // Explicit destructor call
}

```

**Destructor vs Send():**

Understanding when memory is freed:

| Operation      | Buffer State After | Destructor Action    |
|----------------|--------------------|----------------------|
| Send()         | Freed immediately  | None (already freed) |
| SendAndKeep()  | Retained           | Frees on destruction |
| No send        | Allocated          | Frees on destruction |
| Receive() only | Allocated          | Frees on destruction |

**Fragment Buffer Handling:**

```

#ifdef UDP_FRAGMENTS
// For fragmented packets, destructor calls FragFreeBuffer()
// instead of normal FreeBuffer() - handles reassembly cleanup
#endif

```

**Internal Implementation:**

The destructor performs these steps:

1. Check if pool buffer pointer (m\_p) is non-NULL
2. If fragment buffer, call FragFreeBuffer()
3. Otherwise, call FreeBuffer()
4. Set internal pointers to NULL
5. IPv6 cleanup if compiled with IPv6 support

**Best Practices:**

- **Let it work automatically** - Don't call destructor manually
- **Use local variables** - Simplest and safest pattern
- **Minimize lifetime** - Use explicit scopes for early cleanup
- **Avoid new/delete** - Prefer stack allocation
- **Trust RAII** - Don't second-guess automatic cleanup

**What NOT To Do:**

```
// WRONG: Explicit destructor call
UDPPacket pkt;
pkt.~UDPPacket(); // NEVER DO THIS!

// WRONG: Accessing after scope
UDPPacket* ptr;
{
 UDPPacket pkt;
 ptr = &pkt;
} // pkt destroyed here
ptr->Send(ip); // CRASH: Accessing destroyed object

// WRONG: Double-delete
UDPPacket* pkt = new UDPPacket();
delete pkt;
delete pkt; // CRASH: Double-free
```

**Debug Information:**

In debug builds, the destructor may:

- Log destructor calls
- Validate buffer state
- Check for memory corruption
- Track pool statistics

**Warning**

**Critical:** Never manually call the destructor unless you know exactly what you're doing

The destructor is called automatically - manual calls cause double-free bugs

**See also**

UDPPacket(), [Send\(\)](#), [SendAndKeep\(\)](#), [FreeBuffer\(\)](#), PoolPtr

**16.184.3 Member Function Documentation****16.184.3.1 AddData() [1/2]**

```
void UDPPacket::AddData (
 PCSTR pData)
```

Add null-terminated string (includes NULL in packet)

Appends a null-terminated string to the packet, including the terminating NULL character. This is convenient for string-based protocols where the receiver expects NULL-terminated strings.

**Parameters**

|              |                                          |
|--------------|------------------------------------------|
| <i>pData</i> | Pointer to null-terminated string to add |
|--------------|------------------------------------------|

**Note**

The NULL terminator IS included in the packet data

String length is automatically calculated using strlen()

Data size is incremented by (strlen(pData) + 1)

**Warning**

pData must be a valid null-terminated string

For binary data that may contain NULL bytes, use [AddData\(puint8\\_t, uint16\\_t\)](#)

**Example Usage:**

```
UDPPacket pkt;
pkt.AddData("HELLO"); // Adds 6 bytes: 'H','E','L','L','O','\0'
pkt.AddData("WORLD"); // Adds 6 bytes: 'W','O','R','L','D','\0'
// Packet now contains: "HELLO\0WORLD\0" (12 bytes total)
```

**See also**

[AddData\(puint8\\_t, uint16\\_t\)](#), [GetDataSize\(\)](#)

**16.184.3.2 AddData() [2/2]**

```
void UDPPacket::AddData (
 puint8_t pData,
 uint16_t len)
```

Add binary data (auto-updates size - no [SetDataSize\(\)](#) needed)

Appends binary data to the packet buffer and automatically updates the internal data size counter. This is the primary method for building packets with structured data.

**Parameters**

|              |                                    |
|--------------|------------------------------------|
| <i>pData</i> | Pointer to the binary data to add  |
| <i>len</i>   | Number of bytes to copy from pData |

**Note**

The data size is automatically incremented by len

No manual call to [SetDataSize\(\)](#) is required

Data is appended to the end of existing packet data

**Warning**

Ensure pData points to at least len valid bytes

No bounds checking - caller must ensure packet has space

**Example Usage:**

```
struct SensorData {
 uint32_t timestamp;
 float temperature;
 float humidity;
} data;

UDPPacket pkt;
uint8_t header[] = {0xAA, 0xBB, 0x01};
pkt.AddData(header, sizeof(header));
pkt.AddData((puint8_t)&data, sizeof(data));
pkt.Send(destIP, destPort);
```

**See also**

[AddData\(PCSTR\)](#), [AddDataWord\(\)](#), [AddDataByte\(\)](#), [GetDataSize\(\)](#)

**16.184.3.3 AddDataByte()**

```
void UDPPacket::AddDataByte (
 uint8_t b)
```

Add 8-bit value.

Appends a single byte to the packet. Useful for adding flags, command codes, or single-byte values to packet data.

## Parameters

|          |                            |
|----------|----------------------------|
| <i>b</i> | 8-bit value to add (0-255) |
|----------|----------------------------|

## Note

Increments data size by 1 byte

No byte order conversion needed (single byte)

## Example Usage:

```
UDPPacket pkt;
pkt.AddDataByte(0x01); // Command code
pkt.AddDataByte(0xFF); // Flag value
pkt.AddDataByte(42); // Numeric value
```

## See also

[AddDataWord\(\)](#), [AddData\(puint8\\_t, uint16\\_t\)](#)

**16.184.3.4 AddDataWord()**

```
void UDPPacket::AddDataWord (
 uint16_t w)
```

Add 16-bit value.

Appends a 16-bit word to the packet in network byte order (big-endian). Useful for adding integer values, ports, or length fields to packets.

## Parameters

|          |                               |
|----------|-------------------------------|
| <i>w</i> | 16-bit value to add (0-65535) |
|----------|-------------------------------|

## Note

Value is automatically converted to network byte order if needed

Increments data size by 2 bytes

Equivalent to `AddData((puint8_t)&w, 2)` with byte order conversion

## Example Usage:

```
UDPPacket pkt;
pkt.AddDataWord(0x1234); // Adds 2 bytes: 0x12, 0x34 (big-endian)
pkt.AddDataWord(8080); // Adds port number
```

## See also

[AddDataByte\(\)](#), [AddData\(puint8\\_t, uint16\\_t\)](#)

**16.184.3.5 bIsIPv6()**

```
bool UDPPacket::bIsIPv6 () const [inline]
```

Check if packet uses IPv6 addressing.

Determines whether this packet is using IPv6 or IPv4 addressing by checking if the internal IPv6 header pointer is set. Returns true for IPv6 packets, false for IPv4 packets.

## Returns

true if packet is IPv6, false if packet is IPv4

**Note**

This is a const function and does not modify the packet

The check is performed by testing if m\_pIPv6f (IPv6 header pointer) is NULL

Use this before making IP version-specific operations

**Why Check IP Version?**

- IPv6 addresses are 128-bit vs IPv4's 32-bit
- Some protocols behave differently on IPv6
- MTU and fragmentation differ between versions
- Useful for logging and debugging
- May need different processing for IPv4 vs IPv6

**Expand for Example Usage****Examples**

Example Usage:

```
UDPPacket pkt;
if (pkt.Receive()) {
 IPADDR source = pkt.GetSourceAddress();

 if (pkt.bIsIPv6()) {
 printf("IPv6 packet from: %s\r\n", source.ToString());
 ProcessIPv6Specific(pkt);
 } else {
 printf("IPv4 packet from: %s\r\n", source.ToString());
 ProcessIPv4Specific(pkt);
 }
}

// Conditional packet handling
if (!pkt.bIsIPv6() && requireIPv6Only) {
 printf("Rejecting IPv4 packet on IPv6-only interface\r\n");
 return;
}
```

**See also**

[GetSourceAddress\(\)](#), [GetDestinationAddress\(\)](#), [IPADDR::IsIPv6\(\)](#)

**16.184.3.6 GetDataBuffer()**

```
uint8_t UDPPacket::GetDataBuffer (
 bool bReAllocateIfNeeded = false)
```

Get data buffer pointer for reading or writing.

**CRITICAL: After writing, MUST call [SetDataSize\(\)](#) before sending!**

```
// Writing (sprintf method - recommended for text):
uint16_t len = sprintf(pkt.GetDataBuffer(), "Value: %d", val);
pkt.SetDataSize(len); // REQUIRED!
```

```
// Reading (received packets):
uint8_t data = pkt.GetDataBuffer();
uint16_t len = pkt.GetDataSize();
for (int i = 0; i < len; i++) { process(data[i]); }
```

**16.184.3.7 GetDataSize()**

```
uint16_t UDPPacket::GetDataSize (
 void) const
```

Get current data size in bytes.

Returns the number of bytes currently stored in the packet's data buffer. This reflects the actual data length, not the buffer capacity.

**Returns**

Current data size in bytes (0 to maximum packet size)



**Note**

This is a const function and does not modify the packet state

The returned size includes all data added via [AddData\(\)](#) methods

**Example Usage:**

```
UDPPacket packet;
packet.AddData("Test", 4);
uint16_t size = packet.GetDataSize(); // Returns 4
```

**See also**

[ResetData\(\)](#), [AddData\(\)](#)

**16.184.3.8 GetDestinationAddress()**

```
IPADDR UDPPacket::GetDestinationAddress (
 void) [inline]
```

Get destination IP address of the packet.

Returns the destination IP address from the packet header. For received packets, this is YOUR local interface address that received the packet (useful for multi-homed devices). For outgoing packets, this is the target remote address.

**Returns**

IPADDR object containing the destination address (IPv4 or IPv6)

**Note**

This function wraps [GetDestinationAddress6\(\)](#) for unified IPv4/IPv6 handling

For received packets: returns YOUR local IP that received it

For sent packets: returns the remote target IP you're sending to

Useful for determining which interface received a packet

**Understanding Address Roles:****For Received Packets:**

- [GetSourceAddress\(\)](#) = Remote sender's IP
- [GetDestinationAddress\(\)](#) = Your local IP

**For Sent Packets:**

- [GetSourceAddress\(\)](#) = Your local IP
- [GetDestinationAddress\(\)](#) = Remote target's IP

**Example Usage:**

```
UDPPacket pkt;
if (pkt.Receive()) {
 IPADDR myAddr = pkt.GetDestinationAddress();
 IPADDR senderAddr = pkt.GetSourceAddress();

 // Useful for multi-homed devices
 if (myAddr == primaryInterface) {
 ProcessPrimaryNetwork(pkt);
 } else if (myAddr == managementInterface) {
 ProcessManagementNetwork(pkt);
 }

 printf("Packet from %s to my interface %s\r\n",
 senderAddr.ToString(),
 myAddr.ToString());
}
```

**See also**

[GetSourceAddress\(\)](#), [GetDestinationPort\(\)](#), [GetDestinationAddress6\(\)](#)

### 16.184.3.9 GetDestinationPort()

```
uint16_t UDPPacket::GetDestinationPort (
 void) const
```

Get destination port (for received packets: local port that received it)

For outgoing packets, returns the destination port set by [SetDestinationPort\(\)](#). For received packets, returns the LOCAL port that received the packet (i.e., which port on YOUR device the packet was addressed to).

#### Returns

Port number (1-65535)

#### Note

For received packets, this is YOUR listening port, not the sender's port

To get the sender's port on received packets, use [GetSourcePort\(\)](#)

This is a const function and does not modify the packet

#### Understanding Port Roles in Received Packets:

When you receive a UDP packet:

- [GetDestinationPort\(\)](#) = Your local port (where you're listening)
- [GetSourcePort\(\)](#) = Remote sender's port (where they sent from)

#### Example Usage:

```
// Receiving on multiple ports
UDPPacket pkt;
if (pkt.Receive()) {
 uint16_t myPort = pkt.GetDestinationPort();
 uint16_t theirPort = pkt.GetSourcePort();

 if (myPort == 5000) {
 ProcessControlPacket(pkt);
 } else if (myPort == 5001) {
 ProcessDataPacket(pkt);
 }

 // Reply to sender
 UDPPacket reply;
 reply.SetDestinationPort(theirPort);
 reply.Send(pkt.GetSourceAddress());
}
```

#### See also

[SetDestinationPort\(\)](#), [GetSourcePort\(\)](#), [GetSourceAddress\(\)](#)

### 16.184.3.10 GetDSCP()

```
uint8_t UDPPacket::GetDSCP (
 void) const
```

Get DSCP value (0-63)

Returns the Differentiated Services Code Point (DSCP) value from the packet's IP header. DSCP is used for Quality of Service (QoS) classification and traffic prioritization in IP networks.

#### Returns

DSCP value in range 0-63

- 0 = Best Effort (default)
- 46 = Expedited Forwarding (EF) - low latency
- 10,12,14 = Assured Forwarding (AF) classes
- See RFC 2474 for complete DSCP definitions

**Note**

This is a const function and does not modify the packet

DSCP occupies the upper 6 bits of the IP TOS/Traffic Class field

Not all networks honor DSCP markings - depends on router configuration

| Value | Name | Description                 |
|-------|------|-----------------------------|
| 0     | CS0  | Best Effort (default)       |
| 8     | CS1  | Low Priority                |
| 46    | EF   | Expedited Forwarding (VoIP) |
| 34    | AF41 | High Priority Data          |
| 26    | AF31 | Medium Priority Data        |

**Common DSCP Values:****Example Usage:**

```
UDPPacket packet;
// ... receive packet ...
uint8_t dscp = packet.GetDSCP();
if (dscp == 46) {
 printf("High priority packet (EF)\r\n");
} else if (dscp == 0) {
 printf("Best effort packet\r\n");
}
```

**See also**

[SetDSCP\(\)](#), [RFC 2474 \(DSCP\)](#), [RFC 2475 \(DiffServ Architecture\)](#)

**16.184.3.11 GetMacSource()**

```
MACADDR UDPPacket::GetMacSource ()
```

Get sender's MAC address (link-layer identification)

Returns the source MAC address from the Ethernet frame header. This is the hardware address of the last-hop device that transmitted the packet on the local network segment.

**Returns**

[MACADDR](#) object containing the 6-byte MAC address

**Warning**

The returned MAC may be a router's MAC, not the original sender's MAC, if the packet crossed subnet boundaries. Use [GetSourceAddress\(\)](#) for end-to-end identification.

**Note**

Only valid for received packets - undefined for packets not yet sent

MAC address is Link Layer (Layer 2) - only valid on local network segment

For packets from remote subnets, this is the default gateway's MAC

Different from IP source address which identifies the original sender

| Attribute | MAC Address            | IP Address                   |
|-----------|------------------------|------------------------------|
| Scope     | Local network only     | End-to-end (across networks) |
| Identity  | Last-hop device        | Original source device       |
| Routing   | Changes at each router | Remains constant             |
| Use Case  | Link-layer filtering   | Application identification   |

### Understanding MAC vs IP Addresses:

#### Common Use Cases:

- **Device Whitelisting:** Allow only known MAC addresses
- **Local Network Detection:** Identify devices on same subnet
- **ARP Table Management:** Correlate IP to MAC mappings
- **Security Filtering:** MAC-based access control
- **Diagnostics:** Verify correct router/gateway MAC

### Expand for Example Usage

#### Examples

##### Example Usage - Basic MAC Display:

```
UDPPacket pkt;
if (pkt.Receive()) {
 MACADR mac = pkt.GetMacSource();
 printf("From MAC: %02X:%02X:%02X:%02X:%02X:%02X\r\n",
 mac.bytes[0], mac.bytes[1], mac.bytes[2],
 mac.bytes[3], mac.bytes[4], mac.bytes[5]);
}
```

##### Example Usage - MAC-Based Filtering:

```
// Whitelist of allowed MACs
const MACADR trustedDevices[] = {
 {{0x00, 0x03, 0xF4, 0x12, 0x34, 0x56}}, // Device 1
 {{0x00, 0x03, 0xF4, 0x78, 0x9A, 0xBC}} // Device 2
};

UDPPacket pkt;
if (pkt.Receive()) {
 MACADR sourceMac = pkt.GetMacSource();
 bool trusted = false;

 for (int i = 0; i < sizeof(trustedDevices)/sizeof(MACADR); i++) {
 if (memcmp(&sourceMac, &trustedDevices[i], sizeof(MACADR)) == 0) {
 trusted = true;
 break;
 }
 }

 if (trusted) {
 ProcessPacket(pkt);
 } else {
 printf("Rejected packet from untrusted MAC\r\n");
 }
}
```

##### Example Usage - Detecting Local vs Remote Packets:

```
UDPPacket pkt;
if (pkt.Receive()) {
 MACADR sourceMac = pkt.GetMacSource();
 MACADR gatewayMac = GetDefaultGatewayMAC(); // Your function

 if (memcmp(&sourceMac, &gatewayMac, sizeof(MACADR)) == 0) {
 printf("Packet came through router (remote subnet)\r\n");
 } else {
 printf("Packet from device on local subnet\r\n");
 }
}
```

**MAC Address Format:**

The [MACADR](#) structure contains 6 bytes (48 bits):

- Bytes 0-2: OUI (Organizationally Unique Identifier) - manufacturer ID
- Bytes 3-5: NIC (Network Interface Controller) - device-specific
- Example: 00:03:F4 = NetBurner OUI

**See also**

[GetSourceAddress\(\)](#), [GetMacDestination\(\)](#), [MACADR](#)

**16.184.3.12 GetPacketId()**

```
uint16_t UDPPacket::GetPacketId (
 void)
```

Get IP packet ID (used for fragment reassembly and packet tracking)

Returns the IP packet identification field from the IP header. This value is used by the network stack for IP fragmentation/reassembly and can be used by applications for packet tracking and debugging.

**Returns**

16-bit packet ID value (0-65535)

- Unique (or semi-unique) identifier for this IP packet
- Typically increments for each packet sent
- Same ID used for all fragments of a fragmented packet

**Note**

The packet ID is automatically assigned by the network stack for sent packets

For received packets, reflects the ID assigned by the sender

Not guaranteed to be globally unique - only useful within a short time window

**Common Use Cases:**

- **Fragment Reassembly:** Matching fragments of the same original packet
- **Packet Tracking:** Correlating packets in logs and network captures
- **Debugging:** Identifying specific packets in network analysis
- **Duplicate Detection:** Identifying retransmitted packets

**Example Usage:**

```
UDPPacket pkt;
pkt.AddData("Test", 4);
pkt.Send(destIP, destPort);

uint16_t packetId = pkt.GetPacketId();
printf("Sent packet with ID: %u\r\n", packetId);

// Later, when receiving response
UDPPacket response;
if (response.Receive()) {
 uint16_t responseId = response.GetPacketId();
 printf("Received packet with ID: %u\r\n", responseId);
}
```

**Warning**

Do not use packet ID as a reliable sequence number - use application-level sequencing

Packet IDs may wrap around after 65535 packets

**See also**

RFC 791 (IP), RFC 815 (IP Reassembly)

### 16.184.3.13 GetSourceAddress()

```
IPADDR UDPPacket::GetSourceAddress (
 void) [inline]
```

Get source IP address of the packet.

Returns the source IP address from the packet header. For received packets, this is the address of the remote sender. For outgoing packets, this is the local interface address used to send the packet.

#### Returns

IPADDR object containing the source address (IPv4 or IPv6)

#### Note

This function wraps GetSourceAddress6() for unified IPv4/IPv6 handling

The returned IPADDR type supports both IPv4 and IPv6 addresses

For received packets, use this to determine who sent the packet

For sent packets, this reflects which local interface was used

#### Typical Use Cases:

- Identifying the sender of a received packet
- Validating packets are from expected sources
- Building reply packets to the correct address
- Logging packet sources for debugging

#### Example Usage:

```
UDPPacket pkt;
if (pkt.Receive()) {
 IPADDR senderAddr = pkt.GetSourceAddress();
 uint16_t senderPort = pkt.GetSourcePort();

 printf("Received from %s:%u\r\n",
 senderAddr.IsIPv6() ? "IPv6" : "IPv4",
 senderPort);

 // Send reply back to sender
 UDPPacket reply;
 reply.AddData("ACK", 3);
 reply.SetDestinationPort(senderPort);
 reply.Send(senderAddr);
}
```

#### See also

[GetDestinationAddress\(\)](#), [GetSourcePort\(\)](#), [GetSourceAddress6\(\)](#)

### 16.184.3.14 GetSourcePort()

```
uint16_t UDPPacket::GetSourcePort (
 void) const
```

Get source port from received packet or previously set value.

Returns the source port from the packet's UDP header. The meaning depends on whether this is a received packet or one you're preparing to send.

#### Returns

Port number (0-65535)

- For received packets: sender's port (where they sent FROM)
- For outgoing packets: your local port (set via SetSourcePort or auto-assigned)

**Note**

This is a const function and does not modify the packet

For received packets, this is the REMOTE sender's port - use this to reply

For outgoing packets, returns the port you set or 0 if auto-assigned

Essential for implementing request/response patterns

**Understanding Port Roles in Received Packets:**

When you receive a UDP packet:

- `GetSourcePort()` = Remote sender's port (use this to reply back to them)
- `GetDestinationPort()` = Your local listening port

**Common Use Cases:**

- **Reply Routing:** Getting sender's port to send response back
- **Connection Tracking:** Identifying which client sent a request
- **Port Filtering:** Accepting packets only from specific source ports
- **Logging:** Recording which port clients are using
- **Security:** Validating source port is in expected range

**Expand for Example Usage****Examples****Example Usage - Server Replying to Client:**

```
// Server receives request and replies to client
UDPPacket request;
if (request.Receive()) {
 IPADDR clientAddr = request.GetSourceAddress();
 uint16_t clientPort = request.GetSourcePort(); // Where to reply

 printf("Request from %s:%u\r\n",
 clientAddr.ToString(), clientPort);

 // Send response back to client
 UDPPacket response;
 response.SetDestinationPort(clientPort); // Use client's port
 response.AddData("ACK", 3);
 response.Send(clientAddr);
}
```

**Example Usage - Port-Based Request Routing:**

```
// Different handlers based on which port client used
UDPPacket pkt;
if (pkt.Receive()) {
 uint16_t clientPort = pkt.GetSourcePort();

 // Some protocols use specific source port ranges
 if (clientPort >= 49152 && clientPort <= 65535) {
 // Ephemeral port - typical client
 ProcessNormalRequest(pkt);
 } else if (clientPort == 5000) {
 // Specific port - might be peer-to-peer
 ProcessPeerRequest(pkt);
 } else {
 printf("Unusual source port: %u\r\n", clientPort);
 }
}
```

**Example Usage - Session Tracking:**

```
// Track multiple clients by IP:Port combination
struct ClientSession {
 IPADDR addr;
 uint16_t port;
 uint32_t lastSeen;
};

ClientSession sessions[MAX_CLIENTS];

UDPPacket pkt;
```

```

if (pkt.Receive()) {
 IPADDR clientAddr = pkt.GetSourceAddress();
 uint16_t clientPort = pkt.GetSourcePort();

 // Find or create session for this client
 for (int i = 0; i < MAX_CLIENTS; i++) {
 if (sessions[i].addr == clientAddr &&
 sessions[i].port == clientPort) {
 sessions[i].lastSeen = Secs; // Update timestamp
 ProcessKnownClient(pkt, &sessions[i]);
 break;
 }
 }
}

```

#### Example Usage - Security Filtering:

```

// Only accept packets from high ports (non-privileged clients)
UDPPacket pkt;
if (pkt.Receive()) {
 uint16_t sourcePort = pkt.GetSourcePort();

 if (sourcePort < 1024) {
 printf("Rejecting packet from privileged port %u\r\n", sourcePort);
 return; // Ignore packets from system ports
 }

 ProcessPacket(pkt);
}

```

#### See also

[SetSourcePort\(\)](#), [GetDestinationPort\(\)](#), [GetSourceAddress\(\)](#), [Send\(\)](#)

#### 16.184.3.15 Receive()

```

bool UDPPacket::Receive (
 OS_FIFO * pFifo,
 uint32_t wait_ticks) [inline]

```

Reuse packet object for multiple receives (efficiency optimization)

Receives a UDP packet from the specified FIFO queue with a timeout. This function blocks until a packet arrives or the timeout expires. The same packet object can be reused across multiple receives for efficiency (avoids repeated allocation).

#### Parameters

|                   |                                                                                                                                                                                                                                                        |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>pFifo</i>      | Pointer to OS_FIFO queue to receive from (created by <a href="#">RegisterUDPFifo()</a> )                                                                                                                                                               |
| <i>wait_ticks</i> | Maximum time to wait for packet, in system ticks <ul style="list-style-type: none"> <li>• 0 = Non-blocking (return immediately if no packet)</li> <li>• TICKS_PER_SECOND = 1 second timeout</li> <li>• 0xFFFFFFFF = Wait forever (blocking)</li> </ul> |

#### Returns

true if packet received successfully  
false if timeout occurred or error

#### Note

**ALWAYS** call [Validate\(\)](#) after successful receive to check packet integrity

The packet object is reused - no need to create new [UDPPacket](#) each time

Previous packet data is overwritten on each successful receive

This is the preferred receive method for production code (efficient)



### Warning

Never process packet data without calling `Validate()` first

The pFifo must be created with `RegisterUDPFifo()` before use

`wait_ticks=0` returns immediately - check return value before accessing data

### Efficiency Benefits:

Reusing the same `UDPPacket` object across multiple receives:

- **Reduces memory allocation overhead** - no malloc/free per packet
- **Improves cache performance** - same memory addresses
- **Eliminates fragmentation** - stable memory footprint
- **Faster execution** - critical for high packet rates

### FIFO Setup:

Before calling `Receive()`, you must create a FIFO for the port:

```
OS_FIFO udpFifo;
OSFifoInit(&udpFifo);
RegisterUDPFifo(5000, &udpFifo); // Listen on port 5000
```

### Expand for Example Usage

#### Examples

##### Example Usage - Basic Server Loop:

```
// Setup
OS_FIFO udpFifo;
OSFifoInit(&udpFifo);
RegisterUDPFifo(5000, &udpFifo);

// Efficient receive loop - reuses same packet object
UDPPacket pkt;
while (running) {
 if (pkt.Receive(&udpFifo, TICKS_PER_SECOND * 2)) {
 if (pkt.Validate()) {
 ProcessPacket(pkt);
 } else {
 printf("Corrupted packet discarded\r\n");
 }
 } else {
 // Timeout - no packet received in 2 seconds
 HandleTimeout();
 }
}
```

##### Example Usage - Non-Blocking Receive:

```
UDPPacket pkt;

// Check for packet without blocking
if (pkt.Receive(&udpFifo, 0)) { // wait_ticks = 0
 if (pkt.Validate()) {
 printf("Packet received!\r\n");
 ProcessPacket(pkt);
 }
} else {
 // No packet available right now - continue other work
 DoOtherWork();
}
```

##### Example Usage - Multiple Ports:

```
// Listen on multiple ports with different FIFOs
OS_FIFO controlFifo, dataFifo;
OSFifoInit(&controlFifo);
OSFifoInit(&dataFifo);
RegisterUDPFifo(5000, &controlFifo); // Control port
RegisterUDPFifo(5001, &dataFifo); // Data port

UDPPacket pkt; // Reused for both ports
while (running) {
 // Check control port (non-blocking)
 if (pkt.Receive(&controlFifo, 0)) {
 if (pkt.Validate()) {
 ProcessControl(pkt);
 }
 }
}
```

```

 }
}

// Check data port (100ms timeout)
if (pkt.Receive(&dataFifo, TICKS_PER_SECOND / 10)) {
 if (pkt.Validate()) {
 ProcessData(pkt);
 }
}
}

```

#### Example Usage - Heartbeat with Timeout:

```

UDP_PACKET pkt;
uint32_t lastPacketTime = Secs;
const uint32_t HEARTBEAT_TIMEOUT = 10; // seconds

while (running) {
 // Wait up to 1 second for packet
 if (pkt.Receive(&udpFifo, TICKS_PER_SECOND)) {
 if (pkt.Validate()) {
 lastPacketTime = Secs;
 ProcessPacket(pkt);
 }
 }

 // Check for heartbeat timeout
 if ((Secs - lastPacketTime) > HEARTBEAT_TIMEOUT) {
 printf("Connection lost - no packets for %lu seconds\r\n",
 Secs - lastPacketTime);
 HandleConnectionLost();
 lastPacketTime = Secs; // Reset timer
 }
}

```

| wait_ticks Value      | Behavior     | Use Case                  |
|-----------------------|--------------|---------------------------|
| 0                     | Non-blocking | Polling, multi-tasking    |
| TICKS_PER_SECOND / 10 | 100ms        | Responsive UI updates     |
| TICKS_PER_SECOND      | 1 second     | Typical network timeout   |
| TICKS_PER_SECOND * 5  | 5 seconds    | Slow networks, retries    |
| 0xFFFFFFFF            | Forever      | Dedicated receive threads |

#### Timeout Values:

#### Performance Comparison:

```

// INEFFICIENT - Creates new packet each iteration
while (running) {
 UDP_PACKET pkt; // BAD: Allocates memory every loop
 if (pkt.Receive(&fifo, timeout)) {
 // process
 }
}

// EFFICIENT - Reuses same packet object
UDP_PACKET pkt; // GOOD: Allocated once
while (running) {
 if (pkt.Receive(&fifo, timeout)) {
 // process (much faster!)
 }
}

```

#### See also

[Validate\(\)](#), [RegisterUDPFifo\(\)](#), [OSFifoInit\(\)](#), [Send\(\)](#)

#### 16.184.3.16 ResetData()

```

void UDP_PACKET::ResetData (
 void)

```

Reset data size to 0 (clears packet)

This function resets the packet's data length to zero, effectively clearing the packet contents without deallocating memory. The packet buffer remains allocated and can be reused for new data.

**Note**

This does not free memory or destroy the packet object  
The packet buffer remains valid and can be immediately reused  
Does not reset packet metadata (addresses, ports, flags, etc.)

**Warning**

After calling this function, the packet contains no valid data. Any previous data references become invalid.

**Typical Use Cases:**

- Reusing a packet buffer for multiple transmissions
- Clearing received data before processing new packets
- Resetting packet state in error conditions

**Example Usage:**

```
UDPPacket packet;
packet.AddData("Hello", 5);
packet.Send(destIP, destPort);

// Reuse the same packet
packet.ResetData();
packet.AddData("World", 5);
packet.Send(destIP, destPort);
```

**See also**

[Validate\(\)](#)

**16.184.3.17 Send()**

```
void UDPPacket::Send (
 const IPADDR & to,
 uint8_t ttl = 0) [inline]
```

Send and free buffer (recommended - most efficient)

Sends the packet to the specified destination and immediately frees the buffer. This is the standard, most efficient way to send UDP packets. After calling this function, the packet buffer is freed and the [UDPPacket](#) object should not be used for sending again (though it can receive new data).

**Parameters**

|            |                                                                                                                                                                                                                                                                  |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>to</i>  | Destination IP address (IPv4 or IPv6)                                                                                                                                                                                                                            |
| <i>ttl</i> | Time-to-live value for packet (0 = use default TTL, typically 64) <ul style="list-style-type: none"><li>• 0: Use system default TTL</li><li>• 1: Local subnet only</li><li>• 32-64: Typical for Internet routing</li><li>• 255: Maximum (unrestricted)</li></ul> |

**Note**

This function wraps `Send6()` for unified IPv4/IPv6 handling  
Buffer is FREED immediately after send - most memory efficient  
This is the recommended send method for single-destination packets  
After [Send\(\)](#), destructor has nothing to clean up  
Cannot send to additional destinations after this call

**Warning**

- Do NOT access packet buffer after calling `Send()`
- Do NOT call `Send()` multiple times on the same packet
- Use `SendAndKeep()` instead if sending to multiple destinations

**Use Cases:**

- Standard single-destination UDP send (most common)
- Client sending request to server
- Server sending response to client
- Any scenario with one recipient
- When memory efficiency is important

**Typical Pattern:**

The most common UDP send pattern:

```
UDPPacket pkt;
pkt.SetSourcePort(5000);
pkt.SetDestinationPort(8080);
pkt.AddData("REQUEST", 7);
pkt.Send(serverIP); // Send and free - done!
```

**Memory Efficiency:**

```
Send() immediately:
Allocate -> Fill -> Send -> Free (minimal memory footprint)

vs SendAndKeep():
Allocate -> Fill -> Copy+Send -> Copy+Send -> Free (higher footprint)
```

**Expand for Example Usage****Examples****Example - Simple Request/Response:**

```
// Send request
UDPPacket request;
request.SetDestinationPort(5000);
request.AddData("GET_STATUS", 10);
request.Send(serverIP); // Buffer freed here

// Receive response
OS_FIFO responseFifo;
OSFifoInit(&responseFifo);
RegisterUDPFifo(0, &responseFifo);

UDPPacket response;
if (response.Receive(&responseFifo, TICKS_PER_SECOND * 5)) {
 if (response.Validate()) {
 ProcessResponse(response);
 }
}
```

**Example - Sensor Data Transmission:**

```
void SendSensorReading(float temperature, float humidity) {
 struct SensorData {
 uint32_t timestamp;
 float temp;
 float humid;
 } data = {Secs, temperature, humidity};

 UDPPacket pkt;
 pkt.SetDestinationPort(6000);
 pkt.AddData((uint8_t*)&data, sizeof(data));
 pkt.Send(dataCollectorIP); // Efficient single send
}
```

**Example - Efficient Batch Sending:**

```
// Reuse packet for multiple separate sends
UDPPacket pkt;
pkt.SetDestinationPort(7000);

for (int i = 0; i < 100; i++) {
 pkt.ResetData();
 pkt.AddDataWord(i);
 pkt.AddData("Data packet", 11);
 pkt.Send(serverIP); // Each send frees buffer

 OSTimeDly(TICKS_PER_SECOND / 10); // 100ms between packets
}
```

| Method                        | Memory | Speed  | Use When              |
|-------------------------------|--------|--------|-----------------------|
| <a href="#">Send()</a>        | Low    | Fast   | Single destination    |
| <a href="#">SendAndKeep()</a> | High   | Slower | Multiple destinations |

**Performance Comparison:****TTL Common Values:**

```
// Local subnet only (TTL=1)
pkt.Send(localIP, 1);

// Standard Internet (TTL=0, uses default ~64)
pkt.Send(remoteIP, 0);

// Extended reach (TTL=128)
pkt.Send(distantIP, 128);
```

**Lifecycle Comparison:**

```
// Method 1: Send() - Simple and efficient
{
 UDPPacket pkt;
 pkt.AddData("Hello", 5);
 pkt.Send(dest); // Buffer freed here
} // Destructor has nothing to do

// Method 2: No Send() - Destructor cleans up
{
 UDPPacket pkt;
 pkt.AddData("Hello", 5);
 // Forgot to send?
} // Destructor frees buffer (but packet never sent!)

// Method 3: SendAndKeep() - More complex
{
 UDPPacket pkt;
 pkt.AddData("Hello", 5);
 pkt.SendAndKeep(dest1); // Buffer still allocated
 pkt.Send(dest2); // Buffer freed here
} // Destructor has nothing to do
```

**Common Mistakes:**

```
// WRONG: Trying to send after Send()
UDPPacket pkt;
pkt.AddData("Hello", 5);
pkt.Send(dest1); // Buffer freed
pkt.Send(dest2); // CRASH: Buffer already freed!

// CORRECT: Use SendAndKeep() for multiple
UDPPacket pkt;
pkt.AddData("Hello", 5);
pkt.SendAndKeep(dest1); // Buffer retained
pkt.Send(dest2); // Buffer freed

// WRONG: Accessing buffer after Send()
UDPPacket pkt;
pkt.AddData("Hello", 5);
pkt.Send(dest);
pkt.AddData("More", 4); // CRASH: Buffer gone!

// CORRECT: Reset and reuse packet
```

```

UDPPacket pkt;
pkt.AddData("Hello", 5);
pkt.Send(dest);
// Wait for new data or receive
pkt.Receive(&fifo, timeout); // Gets new buffer

```

#### Best Practices:

- **Default choice:** Use [Send\(\)](#) for single-destination packets
- **Memory efficient:** Frees buffer immediately
- **Simple pattern:** Allocate -> Fill -> Send -> Done
- **Reusable packet:** Can receive new data after [Send\(\)](#)
- **Clear ownership:** Buffer freed, no ambiguity

#### See also

[SendAndKeep\(\)](#), [SendViaInterfaceNum\(\)](#), [AddData\(\)](#), [SetDestinationPort\(\)](#)

### 16.184.3.18 SendAndKeep()

```

void UDPPacket::SendAndKeep (
 const IPADDR & to,
 uint8_t ttl = 0) [inline]

```

Send copy, keep original (for sending to multiple destinations)

Sends a copy of the packet to the specified destination while retaining the original buffer for additional sends. This is essential when sending the same packet to multiple destinations. The buffer remains allocated until either [Send\(\)](#) is called or the packet is destroyed.

#### Parameters

|            |                                                                                                                                                                                                                                                                       |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>to</i>  | Destination IP address (IPv4 or IPv6)                                                                                                                                                                                                                                 |
| <i>ttl</i> | Time-to-live value for packet (0 = use default TTL, typically 64) <ul style="list-style-type: none"> <li>• 0: Use system default TTL</li> <li>• 1: Local subnet only</li> <li>• 32-64: Typical for Internet routing</li> <li>• 255: Maximum (unrestricted)</li> </ul> |

#### Note

This function wraps [SendAndKeep6\(\)](#) for unified IPv4/IPv6 handling

Buffer is RETAINED after send - you must eventually call [Send\(\)](#) or let destructor run

Each call creates a new copy of the packet data

Less efficient than [Send\(\)](#) due to buffer copying

Use when you need to send the same data to multiple recipients

#### Warning

Must eventually free buffer with [Send\(\)](#) or allow destructor to run

Memory leak if you only use [SendAndKeep\(\)](#) and never [Send\(\)](#)

Each [SendAndKeep\(\)](#) allocates a new buffer - don't use in tight loops

**Use Cases:**

- Sending same data to multiple servers (redundancy)
- Broadcasting to multiple monitoring stations
- Fan-out messaging patterns
- Multicast alternative for small recipient lists
- Logging to multiple destinations

**Typical Pattern:**

Always end with `Send()` to free the buffer:

```
UDPPacket pkt;
pkt.SetDestinationPort(8080);
pkt.AddData("Alert: System temperature high", 30);

pkt.SendAndKeep(server1); // Copy sent, buffer kept
pkt.SendAndKeep(server2); // Copy sent, buffer kept
pkt.Send(server3); // Copy sent, buffer freed
// Buffer automatically freed by last Send()
```

**Memory Management:**

```
SendAndKeep(dest1) -> Creates copy -> Sends -> Keeps original
SendAndKeep(dest2) -> Creates copy -> Sends -> Keeps original
Send(dest3) -> Sends original -> Frees buffer

OR if no final Send():
SendAndKeep(dest1) -> Creates copy -> Sends -> Keeps original
SendAndKeep(dest2) -> Creates copy -> Sends -> Keeps original
~UDPPacket() -> Destructor frees original buffer
```

**Example - Multiple Monitoring Servers:**

```
void SendStatusToMonitors(const char* status) {
 IPADDR monitors[] = {
 AsciiToIp("192.168.1.10"),
 AsciiToIp("192.168.1.11"),
 AsciiToIp("192.168.1.12")
 };

 UDPPacket pkt;
 pkt.SetDestinationPort(9000);
 pkt.AddData(status);

 // Send to all but last
 for (int i = 0; i < 2; i++) {
 pkt.SendAndKeep(monitors[i]);
 }

 // Last send frees buffer
 pkt.Send(monitors[2]);
}
```

**Example - Redundant Server Communication:**

```
// Send critical data to primary and backup servers
UDPPacket criticalData;
criticalData.SetDestinationPort(5000);
criticalData.AddData("CRITICAL:SHUTDOWN", 17);

criticalData.SendAndKeep(primaryServer); // Send to primary
criticalData.Send(backupServer); // Send to backup and free
```

| Aspect      | Cost       | Notes                                                |
|-------------|------------|------------------------------------------------------|
| Buffer copy | Medium     | Each <code>SendAndKeep()</code> allocates new buffer |
| Network I/O | High       | N separate transmissions                             |
| Memory      | Higher     | Temporary buffers until final <code>Send()</code>    |
| Use when    | N is small | Typically < 10 destinations                          |

**Performance Considerations:**

**SendAndKeep() vs Send():**

```
// SendAndKeep() - Use for multiple destinations
pkt.SendAndKeep(dest1); // Buffer retained
pkt.Send(dest2); // Buffer freed

// Send() - Use for single destination (more efficient)
pkt.Send(dest); // Buffer freed immediately
```

| TTL | Scope                       | Use Case                     |
|-----|-----------------------------|------------------------------|
| 0   | System default (usually 64) | Normal operation             |
| 1   | Local subnet only           | Link-local communication     |
| 16  | Limited regional            | Campus networks              |
| 64  | Internet standard           | General Internet traffic     |
| 128 | Extended reach              | Cross-continent              |
| 255 | Maximum                     | Unrestricted (rarely needed) |

**TTL Values Guide:****Common Mistakes:**

```
// WRONG: Only SendAndKeep(), never freed
UDPPacket pkt;
pkt.AddData("Hello", 5);
pkt.SendAndKeep(dest1);
pkt.SendAndKeep(dest2);
// Memory leak! Buffer never freed

// CORRECT: End with Send()
UDPPacket pkt;
pkt.AddData("Hello", 5);
pkt.SendAndKeep(dest1);
pkt.Send(dest2); // Frees buffer

// ALSO CORRECT: Let destructor handle it
{
 UDPPacket pkt;
 pkt.AddData("Hello", 5);
 pkt.SendAndKeep(dest1);
 pkt.SendAndKeep(dest2);
} // Destructor frees buffer
```

**See also**

[Send\(\)](#), [SendViaInterfaceNum\(\)](#), [SendAndKeepViaInterfaceNum\(\)](#), [~UDPPacket\(\)](#)

**16.184.3.19 SendAndKeepViaIfAddr()**

```
void UDPPacket::SendAndKeepViaIfAddr (
 const IPADDR & to,
 const IPADDR & from_ip,
 uint8_t ttl = 0) [inline]
```

Send a packet and retain a copy via a specific interface address.

This inline function wraps [SendAndKeepViaIfAddr6\(\)](#) to send a packet to a destination IP address while routing through a specific source interface address. The function retains a copy of the packet after sending (useful for retransmission or logging).

**Parameters**

|                |                                                           |
|----------------|-----------------------------------------------------------|
| <i>to</i>      | Destination IP address for the packet                     |
| <i>from_ip</i> | Source interface IP address to use for routing the packet |
| <i>ttl</i>     | Time-to-live value for the packet (0 = use default TTL)   |



**Note**

This is a convenience wrapper that delegates to `SendAndKeepVialfAddr6()`

The "keep" behavior suggests the packet buffer is preserved for potential reuse

**See also**

`SendAndKeepVialfAddr6()`

**16.184.3.20 SendAndKeepViaInterfaceNum()**

```
void UDPPacket::SendAndKeepViaInterfaceNum (
 const IPADDR & to,
 int interface,
 uint8_t ttl = 0) [inline]
```

Send copy via specific interface, keep original.

Sends a copy of the packet via a specific network interface while retaining the original buffer. This is essential for multi-homed systems where you need to send the same packet through multiple network interfaces or send to multiple destinations via specific interfaces.

**Parameters**

|                  |                                                                                                                                                                                                                                                        |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>to</i>        | Destination IP address (IPv4 or IPv6)                                                                                                                                                                                                                  |
| <i>interface</i> | Network interface number (0=first interface, 1=second, etc.) <ul style="list-style-type: none"><li>• 0: Primary interface (usually Ethernet 0)</li><li>• 1: Secondary interface (Ethernet 1, WiFi, etc.)</li><li>• 2+: Additional interfaces</li></ul> |
| <i>ttl</i>       | Time-to-live value for packet (0 = use default TTL) <ul style="list-style-type: none"><li>• 0: Use system default TTL</li><li>• 1: Local subnet only</li><li>• 32-64: Typical for Internet routing</li><li>• 255: Maximum (unrestricted)</li></ul>     |

**Note**

This function wraps `SendAndKeepViaInterfaceNum6()` for IPv4/IPv6 support

Buffer is RETAINED after send - must eventually call [Send\(\)](#) or rely on destructor

Each call creates a new copy of the packet data

Use when you need to send same data through multiple interfaces

Interface number must be valid (0 to `GetNumberOfInterfaces()-1`)

**Warning**

Must eventually free buffer with [Send\(\)](#) or allow destructor to run

Invalid interface numbers may cause send to fail silently

Buffer remains allocated - potential memory pressure if many calls

**Use Cases:**

- **Multi-homed systems:** Send via specific Ethernet port
- **Network redundancy:** Send same data via multiple interfaces
- **Interface testing:** Verify connectivity through specific interfaces
- **Network segregation:** Management vs data networks
- **Failover systems:** Primary and backup network paths

**Typical Pattern:**

Send via multiple interfaces, free with final [Send\(\)](#):

```
UDPPacket pkt;
pkt.SetDestinationPort(8080);
pkt.AddData("Status update", 13);

// Send via interface 0 (primary network)
pkt.SendAndKeepViaInterfaceNum(server, 0);

// Send via interface 1 (backup network)
pkt.SendViaInterfaceNum(server, 1); // Last send frees buffer
```

**Expand for Example Usage****Examples****Example - Redundant Networks:**

```
// System with dual Ethernet ports
void SendCriticalData(const char* data, IPADDR serverIP) {
 UDPPacket pkt;
 pkt.SetDestinationPort(9000);
 pkt.AddData(data);

 // Send via primary Ethernet
 pkt.SendAndKeepViaInterfaceNum(serverIP, 0);

 // Send via backup Ethernet
 pkt.SendViaInterfaceNum(serverIP, 1); // Frees buffer

 printf("Data sent via both interfaces\r\n");
}
```

**Example - Network Segregation:**

```
// Separate management and production networks
#define MGMT_INTERFACE 0
#define PROD_INTERFACE 1

void SendToMonitoring(const char* alert) {
 UDPPacket pkt;
 pkt.SetDestinationPort(514); // Syslog
 pkt.AddData(alert);

 // Send to management server via management interface
 pkt.SendAndKeepViaInterfaceNum(mgmtServer, MGMT_INTERFACE);

 // Send to production monitor via production interface
 pkt.SendViaInterfaceNum(prodMonitor, PROD_INTERFACE);
}
```

**Example - Interface Testing:**

```
// Test connectivity through each interface
void TestInterfaces(IPADDR testServer) {
 int numInterfaces = GetNumberOfInterfaces();

 UDPPacket pkt;
 pkt.SetDestinationPort(7000);
 pkt.AddData("PING", 4);

 for (int i = 0; i < numInterfaces - 1; i++) {
 printf("Testing interface %d\r\n", i);
 pkt.SendAndKeepViaInterfaceNum(testServer, i);
 }

 // Last interface - free buffer
 pkt.SendViaInterfaceNum(testServer, numInterfaces - 1);
}
```

| Scenario       | Interface Selection | Reason                       |
|----------------|---------------------|------------------------------|
| Primary data   | Interface 0         | Main network connection      |
| Management     | Interface 1         | Isolated management network  |
| Testing        | Iterate all         | Verify all paths working     |
| Redundancy     | Multiple            | Ensure delivery via any path |
| Load balancing | Round-robin         | Distribute traffic           |

#### Interface Selection Strategy:

#### Memory Management:

```
SendAndKeepViaIfNum(dest, 0) -> Copy -> Send via if0 -> Keep original
SendAndKeepViaIfNum(dest, 1) -> Copy -> Send via if1 -> Keep original
SendViaInterfaceNum(dest, 2) -> Send via if2 -> Free original
```

OR let destructor handle:

```
SendAndKeepViaIfNum(dest, 0) -> Copy -> Send via if0 -> Keep original
SendAndKeepViaIfNum(dest, 1) -> Copy -> Send via if1 -> Keep original
~UDPPacket() -> Destructor frees original
```

#### Checking Available Interfaces:

```
// Get number of interfaces
int numIfs = GetNumberOfInterfaces();
printf("System has %d network interfaces\r\n", numIfs);

// Get interface details
for (int i = 0; i < numIfs; i++) {
 InterfaceBlock* ifBlock = GetInterfaceBlock(i);
 if (ifBlock) {
 printf("Interface %d: %s\r\n", i,
 ifBlock->GetIPAddr().ToString());
 }
}
```

| Aspect             | Impact | Notes                              |
|--------------------|--------|------------------------------------|
| Buffer copy        | Medium | Each call allocates new buffer     |
| Network I/O        | High   | N separate transmissions           |
| Interface overhead | Low    | Just routing decision              |
| Memory             | Higher | Temporary buffers until final send |

#### Performance Considerations:

#### Common Mistakes:

```
// WRONG: Invalid interface number
pkt.SendAndKeepViaInterfaceNum(dest, 99); // Likely invalid!

// CORRECT: Check interface exists
int numIfs = GetNumberOfInterfaces();
if (interfaceNum < numIfs) {
 pkt.SendAndKeepViaInterfaceNum(dest, interfaceNum);
}

// WRONG: Never freed
pkt.SendAndKeepViaInterfaceNum(dest, 0);
pkt.SendAndKeepViaInterfaceNum(dest, 1);
// Memory leak!

// CORRECT: End with Send()
pkt.SendAndKeepViaInterfaceNum(dest, 0);
pkt.SendViaInterfaceNum(dest, 1); // Frees buffer
```

#### See also

[SendViaInterfaceNum\(\)](#), [SendAndKeep\(\)](#), [Send\(\)](#), [GetNumberOfInterfaces\(\)](#), [GetInterfaceBlock\(\)](#)

**16.184.3.21 SendViaIfAddr()**

```
void UDPPacket::SendViaIfAddr (
 const IPADDR & to,
 const IPADDR & from_ip,
 uint8_t ttl = 0) [inline]
```

Send a packet via a specific interface address.

This inline function sends a packet to a destination IP address while routing through a specific source interface address. Unlike [SendAndKeepViaIfAddr\(\)](#), this function does not retain a copy of the packet after transmission.

**Parameters**

|                |                                                           |
|----------------|-----------------------------------------------------------|
| <i>to</i>      | Destination IP address for the packet                     |
| <i>from_ip</i> | Source interface IP address to use for routing the packet |
| <i>ttl</i>     | Time-to-live value for the packet (0 = use default TTL)   |

**Warning**

This function appears to be recursive - it calls itself. This is likely a bug and should probably call a different underlying implementation function.

**Note**

The recursive call will cause a stack overflow. Verify the intended implementation.

**16.184.3.22 SendViaInterfaceNum()**

```
void UDPPacket::SendViaInterfaceNum (
 const IPADDR & to,
 int interface,
 uint8_t ttl = 0) [inline]
```

Send via specific interface and free buffer.

Sends the packet via a specific network interface and immediately frees the buffer. This is the standard way to send a packet through a specific interface in multi-homed systems. The buffer is freed after sending, making this the most memory-efficient option for single-interface sends.

**Parameters**

|                  |                                                                                                                                                                                                                                                                        |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>to</i>        | Destination IP address (IPv4 or IPv6)                                                                                                                                                                                                                                  |
| <i>interface</i> | Network interface number (0=first interface, 1=second, etc.) <ul style="list-style-type: none"> <li>• 0: Primary interface (usually Ethernet 0)</li> <li>• 1: Secondary interface (Ethernet 1, WiFi, etc.)</li> <li>• 2+: Additional interfaces</li> </ul>             |
| <i>ttl</i>       | Time-to-live value for packet (0 = use default TTL) <ul style="list-style-type: none"> <li>• 0: Use system default TTL (typically 64)</li> <li>• 1: Local subnet only</li> <li>• 32-64: Typical for Internet routing</li> <li>• 255: Maximum (unrestricted)</li> </ul> |

**Note**

This function wraps `SendViaInterfaceNum6()` for IPv4/IPv6 support  
 Buffer is FREED immediately after send  
 This is the recommended method for single-interface sends  
 Interface number must be valid (0 to `GetNumberOfInterfaces()-1`)  
 After this call, the buffer is freed and packet cannot send again

**Warning**

Do NOT access packet buffer after calling this function  
 Do NOT call send functions multiple times on the same packet  
 Invalid interface numbers may cause send to fail silently  
 Use [SendAndKeepViaInterfaceNum\(\)](#) if sending via multiple interfaces

**Use Cases:**

- **Interface-specific routing:** Force packet through specific interface
- **Multi-homed systems:** Separate management from production traffic
- **Network segregation:** Different interfaces for different purposes
- **Testing:** Verify specific interface functionality
- **Policy routing:** Business rules require specific interface

**Typical Pattern:**

Standard single-interface send:

```
UDPPacket pkt;
pkt.SetDestinationPort(8080);
pkt.AddData("Status", 6);

// Send via secondary interface (e.g., management network)
pkt.SendViaInterfaceNum(serverIP, 1); // Send and free
```

---

**Expand for Example Usage****Examples****Example - Management Network:**

```
// Send management traffic via dedicated interface
#define MGMT_INTERFACE 1

void SendManagementAlert(const char* alert, IPADDR mgmtServer) {
 UDPPacket pkt;
 pkt.SetDestinationPort(514); // Syslog
 pkt.AddData(alert);

 // Send via management interface only
 pkt.SendViaInterfaceNum(mgmtServer, MGMT_INTERFACE);

 printf("Alert sent via management interface\r\n");
}
```

**Example - Production Data Routing:**

```
// Route production data via high-speed interface
#define PROD_INTERFACE 0

void SendProductionData(const uint8_t* data, uint16_t len, IPADDR dest) {
 UDPPacket pkt;
 pkt.SetDestinationPort(6000);
 pkt.AddData(data, len);

 // Force through production interface
 pkt.SendViaInterfaceNum(dest, PROD_INTERFACE);
}
```

**Example - Interface Selection Based on Destination:**

```
void SendToDestination(const char* data, IPADDR dest) {
 UDPPacket pkt;
 pkt.SetDestinationPort(7000);
 pkt.AddData(data);

 // Choose interface based on destination
 int interface;
 if (IsManagementNetwork(dest)) {
 interface = 1; // Management interface
 } else {
 interface = 0; // Production interface
 }

 pkt.SendViaInterfaceNum(dest, interface);
}
```

**Example - Round-Robin Load Balancing:**

```
// Distribute traffic across multiple interfaces
static int nextInterface = 0;

void SendLoadBalanced(const char* data, IPADDR dest) {
 int numInterfaces = GetNumberOfInterfaces();

 UDPPacket pkt;
 pkt.SetDestinationPort(8000);
 pkt.AddData(data);

 // Send via next interface in rotation
 pkt.SendViaInterfaceNum(dest, nextInterface);

 // Move to next interface
 nextInterface = (nextInterface + 1) % numInterfaces;
}
```

| System Type       | Interface 0 | Interface 1 | Use Case               |
|-------------------|-------------|-------------|------------------------|
| Dual Ethernet     | Production  | Management  | Network segregation    |
| Ethernet + WiFi   | Wired       | Wireless    | Redundant connectivity |
| Multi-port router | LAN         | WAN         | Traffic direction      |
| Industrial system | Control     | Data        | Real-time vs bulk      |

**Interface Configuration Examples:****Verifying Interface Availability:**

```
bool SendViaInterfaceSafe(UDPPacket& pkt, IPADDR dest, int ifNum) {
 // Check if interface exists and is up
 if (ifNum < 0 || ifNum >= GetNumberOfInterfaces()) {
 printf("Invalid interface: %d\r\n", ifNum);
 return false;
 }

 InterfaceBlock* ifBlock = GetInterfaceBlock(ifNum);
 if (!ifBlock || !ifBlock->IsUp()) {
 printf("Interface %d is down\r\n", ifNum);
 return false;
 }

 // Interface valid and up
 pkt.SendViaInterfaceNum(dest, ifNum);
 return true;
}
```

| Method                                 | Memory | Speed   | Complexity | Use When                   |
|----------------------------------------|--------|---------|------------|----------------------------|
| <a href="#">SendViaInterfaceNum()</a>  | Low    | Fast    | Low        | Single interface           |
| <a href="#">SendAndKeepViaIfNum()</a>  | High   | Slower  | Medium     | Multiple interfaces        |
| <a href="#">Send()</a> (any interface) | Low    | Fastest | Lowest     | Don't care which interface |

**Performance Comparison:**

**TTL and Routing:**

```
// Local network via specific interface
pkt.SendViaInterfaceNum(localDest, 0, 1); // TTL=1, local only

// Internet destination via WAN interface
pkt.SendViaInterfaceNum(remoteDest, 1, 0); // TTL=default

// Extended reach via backup interface
pkt.SendViaInterfaceNum(farDest, 2, 128); // TTL=128, far reach
```

**Common Mistakes:**

```
// WRONG: Invalid interface number
pkt.SendViaInterfaceNum(dest, 99); // Probably invalid!

// CORRECT: Validate interface first
if (interface < GetNumberOfInterfaces()) {
 pkt.SendViaInterfaceNum(dest, interface);
}

// WRONG: Trying to send again after Send()
pkt.SendViaInterfaceNum(dest, 0); // Buffer freed
pkt.SendViaInterfaceNum(dest, 1); // CRASH: Buffer gone!

// CORRECT: Use SendAndKeep() for multiple
pkt.SendAndKeepViaInterfaceNum(dest, 0); // Buffer kept
pkt.SendViaInterfaceNum(dest, 1); // Buffer freed

// WRONG: Assuming interface exists
pkt.SendViaInterfaceNum(dest, 1); // What if only 1 interface?

// CORRECT: Check interface count
int numIf = GetNumberOfInterfaces();
if (numIf > 1) {
 pkt.SendViaInterfaceNum(dest, 1);
} else {
 pkt.Send(dest); // Use default interface
}
```

**Best Practices:**

- **Validate interface numbers** before use
- **Check interface status** (up/down) before sending
- **Use for policy routing** when specific path required
- **Default choice for multi-homed** systems with routing requirements
- **Document interface assignments** in code comments

**See also**

[SendAndKeepViaInterfaceNum\(\)](#), [Send\(\)](#), [SendViaIfAddr\(\)](#), [GetNumberOfInterfaces\(\)](#), [GetInterfaceBlock\(\)](#)

**16.184.3.23 SetDataSize()**

```
void UDPPacket::SetDataSize (
 uint16_t numBytes)
```

Set data size (REQUIRED after [GetDataBuffer\(\)](#) writes)

Maximum ~1472 bytes for standard Ethernet. Larger needs fragmentation.

```
// WRONG:
sprintf(pkt.GetDataBuffer(), "Hello");
pkt.Send(ip); // BUG: Size not set!

// CORRECT:
uint16_t len = sprintf(pkt.GetDataBuffer(), "Hello");
pkt.SetDataSize(len);
pkt.Send(ip);
```

**16.184.3.24 SetDestinationPort()**

```
void UDPPacket::SetDestinationPort (
 uint16_t port)
```

Set destination port (standard services: 53=DNS, 80=HTTP, 123=NTP, 161=SNMP)

Sets the destination UDP port for outgoing packets. The port identifies which service or application on the remote host should receive the packet.

## Parameters

|             |                                                                                                                                                                                                                                                                                                    |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>port</i> | Destination port number (1-65535) <ul style="list-style-type: none"> <li>• 0 is reserved and should not be used</li> <li>• 1-1023 are well-known ports (require privileges on most systems)</li> <li>• 1024-49151 are registered ports</li> <li>• 49152-65535 are dynamic/private ports</li> </ul> |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Note

This only affects outgoing packets sent via [Send\(\)](#)

For received packets, use [GetSourcePort\(\)](#) to see sender's port

| Port  | Service              | Description                |
|-------|----------------------|----------------------------|
| 53    | DNS                  | Domain Name System         |
| 67/68 | <a href="#">DHCP</a> | Dynamic Host Configuration |
| 80    | HTTP                 | Web Server                 |
| 123   | NTP                  | Network Time Protocol      |
| 161   | SNMP                 | Simple Network Management  |
| 443   | HTTPS                | Secure Web Server          |
| 1883  | <a href="#">MQTT</a> | Message Queue Telemetry    |
| 5353  | mDNS                 | Multicast DNS              |

## Common Well-Known Ports:

## Example Usage:

```
UDPPacket pkt;
pkt.SetDestinationPort(8080); // Custom application port
pkt.AddData("Hello", 5);
pkt.Send(serverIP);
```

## See also

[GetDestinationPort\(\)](#), [GetSourcePort\(\)](#), [Send\(\)](#)

## 16.184.3.25 SetDSCP()

```
void UDPPacket::SetDSCP (
 uint8_t dscp)
```

Set DSCP for QoS (0=best effort, 46=VoIP/EF, 34=video/AF41, 8=bulk/CS1)

Standard DSCP values (RFC 4594):

- 0: Best Effort (default)
- 8: CS1 (bulk data, lower priority)
- 34: AF41 (high priority video)
- 46: EF (VoIP, low latency/jitter)
- 48: CS6 (network control)

```
pkt.SetDSCP(46); // Mark as VoIP for priority treatment
```

## Note

Requires network QoS support to have effect



**16.184.3.26 SetSourcePort()**

```
void UDPPacket::SetSourcePort (
 uint16_t port)
```

Set source port (0=auto-assign ephemeral port, 1024-65535=specific port)

Sets the source UDP port for outgoing packets. The source port identifies which local port the packet originates from and where replies should be sent.

**Parameters**

|             |                                                                                                                                                                                                                                                                                                                                                 |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>port</i> | Source port number: <ul style="list-style-type: none"> <li>• 0 = Auto-assign ephemeral port (recommended for clients)</li> <li>• 1-1023 = Well-known/system ports (avoid unless necessary)</li> <li>• 1024-49151 = Registered ports (safe for servers)</li> <li>• 49152-65535 = Dynamic/private ports (system auto-assignment range)</li> </ul> |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Note**

For CLIENT applications, use 0 to let the stack auto-assign an ephemeral port

For SERVER applications, use a fixed port so clients know where to send

The stack automatically assigns ports from the ephemeral range when port=0

Setting port to 0 is equivalent to not calling this function

| Application Type | Recommended Port        | Reason                              |
|------------------|-------------------------|-------------------------------------|
| UDP Server       | Fixed port (e.g., 5000) | Clients need to know where to send  |
| UDP Client       | 0 (auto-assign)         | Don't care which local port is used |
| Peer-to-Peer     | Fixed port              | Both sides need predictable ports   |
| Reply Packet     | Use 0 or leave default  | Stack handles reply routing         |

**Port Selection Guidelines:****Server vs Client Pattern:**

**Servers** listen on FIXED ports:

- Servers: Use fixed port (e.g., 5000, 8080) so clients know where to connect
- Must be consistent across reboots
- Typically documented/configured value

**Clients** use AUTO-ASSIGNED ports:

- Clients: Use 0 for auto-assignment - you don't care which local port
- Different port for each connection/packet
- Server uses [GetSourcePort\(\)](#) from received packet to reply

**Expand for Example Usage****Examples**

**Example Usage - UDP Server:**

```
// Server application listening on port 5000
void ServerTask(void *pd) {
 UDPPacket pkt;
 pkt.SetSourcePort(5000); // Fixed port - clients send here

 while (1) {
 if (pkt.Receive()) {
 ProcessRequest(pkt);

 // Reply to client
 UDPPacket reply;
 reply.SetDestinationPort(pkt.GetSourcePort()); // Client's port
 reply.AddData("Response", 8);
 reply.Send(pkt.GetSourceAddress());
 }
 }
}
```

**Example Usage - UDP Client:**

```
// Client application sending to server
void SendRequest(IPADDR serverIP, uint16_t serverPort) {
 UDPPacket pkt;
 pkt.SetSourcePort(0); // Auto-assign - don't care which port
 // Or just omit SetSourcePort() entirely - same effect

 pkt.SetDestinationPort(serverPort); // Server's known port
 pkt.AddData("Request", 7);
 pkt.Send(serverIP);
}
```

**Example Usage - Peer-to-Peer:**

```
// Both sides use fixed ports for peer-to-peer communication
#define MY_PORT 6000
#define PEER_PORT 6000

UDPPacket pkt;
pkt.SetSourcePort(MY_PORT); // My fixed port
pkt.SetDestinationPort(PEER_PORT); // Peer's fixed port
pkt.AddData("Hello", 5);
pkt.Send(peerIP);
```

**Warning**

- Using port 0 for servers makes them unreachable - clients won't know the port
- Reusing well-known ports (1-1023) may conflict with system services
- Fixed ports must be available (not already in use by another application)

**See also**

[GetSourcePort\(\)](#), [SetDestinationPort\(\)](#), [Send\(\)](#)

**16.184.3.27 Validate()**

```
BOOL UDPPacket::Validate (
 void)
```

Validate received packet (checks checksum and data presence)

**ALWAYS call after receiving packet!**

This function performs integrity checks on a received packet to ensure:

- The packet checksum is correct (data not corrupted in transit)
- Required data fields are present and populated
- The packet structure is valid

**Returns**

- true if the packet is valid and safe to process
- false if the packet is corrupt, incomplete, or timed out

**Warning**

Failure to call this function before processing packet data may result in processing corrupted or malicious data

**Note**

This function should be called immediately after any packet receive operation

**Example Usage:**

```
if (packet.Validate()) {
 // Safe to process packet data
 ProcessPacketData(packet);
} else {
 // Discard corrupted packet
 printf("Invalid packet received\r\n");
}
```

The documentation for this class was generated from the following file:

- udp.h

## 16.185 UDPPKT Struct Reference

Internal UDP Packet Header Structure.

```
#include <ip.h>
```

**Public Attributes**

- `beuint16_t srcPort`  
*Source port number (0-65535)*
- `beuint16_t dstPort`  
*Destination port number (0-65535)*
- `beuint16_t UdpLen`  
*Datagram length in bytes including UDP header and data (minimum 8 bytes)*
- `uint16_t UdpCSum`  
*UDP checksum (optional in IPv4, mandatory in IPv6)*
- `uint8_t DATA []`  
*Flexible array member - UDP payload data follows header (actual length = UdpLen - 8)*

### 16.185.1 Detailed Description

Internal UDP Packet Header Structure.

Standard UDP header as defined in RFC 768. This structure represents the 8-byte UDP header that precedes the UDP payload data.

The DATA[] flexible array member allows direct access to the UDP payload that follows the header.

**Note**

This structure is packed to ensure proper alignment with network packet data.

UDP header is always 8 bytes long (no options like TCP).

The checksum is optional in IPv4 (can be zero) but mandatory in IPv6.

### 16.185.2 Member Data Documentation

#### 16.185.2.1 UdpCSum

```
uint16_t UDPPKT::UdpCSum
```

UDP checksum (optional in IPv4, mandatory in IPv6)

**Warning**

Do not make big endian - see [IPPKT::hCSum](#) for explanation

The documentation for this struct was generated from the following file:

- ip.h

## 16.186 UniqueIdentifier Class Reference

Get the 128-bit Unique Identifier.

```
#include <bsp.h>
```

### Public Member Functions

- `const uint8_t * GetUniqueIdentifier ()`  
*Will attempt to read the Unique Identifier from flash memory.*
- `UniqueIdentifier ()`  
*Create a [UniqueIdentifier](#) object and read the Unique Identifier from flash memory.*
- `const uint8_t * GetBuffer ()`  
*Get a pointer to the buffer containing the uniqueID.*
- `void Print ()`  
*Print the 128-bit UniqueID in hex to serial.*
- `const uint8_t * GetUniqueIdentifier ()`  
*Will attempt to read the Unique Identifier from flash memory.*
- `UniqueIdentifier ()`  
*Create a [UniqueIdentifier](#) object and read the Unique Identifier from flash memory.*
- `const uint8_t * GetBuffer ()`  
*Get a pointer to the buffer containing the uniqueID.*
- `void Print ()`  
*Print the 128-bit UniqueID in hex to serial.*

### 16.186.1 Detailed Description

Get the 128-bit Unique Identifier.

Each MODM7AE70 is programmed with a 128-bit unique identifier in flash memory. This class exposes functions to get the 128-bit unique ID in a 16-byte array of type `uint8_t`.

Each SBE70LC is programmed with a 128-bit unique identifier in flash memory. This class exposes functions to get the 128-bit unique ID in a 16-byte array of type `uint8_t`.

### 16.186.2 Member Function Documentation

#### 16.186.2.1 [GetBuffer\(\)](#) [1/2]

```
const uint8_t * UniqueIdentifier::GetBuffer () [inline]
```

Get a pointer to the buffer containing the uniqueID.

#### Returns

A pointer to the buffer containing the uniqueID.

#### 16.186.2.2 [GetBuffer\(\)](#) [2/2]

```
const uint8_t * UniqueIdentifier::GetBuffer () [inline]
```

Get a pointer to the buffer containing the uniqueID.

#### Returns

A pointer to the buffer containing the uniqueID.

**16.186.2.3 GetUniqueIdentifier() [1/2]**

```
const uint8_t * UniqueIdentifier::GetUniqueIdentifier ()
```

Will attempt to read the Unique Identifier from flash memory.

**Returns**

If successful, returns a pointer to the buffer containing the uniqueID. Otherwise, returns a nullptr.

**16.186.2.4 GetUniqueIdentifier() [2/2]**

```
const uint8_t * UniqueIdentifier::GetUniqueIdentifier ()
```

Will attempt to read the Unique Identifier from flash memory.

**Returns**

If successful, returns a pointer to the buffer containing the uniqueID. Otherwise, returns a nullptr.

The documentation for this class was generated from the following files:

- MODM7AE70/include/bsp.h
- SBE70LC/include/bsp.h

**16.187 UserAuthManager Class Reference**

The user authorization manager class allows application developers the ability to manage user authorization records. The can be loaded and saved to any storage space, including the config system or UserParams. Authorization values are hashed before being saved. Validation compares both the hash as well as the authorization type. Adding, updating, and removing records will automatically call the user devined save functions. For usage, please see the example found in `examples/SSH/sshServerUserAuth`.

```
#include <UserAuthManager.h>
```

**Public Member Functions**

- **UserAuthManager ()**  
*Default constructor.*
- **~UserAuthManager ()**  
*Default destructor.*
- **bool Init (SaveAuthRecordsFn svRcFn, LoadAuthRecordsFn ldRcFn)**  
*Initialization function. Must be called before use.*
- **bool UserExists (const NBString &userName)**  
*Determines if a user record exists.*
- **AuthResponse AddUserAuth (const NBString &userName, const NBString &auth, AuthType authType)**  
*Attempts to add a user authorization record. This will automatically call the save record function if the user is added.*
- **AuthResponse CheckUserAuth (const NBString &userName, const NBString &auth, AuthType authType)**  
*Checks the for a user and compares the authorization value to what is stored.*
- **AuthResponse CheckUserAuth (const NBString &userName, byte \*auth, AuthType authType)**  
*Checks the for a user and compares the authorization value to what is stored.*
- **AuthResponse UpdateUserAuth (const NBString &userName, const NBString &newAuth, AuthType authType)**  
*Updates a user authorization record with the information provided. This function will automatically save all user records if the changes are successful.*
- **AuthResponse RemoveUserAuth (const NBString &userName)**  
*Remove a user authorization record. This function will automatically save all user records if the removal is successful.*
- **AuthResponse CheckUserAuthLevel (const NBString &userName, uint32\_t authLevel, bool hasAll=true)**  
*Checks the user against the specific authLevel.*
- **AuthResponse SetUserAuthLevel (const NBString &userName, uint32\_t authLevel)**

*This adds the authorization levels passed in to the user's current authorization level.*

- **AuthResponse** **ClrUserAuthLevel** (const **NBString** &userName, uint32\_t authLevel)

*Clears the authorization for the specified user.*

- void **ListUsers** ()

*Lists the users currently in the User Authorization Record system, along with their saved authorization type and level.*

- int **GetMaxAuthRecords** ()

*Gets the maximum number of authorization records available to the system. This can be changed with the macro **MAX\_AUTH\_RECORDS**.*

### 16.187.1 Detailed Description

The user authorization manager class allows application developers the ability to manage user authorization records. The can be loaded and saved to any storage space, including the config system or UserParams. Authorization values are hashed before being saved. Validation compares both the hash as well as the authorization type. Adding, updating, and removing records will automatically call the user devined save functions. For usage, please see the example found in `examples/SSH/sshServerUserAuth`.

### 16.187.2 Member Function Documentation

#### 16.187.2.1 AddUserAuth()

```
AuthResponse UserAuthManager::AddUserAuth (
 const NBString & userName,
 const NBString & auth,
 AuthType authType)
```

Attempts to add a user authorization record. This will automatically call the save record function if the user is added.

##### Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>userName</i> | The username to add.                               |
| <i>auth</i>     | The authorization value to hash and store.         |
| <i>authType</i> | The authorization type of the authorization value. |

##### Return values

|                      |                                                                                                 |
|----------------------|-------------------------------------------------------------------------------------------------|
| <i>eAuthSuccess</i>  | If the user was successfully added.                                                             |
| <i>!eAuthSuccess</i> | If the user was not added successfully. The AuthResponse error code returned will indicate why. |

#### 16.187.2.2 CheckUserAuth() [1/2]

```
AuthResponse UserAuthManager::CheckUserAuth (
 const NBString & userName,
 byte * auth,
 AuthType authType)
```

Checks the for a user and compares the authorization value to what is stored.

##### Parameters

|                 |                                                                                                                     |
|-----------------|---------------------------------------------------------------------------------------------------------------------|
| <i>userName</i> | The username to check against.                                                                                      |
| <i>auth</i>     | The authorization value to compare as hashed value. This will be compared directly to what is stored in the record. |
| <i>authType</i> | The authorization type of the authorization value.                                                                  |

## Return values

|                      |                                                                                                                           |
|----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>eAuthSuccess</i>  | If the user exists and the authorization value matches what was stored                                                    |
| <i>!eAuthSuccess</i> | If the user does not exist or the authorization comparison fails. The AuthResponse error code returned will indicate why. |

**16.187.2.3 CheckUserAuth() [2/2]**

```
AuthResponse UserAuthManager::CheckUserAuth (
 const NBString & userName,
 const NBString & auth,
 AuthType authType)
```

Checks the for a user and compares the authorization value to what is stored.

## Parameters

|                 |                                                                                                          |
|-----------------|----------------------------------------------------------------------------------------------------------|
| <i>userName</i> | The username to check against.                                                                           |
| <i>auth</i>     | The authorization value to compare as a plain string. This value will be hashed before getting compared. |
| <i>authType</i> | The authorization type of the authorization value.                                                       |

## Return values

|                      |                                                                                                                           |
|----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>eAuthSuccess</i>  | If the user exists and the authorization value matches what was stored                                                    |
| <i>!eAuthSuccess</i> | If the user does not exist or the authorization comparison fails. The AuthResponse error code returned will indicate why. |

**16.187.2.4 CheckUserAuthLevel()**

```
AuthResponse UserAuthManager::CheckUserAuthLevel (
 const NBString & userName,
 uint32_t authLevel,
 bool hasAll = true)
```

Checks the user against the specific authLevel.

## Parameters

|                  |                                                                                                                                                                                                                                            |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>userName</i>  | The username to check.                                                                                                                                                                                                                     |
| <i>authLevel</i> | The authorization level to check against. This check compares the bits passed in against the bits set on the user's authLevel.                                                                                                             |
| <i>hasAll</i>    | Used to establish the conditions for the check to pass. If hasAll is true, then every bit passed in must be set on the user's authLevel for the check to pass. If hasAll is false, then only one bit needs to match for the check to pass. |

## Return values

|                      |                                                                      |
|----------------------|----------------------------------------------------------------------|
| <i>eAuthSuccess</i>  | If the check passed and the user has the proper authorization level. |
| <i>!eAuthSuccess</i> | If the user does not have the proper authorization level.            |

### 16.187.2.5 ClrUserAuthLevel()

```
AuthResponse UserAuthManager::ClrUserAuthLevel (
 const NBString & userName,
 uint32_t authLevel)
```

Clears the authorization for the specified user.

#### Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>userName</i>  | The username to check.            |
| <i>authLevel</i> | The authorization level to clear. |

#### Return values

|                      |                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------|
| <i>eAuthSuccess</i>  | If the specified authorization level was successfully cleared and saved.                                           |
| <i>!eAuthSuccess</i> | If the authorization level could not be cleared and saved. The AuthResponse error code returned will indicate why. |

### 16.187.2.6 Init()

```
bool UserAuthManager::Init (
 SaveAuthRecordsFn svRcFn,
 LoadAuthRecordsFn ldRcFn)
```

Initialization function. Must be called before use.

#### Parameters

|               |                                                                                                                         |
|---------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>svRcFn</i> | The user defined function that will save authorization records.                                                         |
| <i>ldRcFn</i> | The user defined function that will load the authorization records from storage. Called inside <a href="#">init()</a> . |

#### Return values

|             |                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------|
| <i>true</i> | If successfully initialized, and the users are properly loaded.                                           |
| <i>!0</i>   | If there was an error initializing the object, or the users were not successfully loaded. Error code will |

### 16.187.2.7 RemoveUserAuth()

```
AuthResponse UserAuthManager::RemoveUserAuth (
 const NBString & userName)
```

Remove a user authorization record. This function will automatically save all user records if the removal is successful.

#### Parameters

|                 |                         |
|-----------------|-------------------------|
| <i>userName</i> | The username to remove. |
|-----------------|-------------------------|

#### Return values

|                      |                                                                                                  |
|----------------------|--------------------------------------------------------------------------------------------------|
| <i>eAuthSuccess</i>  | If the user record was successfully removed.                                                     |
| <i>!eAuthSuccess</i> | If the user record could not be removed. The AuthResponse error code returned will indicate why. |



### 16.187.2.8 SetUserAuthLevel()

```
AuthResponse UserAuthManager::SetUserAuthLevel (
 const NSString & userName,
 uint32_t authLevel)
```

This adds the authorization levels passed in to the user's current authorization level.

#### Parameters

|                  |                                                                                                                                |
|------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <i>userName</i>  | The username to check.                                                                                                         |
| <i>authLevel</i> | The authorization level to check against. This check compares the bits passed in against the bits set on the user's authLevel. |

#### Return values

|                      |                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------|
| <i>eAuthSuccess</i>  | If the the user authorization level was properly set and saved.                                                       |
| <i>!eAuthSuccess</i> | If the user's authorization level could not be set and saved. The AuthResponse error code returned will indicate why. |

### 16.187.2.9 UpdateUserAuth()

```
AuthResponse UserAuthManager::UpdateUserAuth (
 const NSString & userName,
 const NSString & newAuth,
 AuthType authType)
```

Updates a user authorization record with the information provided. This function will automatically save all user records if the changes are successful.

#### Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>userName</i> | The username to check against.                     |
| <i>newAuth</i>  | The new authorization value to assign to the user. |
| <i>authType</i> | The authorization type of the authorization value. |

#### Return values

|                      |                                                                                                  |
|----------------------|--------------------------------------------------------------------------------------------------|
| <i>eAuthSuccess</i>  | If the user record was successfully updated.                                                     |
| <i>!eAuthSuccess</i> | If the user record could not be updated. The AuthResponse error code returned will indicate why. |

### 16.187.2.10 UserExists()

```
bool UserAuthManager::UserExists (
 const NSString & userName)
```

Determines if a user record exists.

#### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>userName</i> | The username to check. |
|-----------------|------------------------|

#### Return values

|              |                             |
|--------------|-----------------------------|
| <i>true</i>  | If the user exists.         |
| <i>false</i> | If the user does not exist. |

---

The documentation for this class was generated from the following file:

- `UserAuthManager.h`

## 16.188 UserAuthRecord Struct Reference

A stored record of a user's authorization credentials. The value is hashed when saved so it can't be read directly. User's can currently only have one authorization record per user name.

```
#include <UserAuthManager.h>
```

### Public Attributes

- [NSString](#) `m_userName`  
*The username for the record. This value must be unique across all records.*
- `uint8_t m_authHash [WC_SHA256_DIGEST_SIZE]`  
*The hashed representation of the authentication value.*
- [AuthType](#) `m_authType`  
*The type of authentication hashes. Currently supported are keys and passwords.*
- `uint32_t m_authLevel`

### 16.188.1 Detailed Description

A stored record of a user's authorization credentials. The value is hashed when saved so it can't be read directly. User's can currently only have one authorization record per user name.

### 16.188.2 Member Data Documentation

#### 16.188.2.1 `m_authLevel`

```
uint32_t UserAuthRecord::m_authLevel
```

The authorization level granted to this user. This is treated as a bit field, and authorization checks look for specific bits being set.

The documentation for this struct was generated from the following file:

- `UserAuthManager.h`

## 16.189 WebConfigServlet Class Reference

Class to run web configurator.

```
#include <config_server.h>
```

Inherits `servlet`.

### 16.189.1 Detailed Description

Class to run web configurator.

The documentation for this class was generated from the following file:

- `config_server.h`

## 16.190 NB::WebSocket Class Reference

[WebSocket](#) connection management class.

```
#include <websockets.h>
```

## Public Member Functions

- int [WriteData](#) (const char \*buf, int nbytes)  
*Write data to the [WebSocket](#) connection.*
- int [ReadData](#) (char \*buf, int nbytes)  
*Read data from the [WebSocket](#) connection.*
- void [Flush](#) ()  
*Flush buffered transmit data to the network.*
- void [Pong](#) (uint32\_t len)  
*Send a pong frame in response to a ping.*
- WS\_State [GetState](#) ()  
*Get the current connection state of the [WebSocket](#).*
- int [Ping](#) (uint32\_t len, uint32\_t \*sentTick)  
*Send a [WebSocket](#) ping frame and record the timestamp.*
- int [WaitForPingReply](#) (uint32\_t timeout)  
*Block waiting for a pong reply to a previously sent ping.*
- int [GetPingReplyTick](#) (uint32\_t \*replyTick)  
*Get the tick count when the last pong was received.*
- int [setoption](#) (int option)  
*Set a socket option on this [WebSocket](#) connection.*
- int [clroption](#) (int option)  
*Clear a socket option on this [WebSocket](#) connection.*
- int [getoption](#) ()  
*Get the current socket options bitmap.*
- int [GetWriteSpace](#) ()  
*Get available transmit buffer space in bytes.*
- void [DumpSock](#) ()  
*Dump detailed state information for this [WebSocket](#) connection.*

## Static Public Member Functions

- static int [ws\\_readwto](#) (int fd, char \*buf, int nbytes, int timeout)  
*Read data from a [WebSocket](#) connection with timeout.*
- static int [ws\\_read](#) (int fd, char \*buf, int nbytes)  
*Read data from a [WebSocket](#) connection.*
- static int [ws\\_write](#) (int fd, const char \*buf, int nbytes)  
*Write data to a [WebSocket](#) connection.*
- static int [ws\\_externalclose](#) (int fd)  
*Close a [WebSocket](#) connection from an external caller.*
- static void [ws\\_flush](#) (int fd)  
*Flush the transmit buffer of a [WebSocket](#) connection.*
- static int [ws\\_setoption](#) (int fd, int option)  
*Set a socket option on a [WebSocket](#) connection.*
- static int [ws\\_clroption](#) (int fd, int option)  
*Clear a socket option on a [WebSocket](#) connection.*
- static int [ws\\_getoption](#) (int fd)  
*Get the current socket options for a [WebSocket](#) connection.*
- static [WebSocket](#) \* [GetWebSocketRecord](#) (int fd)  
*Find a [WebSocket](#) record by its file descriptor.*
- static int [promote\\_tcp\\_ws](#) (int tcp\_fd)  
*Promote a TCP socket to a [WebSocket](#) connection after successful handshake.*
- static void [ws\\_read\\_notify](#) (int fd)

- TCP read notification callback invoked when data is available on the [WebSocket](#).*

  - static void [ws\\_write\\_notify](#) (int fd)

*TCP write notification callback invoked when buffer space becomes available.*
- static void [GetFlushTime](#) ()

*Updates the global flush timing for [WebSocket](#) message delivery.*
- static void [RunSkippedCallBack](#) ()

*Executes the callback function when a scheduled run is skipped.*
- static int [Connect](#) (const char \*host, const char \*resource, const char \*protocol, int portnum=80, bool useSSL=false)

*Connect to a [WebSocket](#) server using hostname with optional protocol negotiation.*
- static int [Connect](#) (const char \*host, const char \*resource, int portnum=80, bool useSSL=false)

*Connect to a [WebSocket](#) server using hostname without protocol negotiation.*
- static int [Connect](#) (IPADDR host, const char \*resource, const char \*protocol, int portnum=80, bool useSSL=false)

*Connect to a [WebSocket](#) server using IP address with optional protocol negotiation.*
- static int [Connect](#) (IPADDR host, const char \*resource, int portnum=80, bool useSSL=false)

*Connect to a [WebSocket](#) server using IP address without protocol negotiation.*
- static void [DumpSockets](#) ()

*Dump detailed state information for all active [WebSocket](#) connections.*
- static int [GetFreeWebSocketCount](#) ()

*Get the number of available [WebSocket](#) connection slots.*

## Public Attributes

- int [m\\_fd\\_tcp](#)
- TCP socket file descriptor (underlying transport)*
- int [m\\_fd\\_ws](#)
- [WebSocket](#) file descriptor (application-level identifier)*
- OS\_CRIT [socket\\_crit](#)
- Critical section for thread-safe socket operations.*

## Static Public Attributes

- static int [s\\_openSocketCount](#)
- Number of currently open [WebSocket](#) connections.*
- static [WebSocket](#) WSSockStructs [[WS\\_MAX\\_SOCKETS](#)]
- Static array of [WebSocket](#) connection structures.*

## 16.190.1 Detailed Description

[WebSocket](#) connection management class.

Handles [WebSocket](#) protocol implementation including frame parsing, connection state, and bidirectional communication over TCP. Supports both client and server roles.

```
{
```

## 16.190.2 Member Function Documentation

### 16.190.2.1 clroption()

```
int NB::WebSocket::clroption (
 int option)
```

Clear a socket option on this [WebSocket](#) connection.

## Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>option</i> | The socket option to clear (e.g., WS_SO_TEXT, TCP_NODELAY, SO_KEEPALIVE) |
|---------------|--------------------------------------------------------------------------|

## Returns

int Returns 0 on success, negative value on failure

This member function clears (disables) a specific socket option on the [WebSocket](#) connection. It is the counterpart to [setoption\(\)](#) and allows you to disable previously enabled options or ensure certain options are not active. Clearing an option returns the socket to its default behavior for that particular setting. This is useful when you need to:

- Dynamically adjust socket behavior based on runtime conditions
- Revert configuration changes
- Switch between mutually exclusive options (e.g., text vs binary mode)
- Restore default settings

Clearing an option that was never set has no effect and typically succeeds.

## Note

Some options may not be clearable once set, depending on the underlying socket implementation. Check the return value to verify success.

## See also

[setoption\(\)](#) for setting socket options

[getoption\(\)](#) for querying current option state

## Example Usage

---

Expand for Example Usage

## Examples

## Disable Binary Mode

```
WebSocket *ws = GetActiveWebSocket();

// Clear binary mode, revert to default (text mode)
if (ws->clroption(WS_SO_BINARY) == 0) {
 printf("Binary mode disabled, reverted to text mode\n");
} else {
 printf("Failed to clear binary mode\n");
}
```

## Disable TCP Keepalive

```
WebSocket *ws = GetActiveWebSocket();

// Disable keepalive (maybe to reduce network traffic)
if (ws->clroption(SO_KEEPALIVE) == 0) {
 printf("TCP keepalive disabled\n");
} else {
 printf("Failed to disable keepalive\n");
}
```

## Re-enable Nagle's Algorithm

```
void RestoreBufferingMode(WebSocket *ws) {
 // Clear TCP_NODELAY to re-enable Nagle's algorithm
 if (ws->clroption(TCP_NODELAY) == 0) {
 printf("Nagle's algorithm re-enabled (buffering mode)\n");
 } else {
 printf("WARNING: Could not re-enable buffering\n");
 }
}
```

### Toggle WebSocket Mode

```
void ToggleWebSocketMode(WebSocket *ws, bool enable_binary) {
 if (enable_binary) {
 ws->clroption(WS_SO_TEXT);
 ws->setoption(WS_SO_BINARY);
 printf("Switched to binary mode\n");
 } else {
 ws->clroption(WS_SO_BINARY);
 ws->setoption(WS_SO_TEXT);
 printf("Switched to text mode\n");
 }
}
```

### Reset to Default Configuration

```
void ResetWebSocketOptions(WebSocket *ws) {
 printf("Resetting WebSocket to default options...\n");

 // Clear all custom options
 ws->clroption(WS_SO_BINARY);
 ws->clroption(WS_SO_COMPRESS);
 ws->clroption(TCP_NODELAY);
 ws->clroption(SO_KEEPALIVE);

 // Set default text mode
 ws->setoption(WS_SO_TEXT);

 printf("WebSocket reset to defaults\n");
}
```

### Conditional Option Clearing

```
void AdjustWebSocketForPowerMode(WebSocket *ws, bool low_power_mode) {
 if (low_power_mode) {
 // Disable features that increase power consumption
 ws->clroption(SO_KEEPALIVE);
 ws->clroption(TCP_NODELAY);
 printf("Low power mode: disabled keepalive and nodelay\n");
 } else {
 // Re-enable performance features
 ws->setoption(SO_KEEPALIVE);
 ws->setoption(TCP_NODELAY);
 printf("Normal power mode: enabled keepalive and nodelay\n");
 }
}
```

### Clear All Performance Options

```
void DisablePerformanceOptimizations(WebSocket *ws) {
 printf("Disabling performance optimizations...\n");

 int result = 0;
 result |= ws->clroption(TCP_NODELAY);
 result |= ws->clroption(WS_SO_COMPRESS);

 if (result == 0) {
 printf("All performance options cleared\n");
 } else {
 printf("WARNING: Some options failed to clear\n");
 }
}
```

### Cleanup Before Reconnection

```
void PrepareForReconnect(WebSocket *ws) {
 // Clear all custom options before closing
 ws->clroption(WS_SO_BINARY);
 ws->clroption(TCP_NODELAY);
 ws->clroption(SO_KEEPALIVE);
 ws->clroption(WS_SO_COMPRESS);

 printf("Options cleared, ready for reconnection\n");
}
```

## 16.190.2.2 Connect() [1/4]

```
static int NB::WebSocket::Connect (
 const char * host,
 const char * resource,
 const char * protocol,
```

```
int portnum = 80,
bool useSSL = false) [static]
```

Connect to a [WebSocket](#) server using hostname with optional protocol negotiation.

Establishes a [WebSocket](#) client connection to the specified server by hostname, performing DNS resolution, TCP connection, HTTP upgrade handshake, and optional subprotocol negotiation. This is the primary method for creating client-side [WebSocket](#) connections.

### 16.190.3 Connection Process

1. Resolve hostname to IP address via DNS
2. Establish TCP connection to server:port
3. Send HTTP upgrade request with [WebSocket](#) headers
4. Negotiate subprotocol if specified
5. Complete handshake and upgrade to [WebSocket](#) protocol
6. Return [WebSocket](#) file descriptor on success

### 16.190.4 Parameters

#### Parameters

|                 |                                                                                                                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>host</i>     | Hostname or domain name of the <a href="#">WebSocket</a> server (e.g., "ws.example.com"). Must be a null-terminated string. DNS resolution is performed automatically.                           |
| <i>resource</i> | <a href="#">WebSocket</a> endpoint path (e.g., "/api/v1/stream", "/ws", "/chat"). Must start with "/" and be null-terminated. This is the path portion of the <a href="#">WebSocket</a> URL.     |
| <i>protocol</i> | Optional subprotocol to negotiate (e.g., "chat", "mqtt", "json"). Pass NULL for no subprotocol negotiation. If specified, the server must support and accept this protocol during the handshake. |
| <i>portnum</i>  | TCP port number for the <a href="#">WebSocket</a> connection (default: 80 for ws://, 443 for wss://). Standard ports are 80 (unencrypted) and 443 (SSL/TLS).                                     |
| <i>useSSL</i>   | Set true to use <a href="#">WebSocket</a> Secure (wss://) with TLS encryption, false for unencrypted <a href="#">WebSocket</a> (ws://). Default is false.                                        |

### 16.190.5 Return Value

Returns a [WebSocket](#) file descriptor on success (positive integer), or a negative value on failure:

- Positive value: Valid [WebSocket](#) file descriptor for use with WSRead(), WSWrite(), etc.
- Negative value: Connection failed (DNS resolution, TCP connection, or handshake failure)

### 16.190.6 Connection Failures

Common failure scenarios:

- DNS resolution failure (hostname not found)
- TCP connection timeout or refused
- HTTP upgrade rejected by server (non-101 response)
- Protocol negotiation failure (requested protocol not supported)
- No available [WebSocket](#) slots (WS\_MAX\_SOCKS limit reached)
- Invalid handshake response from server

### 16.190.7 Usage Notes

- The returned file descriptor must be closed with [close\(\)](#) when done
- Use [WSRead\(\)](#) and [WSWrite\(\)](#) for data exchange after successful connection
- Check available slots with [GetFreeWebSocketCount\(\)](#) before connecting
- Connection is blocking; consider running in a separate task for non-blocking behavior
- For wss:// (useSSL=true), ensure SSL/TLS libraries are linked and configured
- Subprotocol is optional; many [WebSocket](#) servers don't require one
- Default port 80 is typical for ws://, use 443 for wss://

---

### Expand for Example Usage

#### Examples

##### Example 1: Basic WebSocket Client Connection

```
#include <websockets.h>

void WebSocketClientTask(void *pd) {
 // Connect to WebSocket server
 int wsFd = NB::WebSocket::Connect("echo.websocket.org",
 "/",
 NULL, // No protocol
 80, // Standard port
 false); // No SSL

 if (wsFd < 0) {
 printf("Failed to connect to WebSocket server\r\n");
 return;
 }

 printf("Connected! WebSocket fd: %d\r\n", wsFd);

 // Send a message
 const char *msg = "Hello WebSocket!";
 WSWrite(wsFd, msg, strlen(msg));

 // Read response
 char buffer[256];
 int len = WSRead(wsFd, buffer, sizeof(buffer) - 1, TICKS_PER_SECOND * 5);
 if (len > 0) {
 buffer[len] = '\0';
 printf("Received: %s\r\n", buffer);
 }

 // Clean up
 close(wsFd);
 printf("Connection closed\r\n");
}
```

##### Example 2: Secure WebSocket Connection (WSS)

```
void SecureWebSocketClient(void *pd) {
 // Connect using WSS (WebSocket Secure)
 int wsFd = NB::WebSocket::Connect("api.example.com",
 "/v1/secure-stream",
 "json-api", // Subprotocol
 443, // HTTPS port
 true); // Enable SSL/TLS

 if (wsFd < 0) {
 printf("Secure connection failed\r\n");
 return;
 }

 printf("Secure WebSocket connected\r\n");

 // Send JSON data
 const char *jsonMsg = "{\"type\":\"subscribe\",\"channel\":\"updates\"}";
 ws_setopt(wsFd, WS_SO_TEXT); // Text mode for JSON
 WSWrite(wsFd, jsonMsg, strlen(jsonMsg));

 // Receive updates
 char rxBuffer[512];
 while (1) {
 int len = WSRead(wsFd, rxBuffer, sizeof(rxBuffer) - 1, TICKS_PER_SECOND * 30);
 }
```



```

 if (len > 0) {
 rxBuffer[len] = '\0';
 printf("Update: %s\r\n", rxBuffer);
 } else {
 printf("Connection closed or timeout\r\n");
 break;
 }
 }

 close(wsFd);
}

```

### Example 3: Connection with Error Handling

```

int ConnectWithRetry(const char *host, const char *resource, int maxRetries) {
 int wsFd = -1;
 int retryCount = 0;

 while (retryCount < maxRetries) {
 // Check available slots
 int freeSlots = NB::WebSocket::GetFreeWebSocketCount();
 if (freeSlots <= 0) {
 printf("No WebSocket slots available, waiting...\r\n");
 OSTimeDly(TICKS_PER_SECOND * 5);
 continue;
 }

 printf("Attempt %d/%d: Connecting to %s%s...\r\n",
 retryCount + 1, maxRetries, host, resource);

 wsFd = NB::WebSocket::Connect(host, resource, NULL, 80, false);

 if (wsFd >= 0) {
 printf("Successfully connected!\r\n");
 return wsFd;
 }

 printf("Connection failed, retrying in 5 seconds...\r\n");
 retryCount++;
 OSTimeDly(TICKS_PER_SECOND * 5);
 }

 printf("Failed to connect after %d attempts\r\n", maxRetries);
 return -1;
}

// Usage
int wsFd = ConnectWithRetry("ws.server.com", "/api/stream", 3);
if (wsFd >= 0) {
 // Use WebSocket connection
 close(wsFd);
}

```

### Example 4: Multiple Server Connections

```

struct ServerEndpoint {
 const char *host;
 const char *resource;
 const char *protocol;
 int port;
 bool useSSL;
};

void ConnectMultipleServers(void *pd) {
 ServerEndpoint servers[] = {
 {"telemetry.example.com", "/data", "json", 80, false},
 {"api.example.com", "/events", "mqtt", 443, true},
 {"stream.example.com", "/live", NULL, 80, false}
 };

 int connections[3];
 int connectedCount = 0;

 for (int i = 0; i < 3; i++) {
 printf("Connecting to %s%s...\r\n",
 servers[i].host, servers[i].resource);

 connections[i] = NB::WebSocket::Connect(servers[i].host,
 servers[i].resource,
 servers[i].protocol,
 servers[i].port,
 servers[i].useSSL);
 }
}

```

```

 if (connections[i] >= 0) {
 printf(" Connected (fd %d)\r\n", connections[i]);
 connectedCount++;
 } else {
 printf(" Connection failed\r\n");
 }
 }

 printf("Successfully connected to %d/%d servers\r\n",
 connectedCount, 3);

 // Use connections...

 // Clean up
 for (int i = 0; i < 3; i++) {
 if (connections[i] >= 0) {
 close(connections[i]);
 }
 }
}

```

### Example 5: Protocol Negotiation

```

void ConnectWithProtocol(void *pd) {
 const char *protocols[] = {"chat", "json-rpc", "mqtt"};
 int wsFd = -1;

 // Try each protocol until one succeeds
 for (int i = 0; i < 3; i++) {
 printf("Attempting connection with protocol: %s\r\n", protocols[i]);

 wsFd = NB::WebSocket::Connect("api.example.com",
 "/ws",
 protocols[i],
 80,
 false);

 if (wsFd >= 0) {
 printf("Connected with protocol: %s\r\n", protocols[i]);
 break;
 }

 printf("Protocol %s not supported, trying next...\r\n", protocols[i]);
 }

 if (wsFd < 0) {
 printf("No supported protocol found\r\n");
 return;
 }

 // Use connection with negotiated protocol
 // ...

 close(wsFd);
}

```

### Example 6: Non-blocking Connection in Task

```

void AsyncConnectTask(void *pd) {
 const char *host = "stream.example.com";
 const char *resource = "/live";

 printf("Background connection task started\r\n");

 // This blocks but runs in separate task
 int wsFd = NB::WebSocket::Connect(host, resource, NULL, 80, false);

 if (wsFd >= 0) {
 printf("Background connection established\r\n");

 // Store fd in global or notify main task
 // ... handle communication ...

 close(wsFd);
 } else {
 printf("Background connection failed\r\n");
 }
}

void StartAsyncConnection() {
 // Create task for non-blocking connection
 OSSimpleTaskCreateWithName(AsyncConnectTask,
 MAIN_PRIO - 1,

```

```

 NULL,
 "WSConnect");

 // Main thread continues without blocking
 printf("Connection initiated in background\r\n");
}

```

**See also**

[Connect\(IPADDR, const char \\*, const char \\*, int, bool\)](#) - [Connect](#) using IP address

[WSRead\(\)](#) - Read data from [WebSocket](#) connection

[WSWrite\(\)](#) - Write data to [WebSocket](#) connection

[close\(\)](#) - Close [WebSocket](#) connection

[GetFreeWebSocketCount\(\)](#) - Check available connection slots

**Note**

DNS resolution may take several seconds; consider timeout handling

Connection blocks until complete or failure; use separate task for async behavior

Ensure WS\_MAX\_SOCKS has available slots before connecting

For production systems, implement connection retry logic with exponential backoff

**16.190.7.1 Connect() [2/4]**

```

static int NB::WebSocket::Connect (
 const char * host,
 const char * resource,
 int portnum = 80,
 bool useSSL = false) [inline], [static]

```

Connect to a [WebSocket](#) server using hostname without protocol negotiation.

Convenience overload that establishes a [WebSocket](#) client connection without specifying a subprotocol. This is the most common connection method as many [WebSocket](#) servers don't require subprotocol negotiation.

**16.190.8 Parameters****Parameters**

|                 |                                                                                          |
|-----------------|------------------------------------------------------------------------------------------|
| <i>host</i>     | Hostname or domain name of the <a href="#">WebSocket</a> server (e.g., "ws.example.com") |
| <i>resource</i> | <a href="#">WebSocket</a> endpoint path (e.g., "/ws", "/api/stream")                     |
| <i>portnum</i>  | TCP port number (default: 80 for ws://, 443 for wss://)                                  |
| <i>useSSL</i>   | Set true for wss:// (encrypted), false for ws:// (default: false)                        |

**16.190.9 Return Value**

Returns [WebSocket](#) file descriptor (positive) on success, negative on failure.

**See also**

[Connect\(const char \\*, const char \\*, const char \\*, int, bool\)](#) - Full version with protocol

**Note**

This is equivalent to calling `Connect(host, resource, NULL, portnum, useSSL)`

Most commonly used overload for simple [WebSocket](#) connections

### 16.190.9.1 Connect() [3/4]

```
static int NB::WebSocket::Connect (
 IPADDR host,
 const char * resource,
 const char * protocol,
 int portnum = 80,
 bool useSSL = false) [static]
```

Connect to a [WebSocket](#) server using IP address with optional protocol negotiation.

Establishes a [WebSocket](#) client connection using a direct IP address, bypassing DNS resolution. This is useful for local networks, static IPs, or when DNS is unavailable. Otherwise identical to the hostname-based Connect method.

### 16.190.10 Connection Process

1. Use provided IP address directly (no DNS lookup)
2. Establish TCP connection to IP:port
3. Send HTTP upgrade request with Host header
4. Negotiate subprotocol if specified
5. Complete handshake and upgrade to [WebSocket](#)
6. Return [WebSocket](#) file descriptor on success

### 16.190.11 Parameters

#### Parameters

|                 |                                                                                                                                                                                                                                      |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>host</i>     | IP address of the <a href="#">WebSocket</a> server as IPADDR type. Can be constructed using AsciiToIp() for dotted notation or direct IP address types. Example: AsciiToIp("192.168.1.100") or <a href="#">IPADDR(192,168,1,100)</a> |
| <i>resource</i> | <a href="#">WebSocket</a> endpoint path (e.g., "/api/v1/stream", "/ws")                                                                                                                                                              |
| <i>protocol</i> | Optional subprotocol to negotiate (NULL for none)                                                                                                                                                                                    |
| <i>portnum</i>  | TCP port number (default: 80 for ws://, 443 for wss://)                                                                                                                                                                              |
| <i>useSSL</i>   | Set true for wss:// (encrypted), false for ws:// (default: false)                                                                                                                                                                    |

### 16.190.12 Return Value

Returns [WebSocket](#) file descriptor (positive) on success, negative on failure.

### 16.190.13 Advantages of IP Address Connection

- Faster connection (no DNS resolution delay)
- Works without DNS server availability
- Deterministic connection to specific server
- Useful for local network devices with static IPs
- Avoids DNS caching issues during development

---

#### Expand for Example Usage

#### Examples

**Example 1: Connect to Local WebSocket Server**

```

void ConnectToLocalServer(void *pd) {
 // Connect to WebSocket server at 192.168.1.100
 IPADDR serverIP = AsciiToIp("192.168.1.100");

 int wsFd = NB::WebSocket::Connect(serverIP,
 "/api/data",
 NULL, // No protocol
 8080, // Custom port
 false); // No SSL

 if (wsFd < 0) {
 printf("Connection to %s failed\r\n",
 IpToString(serverIP));
 return;
 }

 printf("Connected to server at %s:%d\r\n",
 IpToString(serverIP), 8080);

 // Use connection...

 close(wsFd);
}

```

**Example 2: Connect Using Direct IP Construction**

```

void ConnectDirectIP(void *pd) {
 // Construct IP address directly
 IPADDR serverIP(192, 168, 1, 50);

 int wsFd = NB::WebSocket::Connect(serverIP,
 "/ws",
 "telemetry",
 80,
 false);

 if (wsFd >= 0) {
 printf("Connected to 192.168.1.50\r\n");

 // Send telemetry data
 char data[128];
 snprintf(data, sizeof(data),
 "{\"temp\":%.1f,\"time\":%lu}", 25.5f, Secs);
 ws_setopt(wsFd, WS_SO_TEXT);
 WSWrite(wsFd, data, strlen(data));

 close(wsFd);
 }
}

```

**Example 3: Fallback from Hostname to IP**

```

int ConnectWithFallback(const char *hostname, const char *ipStr,
 const char *resource) {
 int wsFd;

 // Try hostname first
 printf("Attempting connection to %s...\r\n", hostname);
 wsFd = NB::WebSocket::Connect(hostname, resource, NULL, 80, false);

 if (wsFd >= 0) {
 printf("Connected via hostname\r\n");
 return wsFd;
 }

 // Fallback to IP address
 printf("Hostname failed, trying IP address %s...\r\n", ipStr);
 IPADDR serverIP = AsciiToIp(ipStr);
 wsFd = NB::WebSocket::Connect(serverIP, resource, NULL, 80, false);

 if (wsFd >= 0) {
 printf("Connected via IP address\r\n");
 return wsFd;
 }

 printf("All connection attempts failed\r\n");
 return -1;
}

// Usage
int wsFd = ConnectWithFallback("server.local", "192.168.1.100", "/ws");

```

**Example 4: Connect to Multiple Devices by IP**

```

struct Device {
 const char *name;
 IPADDR ip;
 int wsFd;
};

void ConnectToDeviceArray(Device *devices, int count) {
 for (int i = 0; i < count; i++) {
 printf("Connecting to %s at %s...\r\n",
 devices[i].name, IpToString(devices[i].ip));

 devices[i].wsFd = NB::WebSocket::Connect(devices[i].ip,
 "/sensor",
 "iot-proto",
 80,
 false);

 if (devices[i].wsFd >= 0) {
 printf(" %s: Connected (fd %d)\r\n",
 devices[i].name, devices[i].wsFd);
 } else {
 printf(" %s: Connection failed\r\n", devices[i].name);
 }
 }
}

// Usage
Device sensors[] = {
 {"Temperature", AsciiToIp("192.168.1.10"), -1},
 {"Humidity", AsciiToIp("192.168.1.11"), -1},
 {"Pressure", AsciiToIp("192.168.1.12"), -1}
};
ConnectToDeviceArray(sensors, 3);

```

**Example 5: Dynamic IP Address Connection**

```

void ConnectDynamicIP(void *pd) {
 // Get IP address from configuration or discovery
 char ipString[16];
 GetServerIPFromConfig(ipString, sizeof(ipString));

 IPADDR serverIP = AsciiToIp(ipString);

 if (serverIP == IPADDR::NullIP()) {
 printf("Invalid IP address: %s\r\n", ipString);
 return;
 }

 printf("Connecting to dynamically configured server: %s\r\n",
 IpToString(serverIP));

 int wsFd = NB::WebSocket::Connect(serverIP, "/api", NULL, 80, false);

 if (wsFd >= 0) {
 printf("Connection successful\r\n");
 // Use connection...
 close(wsFd);
 } else {
 printf("Connection failed\r\n");
 }
}

```

**See also**

[Connect\(const char \\*, const char \\*, const char \\*, int, bool\)](#) - [Connect](#) using hostname

[IPADDR](#) - NetBurner IP address type

[AsciiToIp\(\)](#) - Convert dotted notation string to [IPADDR](#)

[IpToString\(\)](#) - Convert [IPADDR](#) to string for display

**Note**

No DNS lookup performed; connection is faster than hostname-based

Useful for local networks, embedded systems, and static configurations

Host header in HTTP upgrade still requires a hostname; IP is used for connection only

### 16.190.13.1 Connect() [4/4]

```
static int NB::WebSocket::Connect (
 IPADDR host,
 const char * resource,
 int portnum = 80,
 bool useSSL = false) [inline], [static]
```

Connect to a [WebSocket](#) server using IP address without protocol negotiation.

Convenience overload that establishes a [WebSocket](#) client connection using an IP address without specifying a subprotocol. Most commonly used for local network connections or when the server doesn't require protocol negotiation.

### 16.190.14 Parameters

#### Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>host</i>     | IP address of the <a href="#">WebSocket</a> server (IPADDR type)     |
| <i>resource</i> | <a href="#">WebSocket</a> endpoint path (e.g., "/ws", "/api/stream") |
| <i>portnum</i>  | TCP port number (default: 80 for ws://, 443 for wss://)              |
| <i>useSSL</i>   | Set true for wss:// (encrypted), false for ws:// (default: false)    |

### 16.190.15 Return Value

Returns [WebSocket](#) file descriptor (positive) on success, negative on failure.

See also

[Connect\(IPADDR, const char \\*, const char \\*, int, bool\)](#) - Full version with protocol

#### Note

This is equivalent to calling `Connect(host, resource, NULL, portnum, useSSL)`

Most commonly used for local network [WebSocket](#) connections

### 16.190.15.1 DumpSock()

```
void NB::WebSocket::DumpSock ()
```

Dump detailed state information for this [WebSocket](#) connection.

Prints comprehensive diagnostic information about the current [WebSocket](#) connection to the debug console, including connection state, buffer levels, file descriptors, timing information, and internal state variables. Useful for debugging connection issues, buffer management, and protocol state.

### 16.190.16 Displayed Information

Typical output includes:

- File descriptors (TCP and [WebSocket](#))
- Connection state (INIT, ESTABLISHED, CLOSING, etc.)
- Buffer levels (RX, TX, Control)
- Buffer statistics (used/available space)
- Socket options (text/binary mode)
- Timing information (last send, accumulation delay)
- Frame processing state
- Ping/pong state
- Masking keys and payload lengths

## 16.190.17 Usage Scenarios

- Debugging connection establishment issues
- Diagnosing buffer exhaustion problems
- Monitoring connection health and state
- Troubleshooting frame fragmentation
- Analyzing performance bottlenecks
- Verifying protocol state transitions
- Development and testing

---

### Expand for Example Usage

#### Examples

##### Example 1: Basic Connection Diagnostics

```
void DiagnoseConnection(int wsFd) {
 // Get WebSocket record
 NB::WebSocket *ws = NB::WebSocket::GetWebSocketRecord(wsFd);

 if (ws == NULL) {
 printf("Invalid WebSocket fd: %d\r\n", wsFd);
 return;
 }

 printf("=== WebSocket Diagnostics ===\r\n");
 ws->DumpSock();
 printf("=====\r\n");
}
```

##### Example 2: Debug Connection Issues

```
void DebugWebSocketCommunication(int wsFd) {
 NB::WebSocket *ws = NB::WebSocket::GetWebSocketRecord(wsFd);

 if (ws == NULL) return;

 // Check state before operation
 printf("Before write:\r\n");
 ws->DumpSock();

 // Attempt write
 const char *msg = "Test message";
 int result = WSWrite(wsFd, msg, strlen(msg));

 // Check state after operation
 printf("\nAfter write (result: %d):\r\n", result);
 ws->DumpSock();

 // Check buffer space
 int space = ws->GetWriteSpace();
 printf("\nAvailable TX buffer space: %d bytes\r\n", space);
}
```

##### Example 3: Monitor Connection Health

```
void MonitorConnectionHealth(int wsFd) {
 NB::WebSocket *ws = NB::WebSocket::GetWebSocketRecord(wsFd);

 if (ws == NULL) return;

 while (1) {
 // Periodic health check
 OSTimeDly(TICKS_PER_SECOND * 10);

 printf("\n=== Connection Health at %lu seconds ===\r\n", Secs);
 ws->DumpSock();

 // Check state
 NB::WebSocket::WS_State state = ws->GetState();
 if (state != NB::WebSocket::WS_ESTABLISHED) {
 printf("WARNING: Connection not established!\r\n");
 break;
 }
 }
}
```



```

 // Check buffer levels
 int space = ws->GetWriteSpace();
 if (space < 100) {
 printf("WARNING: Low TX buffer space (%d bytes)\r\n", space);
 }
 }
}

```

#### Example 4: Debugging Message Send Failure

```

void DebugSendFailure(int wsFd, const char *data, int len) {
 NB::WebSocket *ws = NB::WebSocket::GetWebSocketRecord(wsFd);

 if (ws == NULL) {
 printf("Invalid WebSocket\r\n");
 return;
 }

 printf("Attempting to send %d bytes...\r\n", len);

 // Check pre-send state
 int spaceBefore = ws->GetWriteSpace();
 printf("TX buffer space before: %d bytes\r\n", spaceBefore);

 if (spaceBefore < len) {
 printf("ERROR: Insufficient buffer space!\r\n");
 printf("Detailed state:\r\n");
 ws->DumpSock();
 return;
 }

 // Attempt send
 int sent = WSWrite(wsFd, data, len);

 if (sent < 0) {
 printf("Send failed! Dumping state:\r\n");
 ws->DumpSock();
 } else if (sent < len) {
 printf("Partial send: %d/%d bytes\r\n", sent, len);
 ws->DumpSock();
 } else {
 printf("Send successful\r\n");
 }
}

```

#### Example 5: Connection State Verification

```

void VerifyConnectionState(int wsFd, const char *operationName) {
 NB::WebSocket *ws = NB::WebSocket::GetWebSocketRecord(wsFd);

 if (ws == NULL) {
 printf("%s: Invalid WebSocket fd\r\n", operationName);
 return;
 }

 NB::WebSocket::WS_State state = ws->GetState();

 printf("\n=== %s: State Verification ===\r\n", operationName);
 ws->DumpSock();

 switch (state) {
 case NB::WebSocket::WS_ESTABLISHED:
 printf("Status: OK - Connection ready\r\n");
 break;

 case NB::WebSocket::WS_CONNECTING:
 printf("Status: WARNING - Still connecting\r\n");
 break;

 case NB::WebSocket::WS_CLOSE_PENDING:
 case NB::WebSocket::WS_CLOSE_SENT:
 case NB::WebSocket::WS_CLOSE_RECV:
 printf("Status: WARNING - Connection closing\r\n");
 break;

 case NB::WebSocket::WS_CLOSED:
 case NB::WebSocket::WS_FAIL:
 printf("Status: ERROR - Connection closed/failed\r\n");
 break;

 default:
 printf("Status: UNKNOWN - Unexpected state\r\n");
 }
}

```

```
 break;
 }
}
```

---

#### See also

[DumpSockets\(\)](#) - Dump state of all [WebSocket](#) connections  
[GetState\(\)](#) - Get current connection state  
[GetWriteSpace\(\)](#) - Get available TX buffer space  
[GetWebSocketRecord\(\)](#) - Get [WebSocket](#) object from file descriptor

#### Note

Output is sent to debug console (printf/serial)  
Useful during development and testing  
Can be called at any time during connection lifetime  
For production code, consider removing or conditionally compiling debug calls  
Not thread-safe; use appropriate synchronization if called from multiple tasks

#### 16.190.17.1 DumpSockets()

```
static void NB::WebSocket::DumpSockets () [static]
```

Dump detailed state information for all active [WebSocket](#) connections.

Prints comprehensive diagnostic information for all [WebSocket](#) connections in the WSSockStructs array to the debug console. This includes both active and inactive slots, showing the overall state of the [WebSocket](#) subsystem. Useful for system-wide debugging and monitoring.

#### 16.190.18 Displayed Information

For each [WebSocket](#) slot (0 to WS\_MAX\_SOCKS-1):

- Slot index and active status
- File descriptors (TCP and [WebSocket](#))
- Connection state
- Buffer levels and statistics
- Socket options and configuration
- Timing and protocol state
- Summary of active vs free connections

#### 16.190.19 Usage Scenarios

- System-wide connection monitoring
- Debugging connection leaks or exhaustion
- Verifying connection cleanup
- Analyzing resource usage
- Development and testing of multi-connection applications
- Troubleshooting "no available slots" errors
- System health checks

---

#### Expand for Example Usage Examples

**Example 1: System Health Check**

```

void SystemHealthCheck(void *pd) {
 while (1) {
 // Periodic system-wide diagnostic
 OSTimeDly(TICKS_PER_SECOND * 60);

 printf("\n=====r\n");
 printf("WebSocket System Status at %lu seconds\r\n", Secs);
 printf("=====r\n");

 NB::WebSocket::DumpSockets();

 int freeCount = NB::WebSocket::GetFreeWebSocketCount();
 printf("\nSummary: %d free slots, %d in use\r\n",
 freeCount, WS_MAX_SOCKS - freeCount);
 printf("=====r\n\n");
 }
}

```

**Example 2: Debug Connection Exhaustion**

```

void DebugConnectionExhaustion() {
 int freeSlots = NB::WebSocket::GetFreeWebSocketCount();

 if (freeSlots <= 0) {
 printf("n!!! NO WEBSOCKET SLOTS AVAILABLE !!!r\n");
 printf("Dumping all connections to identify issue:r\n\n");

 NB::WebSocket::DumpSockets();

 printf("\nPossible causes:r\n");
 printf("- Connections not being closed properly\r\n");
 printf("- Connection leaks in application code\r\n");
 printf("- WS_MAX_SOCKS (%d) too low for application needs\r\n",
 WS_MAX_SOCKS);
 printf("- Connections stuck in non-terminal states\r\n");
 } else {
 printf("%d WebSocket slots available\r\n", freeSlots);
 }
}

```

**Example 3: Monitor Connection Lifecycle**

```

void MonitorConnectionLifecycle() {
 printf("Initial state:r\n");
 NB::WebSocket::DumpSockets();

 // Create some connections
 int ws1 = NB::WebSocket::Connect("server1.com", "/ws", 80);
 printf("\nAfter connecting to server1:r\n");
 NB::WebSocket::DumpSockets();

 int ws2 = NB::WebSocket::Connect("server2.com", "/api", 80);
 printf("\nAfter connecting to server2:r\n");
 NB::WebSocket::DumpSockets();

 // Use connections...
 OSTimeDly(TICKS_PER_SECOND * 5);

 // Close first connection
 close(ws1);
 printf("\nAfter closing server1 connection:r\n");
 NB::WebSocket::DumpSockets();

 // Close second connection
 close(ws2);
 printf("\nAfter closing server2 connection:r\n");
 NB::WebSocket::DumpSockets();
}

```

**Example 4: Find Stuck Connections**

```

void FindStuckConnections() {
 printf("Searching for stuck connections...r\n\n");

 NB::WebSocket::DumpSockets();

 // Analyze the output
 printf("\nAnalyzing connection states:r\n");
}

```

```

for (int i = 0; i < WS_MAX_SOCKS; i++) {
 NB::WebSocket *ws = &NB::WebSocket::WSSockStructs[i];

 if (ws->m_fd_ws >= 0) { // Active connection
 NB::WebSocket::WS_State state = ws->GetState();

 if (state == NB::WebSocket::WS_CLOSE_PENDING ||
 state == NB::WebSocket::WS_CLOSE_SENT) {
 printf(" Slot %d: Stuck in closing state\r\n", i);
 } else if (state == NB::WebSocket::WS_CONNECTING) {
 printf(" Slot %d: Stuck in connecting state\r\n", i);
 }
 }
}
}
}

```

#### Example 5: Application Startup Verification

```

void VerifyWebSocketSubsystem() {
 printf("Verifying WebSocket subsystem initialization...\r\n");

 // Check initial state
 int freeCount = NB::WebSocket::GetFreeWebSocketCount();

 if (freeCount != WS_MAX_SOCKS) {
 printf("WARNING: Expected %d free slots, found %d\r\n",
 WS_MAX_SOCKS, freeCount);
 printf("Dumping connections:\r\n");
 NB::WebSocket::DumpSockets();
 } else {
 printf("OK: All %d WebSocket slots available\r\n", WS_MAX_SOCKS);
 }
}

```

#### Example 6: Debug Handler for Console Command

```

// Console command handler for runtime diagnostics
void HandleWSDumpCommand(void) {
 printf("\n");
 printf("=====\r\n");
 printf(" WebSocket Diagnostic Dump\r\n");
 printf(" Time: %lu seconds\r\n", Secs);
 printf("=====\r\n");

 // Show system stats
 int freeCount = NB::WebSocket::GetFreeWebSocketCount();
 int activeCount = WS_MAX_SOCKS - freeCount;

 printf("Active connections: %d/%d\r\n", activeCount, WS_MAX_SOCKS);
 printf("Free slots: %d\r\n", freeCount);

 // Dump all connections
 NB::WebSocket::DumpSockets();

 printf("\n=====\r\n");
}

// Register in UserMain
void UserMain(void *pd) {
 init();
 // ... other initialization ...

 // Register debug command
 RegisterConsoleCommand("wsdump", HandleWSDumpCommand);

 while (1) {
 OSTimeDly(TICKS_PER_SECOND);
 }
}

```

#### See also

- [DumpSock\(\)](#) - Dump state of single [WebSocket](#) connection
- [GetFreeWebSocketCount\(\)](#) - Get number of available connection slots
- [WSSockStructs](#) - Static array of all [WebSocket](#) structures
- [WS\\_MAX\\_SOCKS](#) - Maximum number of [WebSocket](#) connections

**Note**

Output is sent to debug console (printf/serial)  
Can generate significant output (all WS\_MAX\_SOCKS slots)  
Useful during development, testing, and troubleshooting  
Consider removing or conditionally compiling for production  
Not thread-safe; use appropriate synchronization if needed  
Inactive slots are also displayed for complete system view

**16.190.19.1 Flush()**

```
void NB::WebSocket::Flush ()
```

Flush buffered transmit data to the network.

This member function forces any pending data in the [WebSocket](#)'s transmit buffer to be sent immediately to the peer over the network. Normally, the [WebSocket](#) implementation may buffer small amounts of data to optimize network usage through packet coalescing, but this function overrides that behavior to ensure immediate transmission.

The flush operation is essential for:

- Ensuring critical messages are sent without delay
- Completing data transmission before closing a connection
- Real-time applications requiring immediate delivery
- Interactive protocols with strict timing requirements
- Preventing buffer buildup in high-throughput scenarios
- Coordinating with time-sensitive operations

The function blocks until the buffer contents have been passed to the underlying TCP/IP stack, though actual network transmission depends on network conditions and the peer's receive capability.

**Note**

This function blocks until the flush operation completes or a timeout occurs. The duration depends on buffer size and network speed.

Flushing does not guarantee the peer has received the data, only that it has been transmitted from the local buffer.

**Warning**

Excessive flushing can reduce network efficiency by preventing optimal packet coalescing. Use only when immediate transmission is required.

**See also**

[GetWriteSpace\(\)](#) to check buffer space before writing  
[write\(\)](#) for sending data to the [WebSocket](#)

**Example Usage**

---

**Expand for Example Usage Examples**

### Basic Flush After Write

```
WebSocket *ws = GetActiveWebSocket();

// Send data
const char *message = "Critical alert message";
ws->write(message, strlen(message));

// Ensure it's sent immediately
ws->Flush();
printf("Message flushed to network\n");
```

### Flush Before Closing Connection

```
void CloseWebSocketGracefully(WebSocket *ws) {
 // Send final goodbye message
 ws->write("Connection closing", 18);

 // Ensure message is sent before closing
 ws->Flush();

 // Brief delay to allow network transmission
 OSTimeDly(TICKS_PER_SECOND / 10);

 // Now safe to close
 ws->close();

 printf("Connection closed gracefully\n");
}
```

### Real-Time Data Streaming

```
void StreamSensorData(WebSocket *ws) {
 while (IsStreamingActive()) {
 // Read sensor values
 float temperature = ReadTemperatureSensor();
 float pressure = ReadPressureSensor();

 // Format data
 char buffer[128];
 snprintf(buffer, sizeof(buffer),
 "{\temp\":%.2f,\tpressure\":%.2f}",
 temperature, pressure);

 // Send and flush for real-time delivery
 ws->write(buffer, strlen(buffer));
 ws->Flush();

 // Wait for next sample
 OSTimeDly(TICKS_PER_SECOND);
 }
}
```

### Command-Response Pattern

```
void HandleCommand(WebSocket *ws, const char *command) {
 char response[256];

 // Process command
 ProcessCommand(command, response, sizeof(response));

 // Send response
 ws->write(response, strlen(response));

 // Flush to ensure prompt delivery to client
 ws->Flush();

 printf("Command response sent and flushed\n");
}
```

### Batch Transmission with Final Flush

```
void SendBatchMessages(WebSocket *ws, const char **messages, int count) {
 printf("Sending batch of %d messages...\n", count);

 // Send all messages
 for (int i = 0; i < count; i++) {
 ws->write(messages[i], strlen(messages[i]));
 }

 // Single flush at the end for efficiency
 ws->Flush();

 printf("Batch sent and flushed\n");
}
```

### Interactive Chat Application

```
void SendChatMessage(WebSocket *ws, const char *user, const char *message) {
 char formatted[512];

 // Format chat message with timestamp
 snprintf(formatted, sizeof(formatted),
 "{\"time\":%lu,\"user\":\"%s\",\"msg\":\"%s\"}",
 Secs, user, message);

 // Send immediately for interactive feel
 ws->write(formatted, strlen(formatted));
 ws->Flush();
}
```

### Emergency Alert System

```
void SendEmergencyAlert(WebSocket *ws, const char *alert_type,
 const char *details) {
 char alert[256];

 snprintf(alert, sizeof(alert),
 "{\"type\":\"EMERGENCY\",\"alert\":\"%s\",\"details\":\"%s\"}",
 alert_type, details);

 // Critical message - send immediately
 ws->write(alert, strlen(alert));
 ws->Flush();

 printf("EMERGENCY alert sent: %s\n", alert_type);
}
```

### Periodic Status Updates

```
void StatusUpdateTask(void *pd) {
 WebSocket *ws = (WebSocket *)pd;

 while (1) {
 if (ws->GetState() == WS_STATE_OPEN) {
 // Gather system status
 char status[256];
 GetSystemStatus(status, sizeof(status));

 // Send status update
 ws->write(status, strlen(status));
 ws->Flush();

 printf("Status update sent\n");
 }

 // Update every 10 seconds
 OSTimeDly(TICKS_PER_SECOND * 10);
 }
}
```

### Flow Control with Flush

```
bool SendLargeDataWithFlush(WebSocket *ws, const char *data, int length) {
 const int CHUNK_SIZE = 1024;
 int bytes_sent = 0;

 while (bytes_sent < length) {
 // Check available buffer space
 int space = ws->GetWriteSpace();

 if (space < 256) {
 // Low on space, flush to make room
 printf("Flushing buffer (space: %d bytes)\n", space);
 ws->Flush();
 OSTimeDly(TICKS_PER_SECOND / 10);
 }

 // Send next chunk
 int to_send = (length - bytes_sent);
 if (to_send > CHUNK_SIZE) {
 to_send = CHUNK_SIZE;
 }

 ws->write(data + bytes_sent, to_send);
 bytes_sent += to_send;
 }

 // Final flush
 ws->Flush();
 printf("Large data transfer complete: %d bytes\n", bytes_sent);
}
```

```

 return true;
 }

```

### Conditional Flush Based on Priority

```

void SendMessage(WebSocket *ws, const char *msg, bool high_priority) {
 ws->write(msg, strlen(msg));

 if (high_priority) {
 // Flush immediately for high priority messages
 ws->Flush();
 printf("High priority message flushed\n");
 } else {
 // Let normal priority messages be buffered
 printf("Normal priority message buffered\n");
 }
}

```

#### 16.190.19.2 GetFlushTime()

static void NB::WebSocket::GetFlushTime () [static]  
 Updates the global flush timing for [WebSocket](#) message delivery.

#### 16.190.20 Description

This static function updates timing-related variables that control when buffered [WebSocket](#) messages should be flushed to the network. It helps optimize message batching and delivery by determining appropriate flush intervals based on system conditions, message priorities, or application requirements.

#### 16.190.21 Usage Notes

- Typically called periodically or when flush timing needs recalculation
- May read system time (Secs, TimeTick) to update timing thresholds
- Used to implement Nagle-like algorithms for message coalescing
- Should be lightweight as it may be called frequently
- May update static/global timing variables used by flush logic

#### 16.190.22 Flush Timing Strategies

Common timing strategies include:

- Fixed interval flushing (e.g., every 100ms)
- Adaptive timing based on message rate
- Priority-based immediate vs. delayed flush
- Buffer-size threshold triggers

#### 16.190.23 Thread Safety

If timing variables are accessed from multiple contexts, appropriate synchronization must be used.

#### 16.190.24 See Also

- [ws\\_write\\_notify\(\)](#) - Uses flush timing to control message delivery
- Message buffering and coalescing functions
- Secs, TimeTick - NetBurner timing variables

#### Expand for Example Usage Examples



**Example 1: Fixed Interval Flush Timing**

```

class WebSocketManager {
private:
 static uint32_t nextFlushTime;
 static const uint32_t FLUSH_INTERVAL_MS = 100;

 static void GetFlushTime() {
 // Update next flush time to current time + interval
 nextFlushTime = TimeTick + (FLUSH_INTERVAL_MS * TICKS_PER_SECOND / 1000);
 }

public:
 static bool ShouldFlush() {
 return (TimeTick >= nextFlushTime);
 }

 static void Initialize() {
 GetFlushTime(); // Set initial flush time
 }
};

uint32_t WebSocketManager::nextFlushTime = 0;

```

**Example 2: Adaptive Flush Timing Based on Load**

```

static uint32_t flushIntervalTicks = TICKS_PER_SECOND / 10; // Default 100ms
static uint32_t lastFlushTime = 0;
static uint32_t messagesSinceFlush = 0;

static void GetFlushTime() {
 uint32_t currentTime = TimeTick;

 // Calculate time since last flush
 uint32_t elapsedTicks = currentTime - lastFlushTime;

 // Adapt interval based on message rate
 if (messagesSinceFlush > 100 && elapsedTicks < flushIntervalTicks) {
 // High message rate - flush more frequently
 flushIntervalTicks = flushIntervalTicks / 2;
 if (flushIntervalTicks < TICKS_PER_SECOND / 100) {
 flushIntervalTicks = TICKS_PER_SECOND / 100; // Min 10ms
 }
 }
 else if (messagesSinceFlush < 10 && elapsedTicks > flushIntervalTicks) {
 // Low message rate - flush less frequently
 flushIntervalTicks = flushIntervalTicks * 2;
 if (flushIntervalTicks > TICKS_PER_SECOND) {
 flushIntervalTicks = TICKS_PER_SECOND; // Max 1 second
 }
 }

 lastFlushTime = currentTime;
 messagesSinceFlush = 0;
}

```

**Example 3: Priority-Based Flush Timing**

```

struct FlushTiming {
 uint32_t normalFlushTime;
 uint32_t priorityFlushTime;
 uint32_t idleFlushTime;
 bool hasPriorityMessages;
};

static FlushTiming globalFlushTiming;

static void GetFlushTime() {
 uint32_t currentTime = TimeTick;

 // Update timing based on message priorities in queue
 if (globalFlushTiming.hasPriorityMessages) {
 // Priority messages - flush immediately
 globalFlushTiming.priorityFlushTime = currentTime;
 printf("Priority flush scheduled immediately\r\n");
 }
 else if (GetQueuedMessageCount() > 50) {
 // Many messages - flush soon (50ms)
 globalFlushTiming.normalFlushTime = currentTime + (TICKS_PER_SECOND / 20);
 }
 else if (GetQueuedMessageCount() > 0) {
 // Few messages - batch for efficiency (200ms)
 globalFlushTiming.normalFlushTime = currentTime + (TICKS_PER_SECOND / 5);
 }
 else {
 // Idle - flush any stale data (1 second)
 globalFlushTiming.idleFlushTime = currentTime + TICKS_PER_SECOND;
 }
}

```

```
 }
}
```

#### Example 4: Integration with Periodic Update Task

```
void WebSocketUpdateTask(void* pd) {
 while (1) {
 // Update flush timing every 50ms
 OSTimeDly(TICKS_PER_SECOND / 20);

 GetFlushTime();

 // Check if flush is needed
 if (ShouldFlushNow()) {
 FlushAllConnections();
 }
 }
}

void StartWebSocketManager() {
 // Initialize timing
 GetFlushTime();

 // Start update task
 OSTaskCreate(WebSocketUpdateTask,
 nullptr,
 &TaskStack[USER_TASK_STK_SIZE],
 TaskStack,
 WS_UPDATE_TASK_PRIO);
}
```

#### 16.190.24.1 GetFreeWebSocketCount()

```
static int NB::WebSocket::GetFreeWebSocketCount () [static]
```

Get the number of available [WebSocket](#) connection slots.

Returns the count of free (unused) [WebSocket](#) connection slots available for new connections. This is useful for checking resource availability before attempting to create new connections, preventing "no slots available" errors.

#### 16.190.25 Return Value

Returns the number of free connection slots (0 to WS\_MAX\_SOCKS):

- WS\_MAX\_SOCKS: All slots free, no active connections
- 1 to WS\_MAX\_SOCKS-1: Some slots free, some in use
- 0: All slots in use, cannot accept new connections

#### 16.190.26 Usage Guidelines

- Check before calling [Connect\(\)](#) or [WSUpgrade\(\)](#) to avoid failures
- Monitor resource usage in multi-connection applications
- Implement connection limits and queuing strategies
- Use for load shedding and resource management
- Display in status interfaces and diagnostics

#### Expand for Example Usage

#### Examples

**Example 1: Check Before Connecting**

```

void SafeConnect(const char *host, const char *resource) {
 // Check if slots available
 int freeSlots = NB::WebSocket::GetFreeWebSocketCount();

 if (freeSlots <= 0) {
 printf("Cannot connect: No WebSocket slots available\r\n");
 printf("Active connections: %d/%d\r\n",
 WS_MAX_SOCKETS, WS_MAX_SOCKETS);
 return;
 }

 printf("Connecting (%d slots available)...\r\n", freeSlots);
 int wsFd = NB::WebSocket::Connect(host, resource, NULL, 80, false);

 if (wsFd >= 0) {
 printf("Connected successfully (fd %d)\r\n", wsFd);
 printf("Remaining slots: %d\r\n",
 NB::WebSocket::GetFreeWebSocketCount());
 } else {
 printf("Connection failed\r\n");
 }
}

```

**Example 2: Server Connection Limit**

```

int HandleWebSocketUpgrade(HTTP_Request *req, int sock) {
 // Reserve one slot for admin/priority connections
 const int RESERVED_SLOTS = 1;
 int freeSlots = NB::WebSocket::GetFreeWebSocketCount();

 if (freeSlots <= RESERVED_SLOTS) {
 printf("Rejecting connection: Only %d slots free (need %d reserved)\r\n",
 freeSlots, RESERVED_SLOTS);

 writestring(sock, "HTTP/1.1 503 Service Unavailable\r\n");
 writestring(sock, "Content-Type: text/plain\r\n\r\n");
 writestring(sock, "Server at capacity");
 return -1;
 }

 printf("Accepting WebSocket (%d slots remaining)\r\n",
 freeSlots - 1);

 if (WSUpgrade(req, sock) >= 0) {
 OSSimpleTaskCreateName(WebSocketTask, MAIN_PRIO - 1,
 (void *) (intptr_t) sock, "WSTask");
 return 0;
 }

 return -1;
}

```

**Example 3: Connection Queue Management**

```

#define MAX_QUEUED_CONNECTIONS 10

struct QueuedConnection {
 char host[64];
 char resource[64];
 bool pending;
};

QueuedConnection connQueue[MAX_QUEUED_CONNECTIONS];
int queueCount = 0;
OS_CRIT queueCrit;

void QueueConnection(const char *host, const char *resource) {
 OSCritEnter(&queueCrit, TICKS_PER_SECOND);

 if (queueCount >= MAX_QUEUED_CONNECTIONS) {
 printf("Connection queue full\r\n");
 OSCritLeave(&queueCrit);
 return;
 }

 strncpy(connQueue[queueCount].host, host, 63);
 strncpy(connQueue[queueCount].resource, resource, 63);
 connQueue[queueCount].pending = true;
 queueCount++;
}

```

```

 OSCritLeave(&queueCrit);
 printf("Connection queued (%d in queue)\r\n", queueCount);
}

void ProcessConnectionQueue(void *pd) {
 while (1) {
 OSTimeDly(TICKS_PER_SECOND);

 // Check if we can process queued connections
 int freeSlots = NB::WebSocket::GetFreeWebSocketCount();

 if (freeSlots > 0 && queueCount > 0) {
 OSCritEnter(&queueCrit, TICKS_PER_SECOND);

 printf("Processing queue (%d pending, %d slots)\r\n",
 queueCount, freeSlots);

 for (int i = 0; i < queueCount && freeSlots > 0; i++) {
 if (connQueue[i].pending) {
 int wsFd = NB::WebSocket::Connect(connQueue[i].host,
 connQueue[i].resource,
 NULL, 80, false);

 if (wsFd >= 0) {
 printf("Queued connection established\r\n");
 connQueue[i].pending = false;
 freeSlots--;
 }
 }
 }

 // Compact queue
 int newCount = 0;
 for (int i = 0; i < queueCount; i++) {
 if (connQueue[i].pending) {
 if (i != newCount) {
 connQueue[newCount] = connQueue[i];
 }
 newCount++;
 }
 }
 queueCount = newCount;

 OSCritLeave(&queueCrit);
 }
 }
}

```

#### Example 4: Resource Usage Display

```

void DisplayWebSocketStatus() {
 int freeSlots = NB::WebSocket::GetFreeWebSocketCount();
 int usedSlots = WS_MAX_SOCKS - freeSlots;
 int percentUsed = (usedSlots * 100) / WS_MAX_SOCKS;

 printf("WebSocket Resource Usage:\r\n");
 printf(" Active: %d/%d (%d%%)\r\n",
 usedSlots, WS_MAX_SOCKS, percentUsed);
 printf(" Free: %d\r\n", freeSlots);

 // Visual bar
 printf(" [");
 for (int i = 0; i < WS_MAX_SOCKS; i++) {
 printf(i < usedSlots ? "#" : "-");
 }
 printf("]\r\n");

 // Warning for high usage
 if (percentUsed >= 75) {
 printf(" WARNING: High resource usage!\r\n");
 }
}

```

#### Example 5: Adaptive Connection Strategy

```

typedef enum {
 STRATEGY_AGGRESSIVE, // Use all available slots
 STRATEGY_CONSERVATIVE, // Keep some slots reserved
 STRATEGY_MINIMAL // Use minimal connections
} ConnectionStrategy;

int GetMaxAllowedConnections(ConnectionStrategy strategy) {
 int freeSlots = NB::WebSocket::GetFreeWebSocketCount();
}

```

```

switch (strategy) {
 case STRATEGY_AGGRESSIVE:
 // Use all but one slot
 return (freeSlots > 1) ? (WS_MAX_SOCKS - 1) : 0;

 case STRATEGY_CONSERVATIVE:
 // Use up to 75% of total
 return (WS_MAX_SOCKS * 3) / 4;

 case STRATEGY_MINIMAL:
 // Use up to 50% of total
 return WS_MAX_SOCKS / 2;

 default:
 return 0;
}

bool CanCreateConnection(ConnectionStrategy strategy) {
 int maxAllowed = GetMaxAllowedConnections(strategy);
 int currentUsed = WS_MAX_SOCKS - NB::WebSocket::GetFreeWebSocketCount();

 return currentUsed < maxAllowed;
}

```

#### Example 6: Periodic Resource Monitoring

```

void MonitorWebSocketResources(void *pd) {
 int lastFree = -1;

 while (1) {
 OSTimeDly(TICKS_PER_SECOND * 5);

 int freeNow = NB::WebSocket::GetFreeWebSocketCount();

 // Report changes
 if (freeNow != lastFree) {
 printf("WebSocket slots changed: %d -> %d free\r\n",
 lastFree, freeNow);

 if (freeNow == 0) {
 printf("ALERT: All WebSocket slots in use!\r\n");
 NB::WebSocket::DumpSockets();
 } else if (freeNow == WS_MAX_SOCKS) {
 printf("INFO: All WebSocket slots now free\r\n");
 }

 lastFree = freeNow;
 }
 }
}

```

#### See also

[WS\\_MAX\\_SOCKS](#) - Maximum number of [WebSocket](#) connections  
[Connect\(\)](#) - Create new [WebSocket](#) client connection  
[WSUpgrade\(\)](#) - Upgrade HTTP connection to [WebSocket](#) (server-side)  
[DumpSockets\(\)](#) - Display state of all connections  
[s\\_openSocketCount](#) - Static member tracking open connection count

#### Note

Return value indicates available slots, not necessarily successful connection  
 Value can change between check and connection attempt (race condition in multi-threaded)  
 Always handle connection failure even if slots appear available  
 Thread-safe; uses internal synchronization

#### 16.190.26.1 getopt()

```
int NB::WebSocket::getoption ()
```

Get the current socket options bitmap.

### Returns

int A bitmask of currently enabled socket options, or negative value on failure

This member function retrieves the current state of all socket options for this [WebSocket](#) connection. It returns a bitmask where each bit represents a specific socket option's state (enabled/disabled).

The returned value can be tested against specific option flags using bitwise AND operations to determine if particular options are currently enabled.

Common uses include:

- Debugging socket configuration
- Verifying option settings after configuration
- Conditionally changing options based on current state
- Logging and diagnostics
- Saving/restoring socket configuration

Example option flags that may be present in the bitmask:

- WS\_SO\_TEXT: Text frame mode enabled
- WS\_SO\_BINARY: Binary frame mode enabled
- TCP\_NODELAY: Nagle's algorithm disabled
- SO\_KEEPALIVE: TCP keepalive enabled
- WS\_SO\_COMPRESS: Compression enabled

### Note

The exact interpretation of the bitmask depends on the [WebSocket](#) implementation and which options are supported.

The returned bitmask is a snapshot of the current state. It may change as options are set or cleared.

### See also

[setoption\(\)](#) for setting socket options

[clroption\(\)](#) for clearing socket options

### Example Usage

---

#### Expand for Example Usage

##### Examples

###### Query Current Options

```
WebSocket *ws = GetActiveWebSocket();

int options = ws->getoption();
if (options >= 0) {
 printf("Current socket options: 0x%08X\n", options);
} else {
 printf("Failed to get socket options\n");
}
```

### Check Specific Option

```

WebSocket *ws = GetActiveWebSocket();
int options = ws->getoption();

if (options >= 0) {
 if (options & WS_SO_BINARY) {
 printf("WebSocket is in BINARY mode\n");
 } else if (options & WS_SO_TEXT) {
 printf("WebSocket is in TEXT mode\n");
 } else {
 printf("WebSocket mode unknown\n");
 }

 if (options & TCP_NODELAY) {
 printf("Low latency mode: Nagle's algorithm DISABLED\n");
 } else {
 printf("Buffering mode: Nagle's algorithm ENABLED\n");
 }

 if (options & SO_KEEPALIVE) {
 printf("TCP keepalive: ENABLED\n");
 } else {
 printf("TCP keepalive: DISABLED\n");
 }
}

```

### Conditional Option Setting

```

void EnsureBinaryModeEnabled(WebSocket *ws) {
 int options = ws->getoption();

 if (options < 0) {
 printf("ERROR: Cannot read socket options\n");
 return;
 }

 // Only set binary mode if not already enabled
 if (!(options & WS_SO_BINARY)) {
 ws->setoption(WS_SO_BINARY);
 printf("Binary mode was disabled, now enabled\n");
 } else {
 printf("Binary mode already enabled\n");
 }
}

```

### Diagnostic Logging

```

void LogWebSocketConfiguration(WebSocket *ws) {
 int options = ws->getoption();

 if (options < 0) {
 printf("ERROR: Could not retrieve socket options\n");
 return;
 }

 printf("=== WebSocket Configuration ===\n");
 printf("Frame Mode:\n");
 printf(" WS_SO_TEXT: %s\n", (options & WS_SO_TEXT) ? "ON" : "OFF");
 printf(" WS_SO_BINARY: %s\n", (options & WS_SO_BINARY) ? "ON" : "OFF");
 printf("TCP Options:\n");
 printf(" SO_KEEPALIVE: %s\n", (options & SO_KEEPALIVE) ? "ON" : "OFF");
 printf(" TCP_NODELAY: %s\n", (options & TCP_NODELAY) ? "ON" : "OFF");
 printf("Features:\n");
 printf(" WS_SO_COMPRESS:%s\n", (options & WS_SO_COMPRESS) ? "ON" : "OFF");
 printf("Raw bitmask: 0x%08X\n", options);
 printf("=====\n");
}

```

### Compare Configurations

```

void CompareWebSocketOptions(WebSocket *ws1, WebSocket *ws2) {
 int options1 = ws1->getoption();
 int options2 = ws2->getoption();

 if (options1 < 0 || options2 < 0) {
 printf("ERROR: Could not retrieve options for comparison\n");
 return;
 }

 if (options1 == options2) {
 printf("Both WebSockets have identical options (0x%08X)\n",
 options1);
 } else {
 printf("WebSocket options differ:\n");
 }
}

```

```

 printf(" WS1: 0x%08X\n", options1);
 printf(" WS2: 0x%08X\n", options2);

 int diff = options1 ^ options2;
 printf(" Differences: 0x%08X\n", diff);

 // Show which specific options differ
 if (diff & WS_SO_BINARY) {
 printf(" - Binary mode differs\n");
 }
 if (diff & TCP_NODELAY) {
 printf(" - TCP_NODELAY differs\n");
 }
 if (diff & SO_KEEPALIVE) {
 printf(" - Keepalive differs\n");
 }
}
}

```

### Save and Restore Configuration

```

int SaveWebSocketConfig(WebSocket *ws) {
 int saved_options = ws->getoption();

 if (saved_options >= 0) {
 printf("Configuration saved: 0x%08X\n", saved_options);
 }

 return saved_options;
}

void RestoreWebSocketConfig(WebSocket *ws, int saved_options) {
 if (saved_options < 0) {
 printf("Invalid saved configuration\n");
 return;
 }

 // Clear all options first
 ws->clroption(WS_SO_BINARY);
 ws->clroption(WS_SO_TEXT);
 ws->clroption(TCP_NODELAY);
 ws->clroption(SO_KEEPALIVE);
 ws->clroption(WS_SO_COMPRESS);

 // Restore saved options
 if (saved_options & WS_SO_BINARY) ws->setoption(WS_SO_BINARY);
 if (saved_options & WS_SO_TEXT) ws->setoption(WS_SO_TEXT);
 if (saved_options & TCP_NODELAY) ws->setoption(TCP_NODELAY);
 if (saved_options & SO_KEEPALIVE) ws->setoption(SO_KEEPALIVE);
 if (saved_options & WS_SO_COMPRESS) ws->setoption(WS_SO_COMPRESS);

 printf("Configuration restored: 0x%08X\n", saved_options);
}

// Usage:
int saved = SaveWebSocketConfig(ws);
// ... modify options ...
RestoreWebSocketConfig(ws, saved);

```

### Verify Configuration After Setup

```

bool VerifyWebSocketSetup(WebSocket *ws, int expected_options) {
 int actual_options = ws->getoption();

 if (actual_options < 0) {
 printf("ERROR: Cannot verify configuration\n");
 return false;
 }

 if ((actual_options & expected_options) == expected_options) {
 printf("Configuration verified successfully\n");
 return true;
 } else {
 printf("Configuration mismatch!\n");
 printf(" Expected: 0x%08X\n", expected_options);
 printf(" Actual: 0x%08X\n", actual_options);
 printf(" Missing: 0x%08X\n",
 expected_options & ~actual_options);
 return false;
 }
}

// Usage:
ws->setoption(WS_SO_BINARY | TCP_NODELAY);
VerifyWebSocketSetup(ws, WS_SO_BINARY | TCP_NODELAY);

```



### Runtime Option Monitoring

```
void MonitorWebSocketOptions(WebSocket *ws) {
 static int last_options = -1;

 int current_options = ws->getoption();

 if (current_options >= 0) {
 if (current_options != last_options) {
 printf("Options changed: 0x%08X -> 0x%08X\n",
 last_options, current_options);
 LogWebSocketConfiguration(ws);
 last_options = current_options;
 }
 }
}
```

---

#### 16.190.26.2 GetPingReplyTick()

```
int NB::WebSocket::GetPingReplyTick (
 uint32_t * replyTick)
```

Get the tick count when the last pong was received.

##### Parameters

|                  |                                                                      |
|------------------|----------------------------------------------------------------------|
| <i>replyTick</i> | Pointer to variable to receive the tick count of last pong reception |
|------------------|----------------------------------------------------------------------|

##### Returns

int Returns 0 on success with valid tick count, negative value on error

This function retrieves the system tick count (from TimeTick) at the moment the most recent pong frame was received. This timestamp can be used in conjunction with the ping send timestamp from [Ping\(\)](#) to calculate precise round-trip time.

The function is useful for:

- Calculating round-trip time (RTT) measurements
- Verifying pong reception without blocking
- Implementing non-blocking ping-pong monitoring
- Logging network performance metrics
- Detecting when the last successful communication occurred

The returned tick count represents the absolute system time when the pong was received. Subtract the ping send tick (from [Ping\(\)](#)) to get elapsed ticks, then convert to milliseconds using TICKS\_PER\_SECOND:

```
RTT_ms = ((replyTick - sentTick) * 1000) / TICKS_PER_SECOND
```

##### Note

This function returns the timestamp of the LAST received pong, which may correspond to an earlier ping if multiple pings have been sent.

If no pong has been received since connection establishment, the function may return an error or an initial/invalid timestamp.

This is a non-blocking call that returns immediately with the stored timestamp value.

##### See also

[Ping\(\)](#) to send ping frames and get send timestamps

[WaitForPingReply\(\)](#) for blocking wait on pong reception

## Example Usage

### Expand for Example Usage

#### Examples

##### Calculate Round-Trip Time

```
WebSocket *ws = GetActiveWebSocket();
uint32_t sent_tick;

// Send ping
if (ws->Ping(0, &sent_tick) == 0) {
 // Wait for pong
 if (ws->WaitForPingReply(TICKS_PER_SECOND * 3) == 0) {
 uint32_t reply_tick;

 // Get pong timestamp
 if (ws->GetPingReplyTick(&reply_tick) == 0) {
 // Calculate RTT
 uint32_t rtt_ticks = reply_tick - sent_tick;
 float rtt_ms = (rtt_ticks * 1000.0f) / TICKS_PER_SECOND;

 printf("Round-trip time: %.3f ms\n", rtt_ms);
 }
 }
}
```

##### Non-Blocking Pong Check

```
bool CheckForPongReceived(WebSocket *ws, uint32_t ping_sent_tick) {
 uint32_t reply_tick;

 if (ws->GetPingReplyTick(&reply_tick) == 0) {
 // Check if this pong is newer than our ping
 if (reply_tick > ping_sent_tick) {
 uint32_t rtt_ticks = reply_tick - ping_sent_tick;
 float rtt_ms = (rtt_ticks * 1000.0f) / TICKS_PER_SECOND;

 printf("Pong received: RTT = %.2f ms\n", rtt_ms);
 return true;
 }
 }

 return false;
}
```

##### Track Last Communication Time

```
uint32_t GetTimeSinceLastPong(WebSocket *ws) {
 uint32_t reply_tick;

 if (ws->GetPingReplyTick(&reply_tick) != 0) {
 printf("No pong received yet\n");
 return 0xFFFFFFFF; // Max value indicates no pong
 }

 uint32_t current_tick = TimeTick;
 uint32_t elapsed_ticks = current_tick - reply_tick;

 // Convert to seconds
 uint32_t elapsed_seconds = elapsed_ticks / TICKS_PER_SECOND;

 return elapsed_seconds;
}

// Usage:
uint32_t idle_time = GetTimeSinceLastPong(ws);
if (idle_time > 60) {
 printf("WARNING: No pong received for %lu seconds\n", idle_time);
}
```

##### Periodic RTT Monitoring

```
void MonitorRTT(WebSocket *ws) {
 static uint32_t last_ping_tick = 0;
 static float rtt_history[10];
 static int rtt_index = 0;

 uint32_t current_tick = TimeTick;

 // Send ping every 5 seconds
 if ((current_tick - last_ping_tick) > (TICKS_PER_SECOND * 5)) {
 uint32_t sent_tick;
```

```

 if (ws->Ping(0, &sent_tick) == 0) {
 last_ping_tick = sent_tick;
 }
 }

 // Check for pong
 uint32_t reply_tick;
 if (ws->GetPingReplyTick(&reply_tick) == 0) {
 if (reply_tick > last_ping_tick) {
 // Calculate and store RTT
 uint32_t rtt_ticks = reply_tick - last_ping_tick;
 float rtt_ms = (rtt_ticks * 1000.0f) / TICKS_PER_SECOND;

 rtt_history[rtt_index] = rtt_ms;
 rtt_index = (rtt_index + 1) % 10;

 // Calculate average
 float avg_rtt = 0;
 for (int i = 0; i < 10; i++) {
 avg_rtt += rtt_history[i];
 }
 avg_rtt /= 10.0f;

 printf("RTT: %.2f ms (avg: %.2f ms)\n", rtt_ms, avg_rtt);
 }
 }
}

```

### Detect Slow Responses

```

void DetectSlowConnection(WebSocket *ws, uint32_t ping_sent_tick,
 float threshold_ms) {
 uint32_t reply_tick;

 if (ws->GetPingReplyTick(&reply_tick) == 0) {
 if (reply_tick > ping_sent_tick) {
 uint32_t rtt_ticks = reply_tick - ping_sent_tick;
 float rtt_ms = (rtt_ticks * 1000.0f) / TICKS_PER_SECOND;

 if (rtt_ms > threshold_ms) {
 printf("WARNING: Slow connection detected: %.2f ms "
 "(threshold: %.2f ms)\n", rtt_ms, threshold_ms);
 } else {
 printf("Connection speed acceptable: %.2f ms\n", rtt_ms);
 }
 }
 }
}

```

### Log Pong Reception

```

void LogPongReception(WebSocket *ws, const char *label) {
 uint32_t reply_tick;

 if (ws->GetPingReplyTick(&reply_tick) == 0) {
 uint32_t current_time = Secs;

 printf("[%lu] %s: Pong received at tick %lu\n",
 current_time, label, reply_tick);

 // Log to persistent storage
 LogToFile(current_time, label, reply_tick);
 } else {
 printf("%s: No pong data available\n", label);
 }
}

```

### Compare Multiple Ping Attempts

```

void ComparePingAttempts(WebSocket *ws) {
 const int ATTEMPTS = 5;
 uint32_t ping_ticks[ATTEMPTS];
 float rtt[ATTEMPTS];

 for (int i = 0; i < ATTEMPTS; i++) {
 // Send ping
 if (ws->Ping(0, &ping_ticks[i]) == 0) {
 // Wait for pong
 if (ws->WaitForPingReply(TICKS_PER_SECOND * 3) == 0) {
 uint32_t reply_tick;
 ws->GetPingReplyTick(&reply_tick);

 uint32_t rtt = reply_tick - ping_ticks[i];
 rtt[i] = (rtt * 1000.0f) / TICKS_PER_SECOND;
 } else {

```

```

 rtts[i] = -1.0f; // Timeout
 }

 OSTimeDly(TICKS_PER_SECOND);
}

// Analyze results
printf("Ping Attempt Analysis:\n");
for (int i = 0; i < ATTEMPTS; i++) {
 if (rtts[i] >= 0) {
 printf(" Attempt %d: %.2f ms\n", i + 1, rtts[i]);
 } else {
 printf(" Attempt %d: TIMEOUT\n", i + 1);
 }
}
}
}

```

### Validate Pong Sequence

```

bool ValidatePongSequence(WebSocket *ws, uint32_t expected_min_tick) {
 uint32_t reply_tick;

 if (ws->GetPingReplyTick(&reply_tick) != 0) {
 printf("ERROR: No pong data available\n");
 return false;
 }

 if (reply_tick < expected_min_tick) {
 printf("WARNING: Received stale pong (tick %lu < %lu)\n",
 reply_tick, expected_min_tick);
 return false;
 }

 printf("Pong sequence valid (tick %lu)\n", reply_tick);
 return true;
}

```

### 16.190.26.3 GetState()

WS\_State NB::WebSocket::GetState () [inline]

Get the current connection state of the [WebSocket](#).

#### Returns

WS\_State The current state of the [WebSocket](#) connection

This inline function returns the current state of the [WebSocket](#) connection. The state indicates where the connection is in its lifecycle, from initial handshake through active communication to closure.

Common [WebSocket](#) states include:

- WS\_STATE\_CONNECTING: Connection being established, handshake in progress
- WS\_STATE\_OPEN: Connection established and ready for communication
- WS\_STATE\_CLOSING: Close handshake initiated, waiting for closure
- WS\_STATE\_CLOSED: Connection fully closed

Additional states may include:

- WS\_STATE\_INIT: Initial state before connection attempt
- WS\_STATE\_ERROR: [Error](#) state indicating connection failure

Monitoring the state is essential for:

- Determining if the connection is ready for data transmission
- Handling connection lifecycle events appropriately
- Implementing retry logic for failed connections
- Preventing operations on closed connections
- State machine implementations

**Note**

This function is inline for performance efficiency as it's frequently called in connection management code.

The state represents a snapshot at the time of the call and may change immediately afterward due to network events or other operations.

**See also**

[Ping\(\)](#) for checking connection liveness

[WaitForPingReply\(\)](#) for verifying active connection

**Example Usage****Expand for Example Usage****Examples****Check Connection Ready**

```
WebSocket *ws = GetActiveWebSocket();

if (ws->GetState() == WS_STATE_OPEN) {
 // Connection is ready, safe to send data
 ws->write("Hello, WebSocket!", 17);
} else {
 printf("Connection not ready, state: %d\n", ws->GetState());
}
```

**Wait for Connection to Establish**

```
bool WaitForConnection(WebSocket *ws, uint32_t timeout_seconds) {
 uint32_t start_time = Secs;

 while ((Secs - start_time) < timeout_seconds) {
 WS_State state = ws->GetState();

 if (state == WS_STATE_OPEN) {
 printf("Connection established\n");
 return true;
 } else if (state == WS_STATE_CLOSED || state == WS_STATE_ERROR) {
 printf("Connection failed, state: %d\n", state);
 return false;
 }

 // Still connecting, wait a bit
 OSTimeDly(TICKS_PER_SECOND / 10);
 }

 printf("Connection timeout\n");
 return false;
}
```

**State Machine Implementation**

```
void HandleWebSocketState(WebSocket *ws) {
 WS_State state = ws->GetState();

 switch (state) {
 case WS_STATE_INIT:
 printf("Initializing connection...\n");
 break;

 case WS_STATE_CONNECTING:
 printf("Handshake in progress...\n");
 break;

 case WS_STATE_OPEN:
 printf("Connection active\n");
 // Handle data transmission
 HandleActiveConnection(ws);
 break;

 case WS_STATE_CLOSING:
 printf("Connection closing...\n");
 // Cleanup operations
 break;

 case WS_STATE_CLOSED:
 printf("Connection closed\n");
 // Attempt reconnection or cleanup
 break;
 }
}
```

```

 HandleReconnection(ws);
 break;

 case WS_STATE_ERROR:
 printf("Connection error\n");
 HandleConnectionError(ws);
 break;

 default:
 printf("Unknown state: %d\n", state);
 break;
 }
}

```

### Safe Write with State Check

```

int SafeWebSocketWrite(WebSocket *ws, const char *data, int length) {
 WS_State state = ws->GetState();

 if (state != WS_STATE_OPEN) {
 printf("Cannot write: connection not open (state: %d)\n", state);
 return -1;
 }

 return ws->write(data, length);
}

```

### State Transition Monitoring

```

void MonitorWebSocketState(WebSocket *ws) {
 static WS_State last_state = WS_STATE_INIT;

 WS_State current_state = ws->GetState();

 if (current_state != last_state) {
 printf("State transition: %d -> %d\n", last_state, current_state);

 // Log state change
 LogStateChange(Secs, last_state, current_state);

 last_state = current_state;
 }
}

```

### Automatic Reconnection Logic

```

void AutoReconnectTask(void *pd) {
 WebSocket *ws = (WebSocket *)pd;

 while (1) {
 WS_State state = ws->GetState();

 if (state == WS_STATE_CLOSED || state == WS_STATE_ERROR) {
 printf("Connection lost, attempting reconnection...\n");

 // Attempt reconnection
 ws->connect("ws://server.com");

 // Wait for connection result
 OSTimeDly(TICKS_PER_SECOND * 5);
 }

 OSTimeDly(TICKS_PER_SECOND);
 }
}

```

### Connection Health Check

```

bool IsConnectionHealthy(WebSocket *ws) {
 WS_State state = ws->GetState();

 if (state == WS_STATE_OPEN) {
 // Connection appears open, verify with ping
 uint32_t sent_tick;
 if (ws->Ping(0, &sent_tick) == 0) {
 return true;
 }
 }

 printf("Connection unhealthy, state: %d\n", state);
 return false;
}

```

### State-based Resource Management

```
void ManageWebSocketResources(WebSocket *ws) {
 WS_State state = ws->GetState();

 switch (state) {
 case WS_STATE_OPEN:
 // Allocate buffers and resources for active connection
 AllocateBuffers(ws);
 break;

 case WS_STATE_CLOSED:
 // Release resources when connection is closed
 ReleaseBuffers(ws);
 break;

 default:
 // Minimal resources for other states
 break;
 }
}
```

---

#### 16.190.26.4 GetWebSocketRecord()

```
static WebSocket * NB::WebSocket::GetWebSocketRecord (
 int fd) [static]
```

Find a [WebSocket](#) record by its file descriptor.

##### Parameters

|           |                                                             |
|-----------|-------------------------------------------------------------|
| <i>fd</i> | The <a href="#">WebSocket</a> file descriptor to search for |
|-----------|-------------------------------------------------------------|

##### Returns

WebSocket\* Pointer to the [WebSocket](#) record if found, nullptr otherwise

This function searches through the internal [WebSocket](#) registry to locate a [WebSocket](#) instance associated with the given file descriptor. It is typically used internally by the [WebSocket](#) management system to retrieve the [WebSocket](#) object for operations on an active connection.

##### Note

This is a static function intended for internal use within the [WebSocket](#) implementation.

##### Example Usage

---

#### Expand for Example Usage

##### Examples

###### Basic Usage

```
// Assuming you have a file descriptor from a WebSocket event
int ws_fd = 5; // Example file descriptor

WebSocket *ws = GetWebSocketRecord(ws_fd);
if (ws != nullptr) {
 // WebSocket found, perform operations
 printf("Found WebSocket for fd %d\n", ws_fd);
} else {
 // WebSocket not found
 printf("No WebSocket found for fd %d\n", ws_fd);
}
```

##### Error Handling

```
int ws_fd = GetWebSocketFdFromEvent(); // Get fd from some event

WebSocket *ws = GetWebSocketRecord(ws_fd);
if (ws == nullptr) {
 printf("ERROR: Invalid WebSocket file descriptor: %d\n", ws_fd);
 return;
}

// Safe to use ws here
ws->SendText("Connection validated");
```

---

### 16.190.26.5 GetWriteSpace()

```
int NB::WebSocket::GetWriteSpace () [inline]
```

Get available transmit buffer space in bytes.

#### Returns

int Number of bytes available in the transmit buffer for writing

This inline function returns the amount of free space currently available in the [WebSocket](#)'s transmit (TX) buffer. The value indicates how many bytes can be written to the buffer before it becomes full.

This function is useful for:

- Determining if there's sufficient space before attempting a write
- Implementing flow control to prevent buffer overflow
- Monitoring buffer utilization and performance
- Deciding whether to queue or drop data in high-throughput scenarios

The available space decreases as data is written to the buffer and increases as buffered data is transmitted over the network. The total buffer capacity is fixed at initialization.

#### Note

This function is inline for performance efficiency, as it's often called in performance-critical paths.

The returned value represents instantaneous available space. It may change immediately after the call as other operations occur.

#### Warning

In multi-threaded environments, the available space may change between checking this value and attempting to write. Always verify write operation return values.

#### See also

[ws\\_write\(\)](#) for writing data to the [WebSocket](#)

[ws\\_flush\(\)](#) to force transmission of buffered data

#### Example Usage

---

### Expand for Example Usage

#### Examples

##### Check Buffer Space Before Writing

```
WebSocket *ws = GetActiveWebSocket();
const char *message = "Important data message";
int msg_length = strlen(message);

int available = ws->GetWriteSpace();
if (available >= msg_length) {
 // Sufficient space, safe to write
 ws_write(ws->GetFd(), message, msg_length);
 printf("Message sent (%d bytes, %d bytes remaining)\n",
 msg_length, available - msg_length);
} else {
 printf("Insufficient buffer space: need %d, have %d\n",
 msg_length, available);
}
```



### Monitor Buffer Utilization

```
void MonitorWebSocketBuffer(WebSocket *ws) {
 int available = ws->GetWriteSpace();
 int total_capacity = GetTxBufferSize(); // Assuming this exists
 int used = total_capacity - available;
 float usage_percent = (used * 100.0f) / total_capacity;

 printf("TX Buffer Status:\n");
 printf(" Used: %d bytes\n", used);
 printf(" Available: %d bytes\n", available);
 printf(" Capacity: %d bytes\n", total_capacity);
 printf(" Usage: %.1f%%\n", usage_percent);

 if (usage_percent > 90.0f) {
 printf(" WARNING: Buffer nearly full!\n");
 }
}
```

### Flow Control Implementation

```
bool WriteWithFlowControl(WebSocket *ws, const char *data, int length) {
 const int MIN_BUFFER_SPACE = 256; // Reserve some space

 int available = ws->GetWriteSpace();

 if (available < MIN_BUFFER_SPACE) {
 // Buffer too full, flush first
 printf("Buffer low (%d bytes), flushing...\n", available);
 ws_flush(ws->GetFd());

 // Wait briefly for buffer to drain
 OSTimeDly(TICKS_PER_SECOND / 10);

 available = ws->GetWriteSpace();
 }

 if (available >= length) {
 ws_write(ws->GetFd(), data, length);
 return true;
 } else {
 printf("ERROR: Still insufficient space after flush\n");
 return false;
 }
}
```

### Large Data Transfer with Space Checking

```
bool SendLargeData(WebSocket *ws, const char *data, int total_length) {
 int bytes_sent = 0;

 while (bytes_sent < total_length) {
 int available = ws->GetWriteSpace();

 if (available < 100) {
 // Low space, flush and wait
 ws_flush(ws->GetFd());
 OSTimeDly(TICKS_PER_SECOND / 20);
 continue;
 }

 // Send as much as we can
 int to_send = (total_length - bytes_sent);
 if (to_send > available) {
 to_send = available;
 }

 int written = ws_write(ws->GetFd(), data + bytes_sent, to_send);
 if (written > 0) {
 bytes_sent += written;
 printf("Sent %d/%d bytes (%d buffer space)\n",
 bytes_sent, total_length, ws->GetWriteSpace());
 } else {
 printf("Write error\n");
 return false;
 }
 }

 ws_flush(ws->GetFd());
 return true;
}
```

## Adaptive Data Rate Control

```
void StreamDataWithAdaptiveRate(WebSocket *ws) {
 const int HIGH_WATER_MARK = 2048;
 const int LOW_WATER_MARK = 512;
 uint32_t delay = TICKS_PER_SECOND / 10; // Start at 100ms

 while (IsStreaming()) {
 char data[256];
 int data_length = GenerateStreamData(data, sizeof(data));

 int available = ws->GetWriteSpace();

 if (available > HIGH_WATER_MARK) {
 // Plenty of space, send faster
 delay = TICKS_PER_SECOND / 20; // 50ms
 } else if (available < LOW_WATER_MARK) {
 // Running low, slow down
 delay = TICKS_PER_SECOND / 5; // 200ms
 ws_flush(ws->GetFd());
 }

 ws_write(ws->GetFd(), data, data_length);
 OSTimeDly(delay);
 }
}
```

## Buffer Health Check

```
typedef enum {
 BUFFER_HEALTHY,
 BUFFER_WARNING,
 BUFFER_CRITICAL
} BufferHealth;

BufferHealth CheckBufferHealth(WebSocket *ws) {
 int available = ws->GetWriteSpace();

 if (available > 1024) {
 return BUFFER_HEALTHY;
 } else if (available > 256) {
 printf("WARNING: Buffer space low (%d bytes)\n", available);
 return BUFFER_WARNING;
 } else {
 printf("CRITICAL: Buffer nearly full (%d bytes left)\n", available);
 return BUFFER_CRITICAL;
 }
}

void HandleBufferHealth(WebSocket *ws) {
 BufferHealth health = CheckBufferHealth(ws);

 switch (health) {
 case BUFFER_HEALTHY:
 // Normal operation
 break;

 case BUFFER_WARNING:
 // Flush to prevent congestion
 ws_flush(ws->GetFd());
 break;

 case BUFFER_CRITICAL:
 // Emergency flush and delay
 ws_flush(ws->GetFd());
 OSTimeDly(TICKS_PER_SECOND / 4);
 break;
 }
}
```

## Prioritized Message Sending

```
typedef enum {
 MSG_PRIORITY_HIGH,
 MSG_PRIORITY_NORMAL,
 MSG_PRIORITY_LOW
} MessagePriority;

bool SendPrioritizedMessage(WebSocket *ws, const char *msg,
 int length, MessagePriority priority) {
 int available = ws->GetWriteSpace();

 // High priority messages get special treatment
 if (priority == MSG_PRIORITY_HIGH) {
 if (available < length) {
 // Force flush for high priority

```

```

 ws_flush(ws->GetFd());
 OSTimeDly(TICKS_PER_SECOND / 10);
 available = ws->GetWriteSpace();
 }

 // Send even if space is tight
 ws_write(ws->GetFd(), msg, length);
 ws_flush(ws->GetFd());
 return true;
}

// Normal/low priority - only send if space available
if (available >= length) {
 ws_write(ws->GetFd(), msg, length);
 return true;
}

printf("Dropped %s priority message (insufficient space)\n",
 priority == MSG_PRIORITY_NORMAL ? "normal" : "low");
return false;
}

```

### Diagnostic Logging

```

void LogWebSocketBufferState(WebSocket *ws, const char *context) {
 int available = ws->GetWriteSpace();
 uint32_t timestamp = Secs;

 printf("[%lu] %s: TX buffer = %d bytes available\n",
 timestamp, context, available);

 // Log to persistent storage or diagnostics system
 LogToFile(timestamp, context, available);
}

// Usage examples:
LogWebSocketBufferState(ws, "Before large transfer");
SendLargeData(ws, data, data_size);
LogWebSocketBufferState(ws, "After large transfer");

```

### 16.190.26.6 Ping()

```

int NB::WebSocket::Ping (
 uint32_t len,
 uint32_t * sentTick)

```

Send a [WebSocket](#) ping frame and record the timestamp.

#### Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>len</i>      | Length of optional ping payload data (typically 0 for standard ping) |
| <i>sentTick</i> | Pointer to variable to receive the tick count when ping was sent     |

#### Returns

int Returns 0 on success, negative value on failure

This function sends a [WebSocket](#) ping control frame to the peer and records the system tick count at the time of transmission. Ping frames are used to:

- Verify the connection is still alive (keepalive mechanism)
- Measure round-trip time (RTT) when paired with [WaitForPingReply\(\)](#)
- Detect network issues or dead connections
- Prevent idle connection timeouts

The ping frame may optionally include payload data (up to 125 bytes per [WebSocket](#) specification), though a zero-length ping is most common. The peer should respond with a pong frame containing the same payload.

The sentTick parameter receives the system tick count (from TimeTick) at the moment the ping was sent. This timestamp can be used to calculate round-trip time when the corresponding pong is received.

**Note**

Ping frames are control frames and have priority over data frames in the [WebSocket](#) protocol.  
According to RFC 6455, the peer must respond to a ping with a pong frame containing identical payload data.

**Warning**

Sending too many pings in rapid succession may be considered abusive by the peer and could result in connection closure.

**See also**

[WaitForPingReply\(\)](#) to wait for the corresponding pong response

[GetPingReplyTick\(\)](#) to retrieve the pong receive timestamp

**Example Usage****Expand for Example Usage****Examples****Basic Ping**

```
WebSocket *ws = GetActiveWebSocket();
uint32_t sent_tick;

// Send a ping with no payload
if (ws->Ping(0, &sent_tick) == 0) {
 printf("Ping sent at tick %lu\n", sent_tick);
} else {
 printf("Failed to send ping\n");
}
```

**Measure Round-Trip Time**

```
float MeasureRoundTripTime(WebSocket *ws) {
 uint32_t sent_tick;

 // Send ping
 if (ws->Ping(0, &sent_tick) != 0) {
 printf("Failed to send ping\n");
 return -1.0f;
 }

 // Wait for pong (with 5 second timeout)
 if (ws->WaitForPingReply(TICKS_PER_SECOND * 5) == 0) {
 uint32_t reply_tick;
 ws->GetPingReplyTick(&reply_tick);

 // Calculate RTT in milliseconds
 uint32_t rtt_ticks = reply_tick - sent_tick;
 float rtt_ms = (rtt_ticks * 1000.0f) / TICKS_PER_SECOND;

 printf("Round-trip time: %.2f ms\n", rtt_ms);
 return rtt_ms;
 } else {
 printf("Ping timeout - no pong received\n");
 return -1.0f;
 }
}
```

**Periodic Keepalive**

```
void KeepaliveTask(void *pd) {
 WebSocket *ws = (WebSocket *)pd;
 uint32_t sent_tick;

 while (1) {
 if (ws->GetState() == WS_STATE_OPEN) {
 // Send keepalive ping every 30 seconds
 if (ws->Ping(0, &sent_tick) == 0) {
 printf("Keepalive ping sent\n");
 } else {
 printf("WARNING: Keepalive ping failed\n");
 }
 }

 OSTimeDly(TICKS_PER_SECOND * 30);
 }
}
```

### Connection Health Check

```
bool VerifyConnectionAlive(WebSocket *ws) {
 uint32_t sent_tick;

 printf("Verifying connection health...\n");

 // Send ping
 if (ws->Ping(0, &sent_tick) != 0) {
 printf("ERROR: Cannot send ping\n");
 return false;
 }

 // Wait for response (3 second timeout)
 if (ws->WaitForPingReply(TICKS_PER_SECOND * 3) == 0) {
 printf("Connection verified: peer responsive\n");
 return true;
 } else {
 printf("ERROR: Connection appears dead (no pong)\n");
 return false;
 }
}
```

### Multiple Ping Attempts

```
bool PingWithRetry(WebSocket *ws, int max_attempts) {
 for (int i = 0; i < max_attempts; i++) {
 uint32_t sent_tick;

 printf("Ping attempt %d/%d\n", i + 1, max_attempts);

 if (ws->Ping(0, &sent_tick) == 0) {
 // Wait for pong
 if (ws->WaitForPingReply(TICKS_PER_SECOND * 2) == 0) {
 printf("Pong received on attempt %d\n", i + 1);
 return true;
 }
 }

 // Brief delay before retry
 OSTimeDly(TICKS_PER_SECOND / 2);
 }

 printf("All ping attempts failed\n");
 return false;
}
```

### Network Quality Monitoring

```
void MonitorNetworkQuality(WebSocket *ws) {
 const int SAMPLE_COUNT = 10;
 float rtt_samples[SAMPLE_COUNT];
 int successful_pings = 0;

 printf("Monitoring network quality (%d samples)...\n", SAMPLE_COUNT);

 for (int i = 0; i < SAMPLE_COUNT; i++) {
 uint32_t sent_tick;

 if (ws->Ping(0, &sent_tick) == 0) {
 if (ws->WaitForPingReply(TICKS_PER_SECOND * 2) == 0) {
 uint32_t reply_tick;
 ws->GetPingReplyTick(&reply_tick);

 uint32_t rtt_ticks = reply_tick - sent_tick;
 rtt_samples[successful_pings] =
 (rtt_ticks * 1000.0f) / TICKS_PER_SECOND;
 successful_pings++;
 }
 }

 OSTimeDly(TICKS_PER_SECOND);
 }

 if (successful_pings > 0) {
 // Calculate statistics
 float avg = 0, min = rtt_samples[0], max = rtt_samples[0];
 for (int i = 0; i < successful_pings; i++) {
 avg += rtt_samples[i];
 if (rtt_samples[i] < min) min = rtt_samples[i];
 if (rtt_samples[i] > max) max = rtt_samples[i];
 }
 avg /= successful_pings;

 printf("Network Quality Report:\n");
 printf(" Success rate: %d/%d (%.1f%%)\n",
 successful_pings, SAMPLE_COUNT, (float)successful_pings / SAMPLE_COUNT);
 }
}
```

```

 successful_pings, SAMPLE_COUNT,
 (successful_pings * 100.0f) / SAMPLE_COUNT);
 printf(" RTT: avg=%.2fms, min=%.2fms, max=%.2fms\n",
 avg, min, max);
} else {
 printf("ERROR: No successful pings\n");
}
}

```

### Ping with Payload

```

bool SendPingWithData(WebSocket *ws, const char *payload, uint32_t len) {
 uint32_t sent_tick;

 if (len > 125) {
 printf("ERROR: Ping payload too large (max 125 bytes)\n");
 return false;
 }

 // Send ping with custom payload
 if (ws->Ping(len, &sent_tick) == 0) {
 printf("Ping with %lu byte payload sent\n", len);
 return true;
 } else {
 printf("Failed to send ping\n");
 return false;
 }
}

```

### 16.190.26.7 Pong()

```

void NB::WebSocket::Pong (
 uint32_t len)

```

Send a pong frame in response to a ping.

#### Parameters

|            |                                                                    |
|------------|--------------------------------------------------------------------|
| <i>len</i> | Length of pong payload data (should match the ping payload length) |
|------------|--------------------------------------------------------------------|

This member function sends a [WebSocket](#) pong control frame, typically in response to a received ping frame from the peer. According to the [WebSocket](#) protocol (RFC 6455), when a ping is received, the endpoint must respond with a pong containing the same payload data.

Pong frames serve multiple purposes:

- Acknowledge receipt of ping frames (keepalive confirmation)
- Enable the peer to measure round-trip time
- Demonstrate connection liveness
- Satisfy protocol requirements for ping-pong exchanges

The [WebSocket](#) implementation typically handles ping frames automatically and sends pong responses without application intervention. However, this function allows manual pong transmission if needed for:

- Custom ping-pong protocols
- Testing and debugging
- Explicit control over pong timing
- Unsolicited pongs (though not recommended by the spec)

The length parameter should match the payload length of the ping being responded to. Most pings have zero-length payloads, so len=0 is most common. The maximum payload length for control frames is 125 bytes per the [WebSocket](#) specification.

**Note**

In most cases, the [WebSocket](#) implementation automatically handles incoming pings and sends pongs without requiring application code to call this function.

Pong frames are control frames and have priority over data frames in the [WebSocket](#) protocol.

Per RFC 6455, pong frames may be sent unsolicited, though this is uncommon and not recommended unless you have a specific reason.

**Warning**

The payload data must match the ping payload. This function sends a pong with the specified length, but the actual payload matching is handled internally by the [WebSocket](#) implementation.

**See also**

[Ping\(\)](#) to send ping frames

[WaitForPingReply\(\)](#) to wait for pong responses to sent pings

[GetPingReplyTick\(\)](#) to retrieve pong reception timestamp

**Example Usage****Expand for Example Usage****Examples****Manual Pong Response**

```
WebSocket *ws = GetActiveWebSocket();

// Send pong with no payload (most common case)
ws->Pong(0);
printf("Pong frame sent\n");
```

**Custom Ping Handler**

```
void HandlePingFrame(WebSocket *ws, uint32_t payload_length) {
 printf("Received ping with %lu byte payload\n", payload_length);

 // Send matching pong
 ws->Pong(payload_length);

 printf("Pong response sent\n");
}
```

**Delayed Pong for Testing**

```
void SendDelayedPong(WebSocket *ws, uint32_t delay_ms) {
 printf("Delaying pong by %lu ms...\n", delay_ms);

 // Simulate processing delay
 OSTimeDly((delay_ms * TICKS_PER_SECOND) / 1000);

 // Send pong
 ws->Pong(0);

 printf("Delayed pong sent\n");
}
```

**Pong with Payload**

```
void SendPongWithPayload(WebSocket *ws, const char *payload, uint32_t len) {
 if (len > 125) {
 printf("ERROR: Pong payload too large (max 125 bytes)\n");
 return;
 }

 // Note: This example assumes the WebSocket implementation
 // has been configured with the payload data to echo
 ws->Pong(len);

 printf("Pong sent with %lu byte payload\n", len);
}
```

### Monitored Pong Response

```
void MonitoredPongResponse(WebSocket *ws, uint32_t ping_received_tick) {
 uint32_t current_tick = TimeTick;
 uint32_t processing_time = current_tick - ping_received_tick;

 // Send pong
 ws->Pong(0);

 float processing_ms = (processing_time * 1000.0f) / TICKS_PER_SECOND;
 printf("Pong sent (processing time: %.3f ms)\n", processing_ms);
}
```

### Conditional Pong Sending

```
void ConditionalPong(WebSocket *ws, bool should_respond) {
 if (should_respond) {
 ws->Pong(0);
 printf("Pong sent in response to ping\n");
 } else {
 printf("Pong suppressed (testing mode)\n");
 }
}
```

### Unsolicited Pong (Heartbeat)

```
void SendUnsolicitedPong(WebSocket *ws) {
 // Per RFC 6455, unsolicited pongs are allowed
 // (though uncommon in practice)
 ws->Pong(0);

 printf("Unsolicited pong sent as heartbeat\n");
}

void HeartbeatTask(void *pd) {
 WebSocket *ws = (WebSocket *)pd;

 while (1) {
 if (ws->GetState() == WS_STATE_OPEN) {
 SendUnsolicitedPong(ws);
 }

 // Send heartbeat every 30 seconds
 OSTimeDly(TICKS_PER_SECOND * 30);
 }
}
```

### Pong Statistics Tracking

```
static uint32_t pong_count = 0;

void SendTrackedPong(WebSocket *ws) {
 ws->Pong(0);
 pong_count++;

 printf("Pong sent (total: %lu)\n", pong_count);

 // Log every 100 pongs
 if (pong_count % 100 == 0) {
 printf("Milestone: %lu pongs sent\n", pong_count);
 }
}
```

### State-Aware Pong Handling

```
void SafePongResponse(WebSocket *ws) {
 WS_State state = ws->GetState();

 if (state == WS_STATE_OPEN) {
 ws->Pong(0);
 printf("Pong sent (connection open)\n");
 } else {
 printf("Cannot send pong: connection not open (state: %d)\n", state);
 }
}
```



### Diagnostic Pong with Logging

```
void DiagnosticPong(WebSocket *ws, const char *context) {
 uint32_t timestamp = Secs;
 uint32_t send_tick = TimeTick;

 // Send pong
 ws->Pong(0);

 // Log the event
 printf("[%lu] Pong sent in context: %s (tick: %lu)\n",
 timestamp, context, send_tick);

 // Could also log to persistent storage
 LogEvent(timestamp, "PONG_SENT", context, send_tick);
}
```

### Automatic Ping-Pong Handler

```
// Note: This is typically handled automatically by the WebSocket library
// This example shows what happens internally
void OnPingReceived(WebSocket *ws, const uint8_t *payload, uint32_t len) {
 printf("Ping received with %lu byte payload\n", len);

 // Store payload for echo in pong (done internally)
 // SetPongPayload(payload, len);

 // Send pong with matching length
 ws->Pong(len);

 printf("Automatic pong response sent\n");
}
```

### Throttled Pong Responses

```
void ThrottledPong(WebSocket *ws) {
 static uint32_t last_pong_tick = 0;
 const uint32_t MIN_PONG_INTERVAL = TICKS_PER_SECOND / 10; // 100ms

 uint32_t current_tick = TimeTick;

 if ((current_tick - last_pong_tick) >= MIN_PONG_INTERVAL) {
 ws->Pong(0);
 last_pong_tick = current_tick;
 printf("Pong sent\n");
 } else {
 printf("Pong throttled (too soon after last pong)\n");
 }
}
```

#### 16.190.26.8 promote\_tcp\_ws()

```
static int NB::WebSocket::promote_tcp_ws (
 int tcp_fd) [static]
```

Promote a TCP socket to a [WebSocket](#) connection after successful handshake.

#### Parameters

|                           |                                           |
|---------------------------|-------------------------------------------|
| <code>tcp↔<br/>_fd</code> | The TCP socket file descriptor to promote |
|---------------------------|-------------------------------------------|

#### Returns

int Returns 0 on success, negative value on failure

This function converts an existing TCP connection into a [WebSocket](#) connection after the [WebSocket](#) handshake has been completed. It performs the necessary protocol upgrade and registers the connection in the [WebSocket](#) management system.

The promotion process typically involves:

- Validating the [WebSocket](#) handshake headers
- Switching the socket protocol from HTTP to [WebSocket](#)
- Registering the new [WebSocket](#) connection

- Configuring appropriate callbacks for [WebSocket](#) events

#### Note

This function should only be called after a successful [WebSocket](#) handshake. Calling it on an invalid or non-handshaked connection will fail.

#### Warning

The original TCP socket should not be used directly after promotion. All further communication must use [WebSocket](#) APIs.

#### Example Usage

---

### Expand for Example Usage

#### Examples

##### Basic Promotion

```
// After receiving WebSocket handshake on TCP connection
int tcp_fd = AcceptTcpConnection(); // Accept incoming TCP connection

// ... validate WebSocket handshake headers ...

// Promote TCP socket to WebSocket
int result = promote_tcp_ws(tcp_fd);
if (result == 0) {
 printf("Successfully promoted TCP socket %d to WebSocket\n", tcp_fd);
} else {
 printf("Failed to promote socket %d, error: %d\n", tcp_fd, result);
 close(tcp_fd);
}
```

##### HTTP Upgrade Handler

```
void HandleWebSocketUpgrade(int tcp_fd, HTTP_Request &request) {
 // Verify this is a WebSocket upgrade request
 if (request.GetHeader("Upgrade") != "websocket") {
 SendHttpRequest(tcp_fd, 400, "Bad Request");
 return;
 }

 // Send handshake response
 SendWebSocketHandshakeResponse(tcp_fd, request);

 // Promote to WebSocket
 if (promote_tcp_ws(tcp_fd) == 0) {
 printf("WebSocket connection established\n");
 // WebSocket is now ready for use
 } else {
 printf("ERROR: Failed to promote connection\n");
 close(tcp_fd);
 }
}
```

##### Complete Server Example

```
void WebSocketServerTask(void *pd) {
 int listen_fd = CreateTcpListenSocket(8080);

 while (1) {
 int tcp_fd = accept(listen_fd, nullptr, nullptr);

 if (tcp_fd > 0) {
 // Read HTTP upgrade request
 if (ValidateWebSocketHandshake(tcp_fd)) {
 // Handshake valid, promote to WebSocket
 if (promote_tcp_ws(tcp_fd) == 0) {
 printf("New WebSocket client connected\n");
 // Connection now managed by WebSocket system
 } else {
 printf("Failed to create WebSocket\n");
 close(tcp_fd);
 }
 } else {
 // Not a valid WebSocket request
 SendHttpRequest(tcp_fd, 400);
 close(tcp_fd);
 }
 }
 }
}
```

```
 }
 OSTimeDly(TICKS_PER_SECOND / 10);
}
}
```

### 16.190.26.9 ReadData()

```
int NB::WebSocket::ReadData (
 char * buf,
 int nbytes)
```

Read data from the [WebSocket](#) connection.

#### Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>buf</i>    | Pointer to buffer where received data will be stored |
| <i>nbytes</i> | Maximum number of bytes to read into the buffer      |

#### Returns

int Number of bytes actually read on success, 0 if no data available, negative value on error or connection closed

This member function reads data from the [WebSocket](#) connection into the provided buffer. It handles [WebSocket](#) frame decoding automatically, presenting the application data to the caller without protocol overhead.

Key characteristics:

- Automatic [WebSocket](#) frame parsing and payload extraction
- Handles fragmented messages transparently
- Processes control frames (ping, pong, close) internally
- Non-blocking operation when no data is available
- Removes [WebSocket](#) framing and masking

The function operates in non-blocking mode:

- Returns immediately with available data
- Returns 0 if no data is currently available
- Returns negative value on error or connection closure

Return value interpretation:

- Positive value: Number of bytes read successfully
- Zero: No data currently available (not an error)
- Negative: [Error](#) occurred or connection closed

#### Note

The buffer must be large enough to hold the requested number of bytes. Reading less than the available data will leave remaining data for subsequent reads.

This function performs non-blocking reads. For blocking reads with timeout, use `ReadDataWithTimeout()` or similar if available.

Control frames (ping, pong, close) are typically handled automatically and not returned to the application.

## Warning

Always check the return value. A return of 0 or negative indicates no data was read, which requires different handling than successful reads.

The buffer pointer must remain valid and the buffer must be large enough to hold nbytes.

## See also

[WriteData\(\)](#) for sending data over the [WebSocket](#)

[GetState\(\)](#) to check connection state before reading

## Example Usage

### Expand for Example Usage

#### Examples Basic Read

```
WebSocket *ws = GetActiveWebSocket();
char buffer[512];

int bytes_read = ws->ReadData(buffer, sizeof(buffer) - 1);
if (bytes_read > 0) {
 buffer[bytes_read] = '\0'; // Null terminate for string operations
 printf("Received %d bytes: %s\n", bytes_read, buffer);
} else if (bytes_read == 0) {
 printf("No data available\n");
} else {
 printf("Error reading data\n");
}
```

#### Read with Error Handling

```
int SafeRead(WebSocket *ws, char *buffer, int buffer_size) {
 if (buffer == nullptr || buffer_size <= 0) {
 printf("ERROR: Invalid buffer parameters\n");
 return -1;
 }

 if (ws->GetState() != WS_STATE_OPEN) {
 printf("ERROR: Connection not open\n");
 return -1;
 }

 int bytes_read = ws->ReadData(buffer, buffer_size - 1);

 if (bytes_read > 0) {
 buffer[bytes_read] = '\0';
 return bytes_read;
 } else if (bytes_read == 0) {
 // No data available, not an error
 return 0;
 } else {
 // Connection error or closed
 printf("Read error, closing connection\n");
 return -1;
 }
}
```

#### Continuous Reading Loop

```
void ReadTask(void *pd) {
 WebSocket *ws = (WebSocket *)pd;
 char buffer[1024];

 while (1) {
 int bytes_read = ws->ReadData(buffer, sizeof(buffer) - 1);

 if (bytes_read > 0) {
 buffer[bytes_read] = '\0';
 printf("Received: %s\n", buffer);

 // Process the received data
 ProcessMessage(buffer, bytes_read);
 } else if (bytes_read < 0) {
 printf("Connection error or closed\n");
 break;
 }
 }
}
```

```

 }

 // Brief delay to prevent CPU spinning
 OSTimeDly(TICKS_PER_SECOND / 100);
}

printf("Read task exiting\n");
}

```

### JSON Message Reading

```

bool ReadJsonMessage(WebSocket *ws, char *json_buffer, int buffer_size) {
 int bytes_read = ws->ReadData(json_buffer, buffer_size - 1);

 if (bytes_read > 0) {
 json_buffer[bytes_read] = '\0';

 // Validate it's JSON
 if (json_buffer[0] == '{' || json_buffer[0] == '[') {
 printf("Received JSON: %s\n", json_buffer);
 return true;
 } else {
 printf("WARNING: Non-JSON data received\n");
 return false;
 }
 }

 return false;
}

```

### Binary Data Reading

```

int ReadBinaryFrame(WebSocket *ws, uint8_t *data_buffer, int max_size) {
 int bytes_read = ws->ReadData((char *)data_buffer, max_size);

 if (bytes_read > 0) {
 printf("Received %d bytes of binary data:\n", bytes_read);

 // Display as hex
 for (int i = 0; i < bytes_read; i++) {
 printf("%02X ", data_buffer[i]);
 if ((i + 1) % 16 == 0) printf("\n");
 }
 printf("\n");
 }

 return bytes_read;
}

```

### Multi-Buffer Reading (Large Messages)

```

void ReadLargeMessage(WebSocket *ws) {
 const int CHUNK_SIZE = 256;
 char chunk[CHUNK_SIZE];
 int total_bytes = 0;

 printf("Reading large message in chunks...\n");

 while (1) {
 int bytes_read = ws->ReadData(chunk, CHUNK_SIZE - 1);

 if (bytes_read > 0) {
 chunk[bytes_read] = '\0';
 total_bytes += bytes_read;

 // Process this chunk
 ProcessChunk(chunk, bytes_read);
 } else if (bytes_read == 0) {
 // No more data available
 break;
 } else {
 printf("ERROR: Read failed\n");
 break;
 }
 }

 printf("Total bytes read: %d\n", total_bytes);
}

```

### Read with Manual Timeout

```
int ReadWithTimeout(WebSocket *ws, char *buffer, int buffer_size,
 uint32_t timeout_ms) {
 uint32_t start_tick = TimeTick;
 uint32_t timeout_ticks = (timeout_ms * TICKS_PER_SECOND) / 1000;

 while (1) {
 int bytes_read = ws->ReadData(buffer, buffer_size - 1);

 if (bytes_read > 0) {
 buffer[bytes_read] = '\0';
 return bytes_read;
 } else if (bytes_read < 0) {
 // Error
 return -1;
 }

 // Check timeout
 if ((TimeTick - start_tick) > timeout_ticks) {
 printf("Read timeout after %lu ms\n", timeout_ms);
 return 0; // Timeout
 }

 OSTimeDly(TICKS_PER_SECOND / 100);
 }
}
```

### Command Processing Pattern

```
void ProcessCommands(WebSocket *ws) {
 char command[256];

 while (1) {
 int bytes_read = ws->ReadData(command, sizeof(command) - 1);

 if (bytes_read > 0) {
 command[bytes_read] = '\0';

 // Process command and generate response
 char response[256];
 HandleCommand(command, response, sizeof(response));

 // Send response
 ws->WriteData(response, strlen(response));
 ws->Flush();

 } else if (bytes_read < 0) {
 break;
 }

 OSTimeDly(TICKS_PER_SECOND / 50);
 }
}
```

### Echo Server Implementation

```
void EchoServerTask(WebSocket *ws) {
 char buffer[512];

 printf("Echo server started\n");

 while (1) {
 int bytes_read = ws->ReadData(buffer, sizeof(buffer) - 1);

 if (bytes_read > 0) {
 buffer[bytes_read] = '\0';
 printf("Echo: %s\n", buffer);

 // Echo back
 int bytes_written = ws->WriteData(buffer, bytes_read);
 if (bytes_written > 0) {
 ws->Flush();
 }

 } else if (bytes_read < 0) {
 printf("Connection closed\n");
 break;
 }

 OSTimeDly(TICKS_PER_SECOND / 100);
 }
}
```

### Message Queue Implementation

```
#define MAX_MESSAGES 10

typedef struct {
 char data[256];
 int length;
} Message;

Message message_queue[MAX_MESSAGES];
int queue_head = 0;
int queue_tail = 0;

void ReadToQueue(WebSocket *ws) {
 char buffer[256];

 int bytes_read = ws->ReadData(buffer, sizeof(buffer) - 1);

 if (bytes_read > 0) {
 // Add to queue if space available
 int next_tail = (queue_tail + 1) % MAX_MESSAGES;

 if (next_tail != queue_head) {
 memcpy(message_queue[queue_tail].data, buffer, bytes_read);
 message_queue[queue_tail].length = bytes_read;
 queue_tail = next_tail;

 printf("Message queued (%d in queue)\n",
 (queue_tail - queue_head + MAX_MESSAGES) % MAX_MESSAGES);
 } else {
 printf("WARNING: Message queue full, dropping message\n");
 }
 }
}
```

### Statistical Monitoring

```
void MonitorReadStatistics(WebSocket *ws) {
 static uint32_t total_bytes = 0;
 static uint32_t message_count = 0;
 static uint32_t start_time = 0;

 if (start_time == 0) {
 start_time = Secs;
 }

 char buffer[1024];
 int bytes_read = ws->ReadData(buffer, sizeof(buffer) - 1);

 if (bytes_read > 0) {
 total_bytes += bytes_read;
 message_count++;

 uint32_t elapsed = Secs - start_time;
 if (elapsed > 0) {
 float bytes_per_sec = total_bytes / (float)elapsed;
 float msgs_per_sec = message_count / (float)elapsed;

 printf("Stats: %lu bytes (%lu msgs) in %lu sec "
 "(%.1f B/s, %.1f msg/s)\n",
 total_bytes, message_count, elapsed,
 bytes_per_sec, msgs_per_sec);
 }
 }
}
```

#### 16.190.26.10 RunSkippedCallback()

```
static void NB::WebSocket::RunSkippedCallback () [static]
```

Executes the callback function when a scheduled run is skipped.

### 16.190.27 Description

This static function is invoked when a scheduled task or operation cannot be executed at its intended time (e.g., due to system overload, timing conflicts, or the previous execution still running). It allows the system to handle skip events gracefully by notifying registered handlers.

### 16.190.28 Usage Notes

- This is a static member function and does not require an object instance

- Typically called internally by the scheduler when a run is skipped
- Should be lightweight to avoid further timing issues
- May log skip events or increment skip counters for monitoring

### 16.190.29 Thread Safety

Implementation should be thread-safe if called from multiple scheduler contexts.

### 16.190.30 See Also

- Related scheduling functions
- Skip event handling documentation

---

### Expand for Example Usage

#### Examples

##### Example 1: Basic Skip Callback Registration

```
// In your scheduler class
class TaskScheduler {
private:
 static void RunSkippedCallBack() {
 printf("Task execution was skipped\r\n");
 }

public:
 void RegisterSkipHandler() {
 // Register the callback with scheduler
 SetSkipCallback(RunSkippedCallBack);
 }
};
```

##### Example 2: Skip Tracking Implementation

```
class PeriodicTask {
private:
 static uint32_t skipCount;

 static void RunSkippedCallBack() {
 skipCount++;
 printf("Task skipped (total: %lu)\r\n", skipCount);

 // Log if skip count is high
 if (skipCount > 10) {
 printf("WARNING: High skip count detected\r\n");
 }
 }

public:
 static uint32_t GetSkipCount() { return skipCount; }
};

uint32_t PeriodicTask::skipCount = 0;
```

##### Example 3: Detailed Skip Event Handling

```
class MonitoredTask {
private:
 static void RunSkippedCallBack() {
 uint32_t currentTime = Secs;

 // Log skip event with timestamp
 printf("[%lu] Task execution skipped\r\n", currentTime);

 // Record in event log
 LogEvent("TASK_SKIP", currentTime);

 // Optionally trigger recovery action
 if (GetSystemLoad() < 50) {
 printf("System load normal, attempting retry\r\n");
 ScheduleRetry();
 }
 }
};
```

---



### 16.190.30.1 setoption()

```
int NB::WebSocket::setoption (
 int option)
```

Set a socket option on this [WebSocket](#) connection.

#### Parameters

|               |                                                                                      |
|---------------|--------------------------------------------------------------------------------------|
| <i>option</i> | The socket option to set (e.g., WS_SO_TEXT, WS_SO_BINARY, SO_KEEPALIVE, TCP_NODELAY) |
|---------------|--------------------------------------------------------------------------------------|

#### Returns

int Returns 0 on success, negative value on failure

This member function sets a specific socket option on the [WebSocket](#) connection. Options control various aspects of the [WebSocket](#) behavior and the underlying TCP socket configuration.

Common WebSocket-specific options:

- WS\_SO\_TEXT: Treat data as text frames (default)
- WS\_SO\_BINARY: Treat data as binary frames
- WS\_SO\_COMPRESS: Enable per-message compression (if supported)

Common TCP socket options:

- SO\_KEEPALIVE: Enable TCP keepalive packets
- TCP\_NODELAY: Disable Nagle's algorithm for low-latency
- SO\_REUSEADDR: Allow address reuse
- SO\_SNDBUF/SO\_RCVBUF: Adjust send/receive buffer sizes

Options can be combined using bitwise OR to set multiple options simultaneously. Setting an already-set option has no adverse effect.

#### Note

Options should be set before or shortly after connection establishment for best results. Some options may not take effect if changed mid-connection.

#### Warning

Setting incompatible options (e.g., both WS\_SO\_TEXT and WS\_SO\_BINARY) may result in undefined behavior. The last option set typically takes precedence.

#### See also

[cloption\(\)](#) for clearing socket options

[getoption\(\)](#) for querying current option state

#### Example Usage

---

### Expand for Example Usage

#### Examples

### Set Binary Mode

```
WebSocket *ws = new WebSocket("ws://example.com");

// Enable binary frame mode for sending binary data
if (ws->setoption(WS_SO_BINARY) == 0) {
 printf("Binary mode enabled\n");

 // Now send binary data
 uint8_t data[] = {0x01, 0x02, 0x03, 0x04};
 ws->write((char *)data, sizeof(data));
} else {
 printf("Failed to enable binary mode\n");
}
```

### Enable TCP Keepalive

```
WebSocket *ws = GetActiveWebSocket();

// Enable keepalive to detect dead connections
if (ws->setoption(SO_KEEPALIVE) == 0) {
 printf("TCP keepalive enabled\n");
} else {
 printf("Failed to enable keepalive\n");
}
```

### Configure for Low Latency

```
void ConfigureLowLatencyWebSocket(WebSocket *ws) {
 // Disable Nagle's algorithm for immediate packet transmission
 if (ws->setoption(TCP_NODELAY) == 0) {
 printf("Low latency mode enabled\n");
 } else {
 printf("WARNING: Could not enable TCP_NODELAY\n");
 }
}
```

### Set Multiple Options

```
WebSocket *ws = new WebSocket("ws://realtime-server.com");

// Configure for real-time binary streaming
int result = 0;

// Set binary mode
result |= ws->setoption(WS_SO_BINARY);

// Enable low latency
result |= ws->setoption(TCP_NODELAY);

// Enable keepalive
result |= ws->setoption(SO_KEEPALIVE);

if (result == 0) {
 printf("All options set successfully\n");
} else {
 printf("WARNING: Some options failed to set\n");
}
```

### Switch Between Text and Binary Modes

```
void SwitchToBinaryMode(WebSocket *ws) {
 // Clear text mode if set
 ws->cloption(WS_SO_TEXT);

 // Set binary mode
 if (ws->setoption(WS_SO_BINARY) == 0) {
 printf("Switched to binary mode\n");
 }
}

void SwitchToTextMode(WebSocket *ws) {
 // Clear binary mode if set
 ws->cloption(WS_SO_BINARY);

 // Set text mode
 if (ws->setoption(WS_SO_TEXT) == 0) {
 printf("Switched to text mode\n");
 }
}
```

### Conditional Option Setting

```
void ConfigureWebSocketForDataType(WebSocket *ws, bool is_binary) {
 if (is_binary) {
 ws->setoption(WS_SO_BINARY);
 printf("Configured for binary data\n");
 } else {
 ws->setoption(WS_SO_TEXT);
 printf("Configured for text data\n");
 }
}
```

### Production vs Development Configuration

```
void ConfigureWebSocket(WebSocket *ws, bool production_mode) {
 if (production_mode) {
 // Production: optimize for reliability
 ws->setoption(SO_KEEPALIVE);
 printf("Production configuration applied\n");
 } else {
 // Development: optimize for debugging
 ws->setoption(TCP_NODELAY);
 printf("Development configuration applied\n");
 }
}
```

### Enable Compression (if supported)

```
WebSocket *ws = new WebSocket("ws://example.com");

// Try to enable per-message compression
if (ws->setoption(WS_SO_COMPRESS) == 0) {
 printf("Compression enabled - reduced bandwidth usage\n");
} else {
 printf("Compression not available or not supported\n");
}
```

#### 16.190.30.2 WaitForPingReply()

```
int NB::WebSocket::WaitForPingReply (
 uint32_t timeout)
```

Block waiting for a pong reply to a previously sent ping.

#### Parameters

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| <i>timeout</i> | Timeout value in system ticks (use TICKS_PER_SECOND for conversion) |
|----------------|---------------------------------------------------------------------|

#### Returns

int Returns 0 if pong received within timeout, negative value on timeout or error

This function blocks the calling task until a pong frame is received in response to a previously sent ping, or until the specified timeout expires. It is typically called immediately after [Ping\(\)](#) to wait for the peer's response.

The function is useful for:

- Measuring round-trip time with known timeout bounds
- Verifying connection responsiveness
- Implementing synchronous ping-pong interactions
- Detecting dead or slow connections

The timeout parameter is specified in system ticks. Use TICKS\_PER\_SECOND to convert from seconds:

- 1 second: TICKS\_PER\_SECOND
- 5 seconds: TICKS\_PER\_SECOND \* 5
- 500 ms: TICKS\_PER\_SECOND / 2

Return value interpretation:

- 0: Pong received successfully within timeout
- Negative: Timeout expired or error occurred

**Note**

This function blocks the calling task. Ensure the timeout is appropriate for your application to avoid excessive blocking.

A pong should typically be received within a few hundred milliseconds on a healthy network. Timeouts of 2-5 seconds are common.

**Warning**

Do not call this function without first calling [Ping\(\)](#). The behavior is undefined if no ping has been sent.

Very long timeouts may make the application appear unresponsive.

**See also**

[Ping\(\)](#) to send the ping frame that this function waits for

[GetPingReplyTick\(\)](#) to retrieve the exact pong receive timestamp

**Example Usage****Expand for Example Usage****Examples****Basic Ping-Pong Sequence**

```
WebSocket *ws = GetActiveWebSocket();
uint32_t sent_tick;

// Send ping
if (ws->Ping(0, &sent_tick) == 0) {
 printf("Ping sent, waiting for pong...\n");

 // Wait up to 3 seconds for pong
 if (ws->WaitForPingReply(TICKS_PER_SECOND * 3) == 0) {
 printf("Pong received!\n");
 } else {
 printf("Timeout: No pong received\n");
 }
}
```

**Measure Exact Round-Trip Time**

```
void MeasurePingLatency(WebSocket *ws) {
 uint32_t sent_tick;

 // Send ping and record timestamp
 if (ws->Ping(0, &sent_tick) != 0) {
 printf("Failed to send ping\n");
 return;
 }

 // Wait for pong (5 second timeout)
 int result = ws->WaitForPingReply(TICKS_PER_SECOND * 5);

 if (result == 0) {
 uint32_t reply_tick;
 ws->GetPingReplyTick(&reply_tick);

 // Calculate precise RTT
 uint32_t rtt_ticks = reply_tick - sent_tick;
 float rtt_ms = (rtt_ticks * 1000.0f) / TICKS_PER_SECOND;

 printf("Latency: %.3f ms\n", rtt_ms);
 } else {
 printf("Ping timeout - connection may be dead\n");
 }
}
```

### Connection Liveness Test

```
bool TestConnectionLiveness(WebSocket *ws) {
 uint32_t sent_tick;

 printf("Testing connection liveness...\n");

 // Send ping
 if (ws->Ping(0, &sent_tick) != 0) {
 printf("ERROR: Cannot send ping\n");
 return false;
 }

 // Short timeout for quick response detection
 int result = ws->WaitForPingReply(TICKS_PER_SECOND * 2);

 if (result == 0) {
 printf("Connection is ALIVE\n");
 return true;
 } else {
 printf("Connection is DEAD or SLOW\n");
 return false;
 }
}
```

### Multiple Timeout Strategy

```
bool SmartPingWait(WebSocket *ws) {
 uint32_t sent_tick;

 if (ws->Ping(0, &sent_tick) != 0) {
 return false;
 }

 // First try short timeout (fast network)
 if (ws->WaitForPingReply(TICKS_PER_SECOND) == 0) {
 printf("Fast response received\n");
 return true;
 }

 printf("No quick response, waiting longer...\n");

 // Try longer timeout (slow network)
 if (ws->WaitForPingReply(TICKS_PER_SECOND * 5) == 0) {
 printf("Slow response received\n");
 return true;
 }

 printf("No response - connection timeout\n");
 return false;
}
```

### Periodic Health Monitoring

```
void HealthMonitorTask(void *pd) {
 WebSocket *ws = (WebSocket *)pd;
 int consecutive_failures = 0;
 const int MAX_FAILURES = 3;

 while (1) {
 if (ws->GetState() == WS_STATE_OPEN) {
 uint32_t sent_tick;

 if (ws->Ping(0, &sent_tick) == 0) {
 int result = ws->WaitForPingReply(TICKS_PER_SECOND * 3);

 if (result == 0) {
 // Success - reset failure counter
 consecutive_failures = 0;
 printf("Health check passed\n");
 } else {
 // Timeout
 consecutive_failures++;
 printf("Health check failed (%d/%d)\n",
 consecutive_failures, MAX_FAILURES);

 if (consecutive_failures >= MAX_FAILURES) {
 printf("Connection declared dead, closing\n");
 ws->close();
 break;
 }
 }
 }
 }
 }

 // Check every 30 seconds
}
```

```

 OSTimeDly(TICKS_PER_SECOND * 30);
 }
}

```

### Non-blocking Alternative Pattern

```

// For comparison - non-blocking approach
void NonBlockingPingCheck(WebSocket *ws) {
 static uint32_t ping_sent_tick = 0;
 static bool ping_pending = false;

 if (!ping_pending) {
 // Send new ping
 if (ws->Ping(0, &ping_sent_tick) == 0) {
 ping_pending = true;
 printf("Ping sent\n");
 }
 } else {
 // Check if pong received
 uint32_t reply_tick;
 if (ws->GetPingReplyTick(&reply_tick) == 0) {
 if (reply_tick > ping_sent_tick) {
 // Pong received
 uint32_t rtt = reply_tick - ping_sent_tick;
 printf("Pong received, RTT: %lu ticks\n", rtt);
 ping_pending = false;
 }
 }

 // Check for timeout
 if ((TimeTick - ping_sent_tick) > (TICKS_PER_SECOND * 5)) {
 printf("Ping timeout\n");
 ping_pending = false;
 }
 }
}

```

### Connection Quality Assessment

```

typedef enum {
 QUALITY_EXCELLENT, // < 50ms
 QUALITY_GOOD, // 50-200ms
 QUALITY_FAIR, // 200-500ms
 QUALITY_POOR // > 500ms or timeout
} ConnectionQuality;

ConnectionQuality AssessConnectionQuality(WebSocket *ws) {
 uint32_t sent_tick;

 if (ws->Ping(0, &sent_tick) != 0) {
 return QUALITY_POOR;
 }

 if (ws->WaitForPingReply(TICKS_PER_SECOND * 2) != 0) {
 // Timeout
 return QUALITY_POOR;
 }

 uint32_t reply_tick;
 ws->GetPingReplyTick(&reply_tick);

 float rtt_ms = ((reply_tick - sent_tick) * 1000.0f) / TICKS_PER_SECOND;

 if (rtt_ms < 50.0f) {
 return QUALITY_EXCELLENT;
 } else if (rtt_ms < 200.0f) {
 return QUALITY_GOOD;
 } else if (rtt_ms < 500.0f) {
 return QUALITY_FAIR;
 } else {
 return QUALITY_POOR;
 }
}

```

#### 16.190.30.3 WriteData()

```

int NB::WebSocket::WriteData (
 const char * buf,
 int nbytes)

```

Write data to the [WebSocket](#) connection.

## Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>buf</i>    | Pointer to buffer containing data to write |
| <i>nbytes</i> | Number of bytes to write from the buffer   |

## Returns

int Number of bytes actually written on success, negative value on error

This member function writes data to the [WebSocket](#) connection with automatic frame encoding. The function handles the [WebSocket](#) protocol overhead, allowing the application to send raw data that will be properly framed for transmission.

Key characteristics:

- Automatically creates [WebSocket](#) frames with appropriate headers
- Applies masking if required by protocol direction (client to server)
- Handles message fragmentation for large payloads automatically
- Buffers data for efficient transmission
- Respects current frame type setting (text or binary mode)

The function operates according to the [WebSocket](#) options set via [setoption\(\)](#):

- WS\_SO\_TEXT: Data is sent as text frames (default)
- WS\_SO\_BINARY: Data is sent as binary frames

Return value interpretation:

- Positive value: Number of bytes successfully queued for transmission
- Zero: No bytes written (unusual, may indicate buffer full)
- Negative: [Error](#) occurred (connection closed, invalid parameters, etc.)

## Note

The function may return successfully even if data hasn't been physically transmitted yet. Use [Flush\(\)](#) to ensure immediate transmission.

For large messages, the function may fragment data into multiple [WebSocket](#) frames automatically according to the protocol specification.

Written data is buffered in the transmit buffer. Check [GetWriteSpace\(\)](#) before writing if buffer space is a concern.

## Warning

Always check the return value. A negative return indicates the write failed and the connection may need to be closed.

The buffer pointer must remain valid for the duration of the call. Do not pass stack buffers that will go out of scope.

## See also

[ReadData\(\)](#) for reading data from the [WebSocket](#)  
[Flush\(\)](#) to force immediate transmission of buffered data  
[GetWriteSpace\(\)](#) to check available buffer space  
[setoption\(\)](#) to configure text or binary frame mode

## Example Usage

### Expand for Example Usage

#### Examples

##### Basic Text Write

```
WebSocket *ws = GetActiveWebSocket();
const char *message = "Hello, WebSocket!";

int bytes_written = ws->WriteData(message, strlen(message));
if (bytes_written > 0) {
 printf("Sent %d bytes\n", bytes_written);
} else {
 printf("Failed to send message, error: %d\n", bytes_written);
}
```

##### Write with Error Handling

```
int SafeWrite(WebSocket *ws, const char *data, int length) {
 if (data == nullptr || length <= 0) {
 printf("ERROR: Invalid write parameters\n");
 return -1;
 }

 if (ws->GetState() != WS_STATE_OPEN) {
 printf("ERROR: Connection not open\n");
 return -1;
 }

 int bytes_written = ws->WriteData(data, length);

 if (bytes_written == length) {
 // All data written successfully
 return bytes_written;
 } else if (bytes_written > 0) {
 // Partial write
 printf("WARNING: Partial write (%d of %d bytes)\n",
 bytes_written, length);
 return bytes_written;
 } else {
 // Write failed
 printf("ERROR: Write failed\n");
 return -1;
 }
}
```

##### Binary Data Transmission

```
bool SendBinaryData(WebSocket *ws, const uint8_t *data, int length) {
 // Ensure binary mode is enabled
 ws->setoption(WS_SO_BINARY);

 printf("Sending %d bytes of binary data...\n", length);

 int bytes_written = ws->WriteData((const char *)data, length);

 if (bytes_written == length) {
 printf("Binary data sent successfully\n");
 return true;
 } else {
 printf("ERROR: Binary send failed\n");
 return false;
 }
}
```

##### JSON Message Transmission

```
bool SendJsonMessage(WebSocket *ws, const char *json_str) {
 // Ensure text mode for JSON
 ws->setoption(WS_SO_TEXT);

 int length = strlen(json_str);
 printf("Sending JSON (%d bytes): %s\n", length, json_str);

 int bytes_written = ws->WriteData(json_str, length);

 if (bytes_written == length) {
 ws->Flush(); // Ensure immediate delivery
 return true;
 } else {
 printf("ERROR: JSON send failed\n");
 return false;
 }
}
```



### Formatted Message Writing

```
int WriteFormattedMessage(WebSocket *ws, const char *format, ...) {
 char buffer[512];
 va_list args;

 va_start(args, format);
 int length = vsnprintf(buffer, sizeof(buffer), format, args);
 va_end(args);

 if (length > 0 && length < (int)sizeof(buffer)) {
 int result = ws->WriteData(buffer, length);

 if (result > 0) {
 ws->Flush();
 printf("Formatted message sent: %s\n", buffer);
 }

 return result;
 } else {
 printf("ERROR: Message too large for buffer\n");
 return -1;
 }
}

// Usage:
WriteFormattedMessage(ws, "{\"sensor\":\"%s\",\"value\":%.2f}",
 "temperature", 23.45);
```

### Large Data Transfer with Chunking

```
bool WriteLargeData(WebSocket *ws, const char *data, int total_length) {
 const int CHUNK_SIZE = 1024;
 int bytes_sent = 0;

 printf("Sending large payload: %d bytes\n", total_length);

 while (bytes_sent < total_length) {
 // Check buffer space
 int space = ws->GetWriteSpace();
 if (space < 256) {
 printf("Buffer low, flushing...\n");
 ws->Flush();
 OSTimeDly(TICKS_PER_SECOND / 10);
 }

 // Calculate chunk size
 int remaining = total_length - bytes_sent;
 int to_send = (remaining > CHUNK_SIZE) ? CHUNK_SIZE : remaining;

 // Write chunk
 int bytes_written = ws->WriteData(data + bytes_sent, to_send);

 if (bytes_written > 0) {
 bytes_sent += bytes_written;
 printf("Progress: %d/%d bytes (%.1f%%)\n",
 bytes_sent, total_length,
 (bytes_sent * 100.0f) / total_length);
 } else {
 printf("ERROR: Write failed at %d bytes\n", bytes_sent);
 return false;
 }
 }

 ws->Flush();
 printf("Large data transfer complete\n");
 return true;
}
```

### Streaming Sensor Data

```
void StreamSensorReadings(WebSocket *ws) {
 char buffer[128];

 while (IsStreamingEnabled()) {
 // Read sensors
 float temp = ReadTemperature();
 float humidity = ReadHumidity();
 float pressure = ReadPressure();

 // Format as JSON
 snprintf(buffer, sizeof(buffer),
 "{\"temp\":%.2f,\"humidity\":%.2f,\"pressure\":%.2f}",
 temp, humidity, pressure);

 // Write and flush for real-time delivery
```

```

 int written = ws->WriteData(buffer, strlen(buffer));
 if (written > 0) {
 ws->Flush();
 } else {
 printf("WARNING: Failed to send sensor data\n");
 }
 OSTimeDly(TICKS_PER_SECOND);
 }
}

```

### Broadcast to Multiple WebSockets

```

void BroadcastMessage(const char *message) {
 int length = strlen(message);
 int success_count = 0;
 int fail_count = 0;

 printf("Broadcasting: %s\n", message);

 for (int i = 0; i < WS_MAX_SOCKETS; i++) {
 WebSocket *ws = &WebSocket::WSocketStructs[i];

 if (ws->GetState() == WS_STATE_OPEN) {
 int written = ws->WriteData(message, length);

 if (written == length) {
 ws->Flush();
 success_count++;
 } else {
 fail_count++;
 }
 }
 }

 printf("Broadcast complete: %d sent, %d failed\n",
 success_count, fail_count);
}

```

### Write with Buffer Space Check

```

bool WriteWithSpaceCheck(WebSocket *ws, const char *data, int length) {
 int available = ws->GetWriteSpace();

 if (available < length) {
 printf("Insufficient buffer space: need %d, have %d\n",
 length, available);

 // Flush and retry
 ws->Flush();
 OSTimeDly(TICKS_PER_SECOND / 10);

 available = ws->GetWriteSpace();
 if (available < length) {
 printf("ERROR: Still insufficient space after flush\n");
 return false;
 }
 }

 int written = ws->WriteData(data, length);
 return (written == length);
}

```

### Rate-Limited Writing

```

void WriteWithRateLimit(WebSocket *ws, const char *data, int length,
 int max_bytes_per_second) {
 int bytes_sent = 0;
 uint32_t start_time = Secs;

 while (bytes_sent < length) {
 // Calculate how much we can send this second
 uint32_t elapsed = Secs - start_time;
 int allowed = (elapsed + 1) * max_bytes_per_second;
 int to_send = allowed - bytes_sent;

 if (to_send > (length - bytes_sent)) {
 to_send = length - bytes_sent;
 }

 if (to_send > 0) {
 int written = ws->WriteData(data + bytes_sent, to_send);
 if (written > 0) {
 bytes_sent += written;
 }
 }
 }
}

```

```
 }

 OSTimeDly(TICKS_PER_SECOND / 10);
}

ws->Flush();
printf("Rate-limited transfer complete: %d bytes\n", bytes_sent);
}
```

#### 16.190.30.4 ws\_clroption()

```
static int NB::WebSocket::ws_clroption (
 int fd,
 int option) [static]
```

Clear a socket option on a [WebSocket](#) connection.

##### Parameters

|               |                                                              |
|---------------|--------------------------------------------------------------|
| <i>fd</i>     | The <a href="#">WebSocket</a> file descriptor                |
| <i>option</i> | The socket option to clear (e.g., SO_KEEPALIVE, TCP_NODELAY) |

##### Returns

int Returns 0 on success, negative value on failure

This static wrapper function clears (disables) a specific socket option on the underlying socket associated with the [WebSocket](#) connection. It is the counterpart to [ws\\_setoption\(\)](#) and allows you to disable previously enabled options. Clearing an option returns the socket to its default behavior for that particular setting. This is useful when you need to dynamically adjust socket behavior based on runtime conditions or when reverting configuration changes.

##### Note

Clearing an option that was never set has no effect and typically succeeds.

##### See also

[ws\\_setoption\(\)](#) for setting socket options

[ws\\_getoption\(\)](#) for querying current option state

##### Example Usage

---

#### Expand for Example Usage

##### Examples

###### Disable TCP Keepalive

```
int ws_fd = GetWebSocketFd();

// Disable keepalive (maybe to reduce network traffic)
if (ws_clroption(ws_fd, SO_KEEPALIVE) == 0) {
 printf("Keepalive disabled for WebSocket fd %d\n", ws_fd);
} else {
 printf("Failed to disable keepalive\n");
}
```

###### Re-enable Nagle's Algorithm

```
// Clear TCP_NODELAY to re-enable Nagle's algorithm
int ws_fd = GetActiveWebSocketFd();

if (ws_clroption(ws_fd, TCP_NODELAY) == 0) {
 printf("Nagle's algorithm re-enabled - buffering mode\n");
} else {
 printf("ERROR: Could not clear TCP_NODELAY option\n");
}
```

### Dynamic Option Management

```
void ToggleWebSocketMode(int ws_fd, bool lowLatencyMode) {
 if (lowLatencyMode) {
 // Enable low-latency settings
 ws_setopt(ws_fd, TCP_NODELAY);
 printf("Switched to low-latency mode\n");
 } else {
 // Disable low-latency, allow buffering
 ws_clroption(ws_fd, TCP_NODELAY);
 printf("Switched to standard mode\n");
 }
}
```

### Cleanup Before Reconnection

```
void ResetWebSocketOptions(int ws_fd) {
 // Clear all custom options before reconnecting
 ws_clroption(ws_fd, SO_KEEPALIVE);
 ws_clroption(ws_fd, TCP_NODELAY);
 ws_clroption(ws_fd, SO_REUSEADDR);

 printf("WebSocket options reset to defaults\n");
}
```

#### 16.190.30.5 ws\_externalclose()

```
static int NB::WebSocket::ws_externalclose (
 int fd) [static]
```

Close a [WebSocket](#) connection from an external caller.

#### Parameters

|           |                                                        |
|-----------|--------------------------------------------------------|
| <i>fd</i> | The <a href="#">WebSocket</a> file descriptor to close |
|-----------|--------------------------------------------------------|

#### Returns

int Returns 0 on success, negative value on failure

This static wrapper function provides a mechanism to close a [WebSocket](#) connection from external code (outside the [WebSocket](#) class). It performs a clean shutdown of the [WebSocket](#), including:

- Sending a [WebSocket](#) close frame to the peer
- Flushing any pending data in buffers
- Releasing associated resources
- Removing the connection from the [WebSocket](#) registry

This function differs from a simple socket [close\(\)](#) by properly handling the [WebSocket](#) protocol's closing handshake, ensuring graceful disconnection.

#### Note

After calling this function, the file descriptor should not be used for any further operations.

#### Warning

Do not call standard [close\(\)](#) on a [WebSocket](#) file descriptor. Always use this function to ensure proper cleanup of [WebSocket](#) resources.

#### See also

[ws\\_flush\(\)](#) to flush buffers before closing

## Example Usage

### Expand for Example Usage

#### Examples

##### Basic Close

```
int ws_fd = GetActiveWebSocketFd();

// Close the WebSocket connection
if (ws_externalclose(ws_fd) == 0) {
 printf("WebSocket fd %d closed successfully\n", ws_fd);
} else {
 printf("Failed to close WebSocket fd %d\n", ws_fd);
}
```

##### Close with Flush

```
void CleanCloseWebSocket(int ws_fd) {
 // Ensure all pending data is sent before closing
 ws_flush(ws_fd);

 // Now close the connection
 int result = ws_externalclose(ws_fd);
 if (result == 0) {
 printf("WebSocket cleanly closed\n");
 } else {
 printf("ERROR: Failed to close WebSocket, code: %d\n", result);
 }
}
```

##### Timeout-based Close

```
void CloseIdleWebSockets(uint32_t maxIdleSeconds) {
 uint32_t currentTime = Secs;

 for (int i = 0; i < MAX_WEBSOCKETS; i++) {
 WebSocket *ws = GetWebSocketByIndex(i);
 if (ws != nullptr) {
 uint32_t idleTime = currentTime - ws->GetLastActivityTime();

 if (idleTime > maxIdleSeconds) {
 int fd = ws->GetFd();
 printf("Closing idle WebSocket fd %d (idle %lu sec)\n",
 fd, idleTime);
 ws_externalclose(fd);
 }
 }
 }
}
```

##### Error Condition Close

```
void HandleWebSocketError(int ws_fd, int error_code) {
 printf("WebSocket error %d on fd %d, closing connection\n",
 error_code, ws_fd);

 // Attempt to close gracefully
 if (ws_externalclose(ws_fd) != 0) {
 printf("WARNING: Error during close, connection may not be clean\n");
 }
}
```

##### Shutdown All WebSockets

```
void ShutdownAllWebSockets() {
 printf("Shutting down all WebSocket connections...\n");

 int closed_count = 0;
 int error_count = 0;

 for (int i = 0; i < MAX_WEBSOCKETS; i++) {
 WebSocket *ws = GetWebSocketByIndex(i);
 if (ws != nullptr) {
 int fd = ws->GetFd();

 // Flush before closing
 ws_flush(fd);

 if (ws_externalclose(fd) == 0) {
 closed_count++;
 } else {
 error_count++;
 printf("ERROR: Failed to close WebSocket fd %d\n", fd);
 }
 }
 }
}
```

```

 }
}

printf("Shutdown complete: %d closed, %d errors\n",
 closed_count, error_count);
}

```

### Conditional Close

```

void CloseWebSocketIfCondition(int ws_fd, bool force) {
 WebSocket *ws = GetWebSocketRecord(ws_fd);

 if (ws == nullptr) {
 printf("Invalid WebSocket fd: %d\n", ws_fd);
 return;
 }

 // Check if there's pending data
 if (!force && ws->HasPendingData()) {
 printf("WebSocket has pending data, flushing before close\n");
 ws_flush(ws_fd);
 OSTimeDly(TICKS_PER_SECOND / 10); // Brief delay for flush
 }

 ws_externalclose(ws_fd);
}

```

#### 16.190.30.6 ws\_flush()

```

static void NB::WebSocket::ws_flush (
 int fd) [static]

```

Flush the transmit buffer of a [WebSocket](#) connection.

#### Parameters

|           |                                               |
|-----------|-----------------------------------------------|
| <i>fd</i> | The <a href="#">WebSocket</a> file descriptor |
|-----------|-----------------------------------------------|

This static wrapper function forces any pending data in the [WebSocket](#)'s transmit buffer to be sent immediately to the peer. Normally, the [WebSocket](#) implementation may buffer small amounts of data to optimize network usage, but this function ensures immediate transmission.

Common use cases include:

- Before closing a connection to ensure all data is sent
- After sending critical messages that need immediate delivery
- When implementing real-time protocols with strict timing requirements
- Before entering low-power or idle states

#### Note

This function blocks until the buffer is flushed or a timeout occurs. The actual transmission completion depends on network conditions and the peer's receive capability.

#### Warning

Excessive flushing can reduce network efficiency. Use only when immediate transmission is required.

#### See also

[ws\\_externalclose\(\)](#) which should be called after flushing when closing

#### Example Usage

---

### Expand for Example Usage Examples

### Basic Flush

```
int ws_fd = GetActiveWebSocketFd();

// Send some data
SendWebSocketData(ws_fd, "Important message");

// Ensure it's sent immediately
ws_flush(ws_fd);
printf("Data flushed to network\n");
```

### Flush Before Close

```
void SafeCloseWebSocket(int ws_fd) {
 // Send final message
 SendWebSocketText(ws_fd, "Goodbye!");

 // Ensure message is sent before closing
 ws_flush(ws_fd);

 // Small delay to allow network transmission
 OSTimeDly(TICKS_PER_SECOND / 20);

 // Now safe to close
 ws_externalclose(ws_fd);
}
```

### Critical Message Delivery

```
void SendCriticalAlert(int ws_fd, const char *alert_msg) {
 // Send alert message
 SendWebSocketText(ws_fd, alert_msg);

 // Flush immediately - this is critical
 ws_flush(ws_fd);

 printf("Critical alert sent and flushed: %s\n", alert_msg);
}
```

### Real-time Data Streaming

```
void StreamSensorData(int ws_fd) {
 while (IsStreamingActive()) {
 // Read sensor
 float temperature = ReadTemperature();
 float humidity = ReadHumidity();

 // Format and send
 char buffer[128];
 snprintf(buffer, sizeof(buffer),
 "{\temp\":%.2f,\humidity\":%.2f}",
 temperature, humidity);
 SendWebSocketText(ws_fd, buffer);

 // Flush for real-time delivery
 ws_flush(ws_fd);

 // Wait before next reading
 OSTimeDly(TICKS_PER_SECOND);
 }
}
```

### Batch Send with Flush

```
void SendBatchData(int ws_fd, const char **messages, int count) {
 for (int i = 0; i < count; i++) {
 SendWebSocketText(ws_fd, messages[i]);
 }

 // Flush all messages at once
 ws_flush(ws_fd);

 printf("Sent and flushed %d messages\n", count);
}
```

### Command-Response Pattern

```
void HandleWebSocketCommand(int ws_fd, const char *command) {
 char response[256];

 // Process command
 ProcessCommand(command, response, sizeof(response));

 // Send response
 SendWebSocketText(ws_fd, response);
}
```

```

 // Flush to ensure client receives response promptly
 ws_flush(ws_fd);

 printf("Command processed and response sent\n");
 }

```

### Periodic Heartbeat

```

void WebSocketHeartbeatTask(void *pd) {
 int ws_fd = GetWebSocketFd();

 while (IsConnected(ws_fd)) {
 // Send ping/heartbeat
 SendWebSocketPing(ws_fd);

 // Flush to ensure timely delivery
 ws_flush(ws_fd);

 // Wait for next heartbeat interval
 OSTimeDly(TICKS_PER_SECOND * 30); // 30 second heartbeat
 }
}

```

### Error Recovery

```

void RecoverFromBufferOverflow(int ws_fd) {
 printf("Buffer overflow detected, flushing...\n");

 // Attempt to flush buffer
 ws_flush(ws_fd);

 // Check if connection is still valid
 int options = ws_getoption(ws_fd);
 if (options < 0) {
 printf("Connection lost after flush, closing\n");
 ws_externalclose(ws_fd);
 } else {
 printf("Buffer flushed successfully, connection stable\n");
 }
}

```

## 16.190.30.7 ws\_getoption()

```

static int NB::WebSocket::ws_getoption (
 int fd) [static]

```

Get the current socket options for a [WebSocket](#) connection.

### Parameters

|           |                                               |
|-----------|-----------------------------------------------|
| <i>fd</i> | The <a href="#">WebSocket</a> file descriptor |
|-----------|-----------------------------------------------|

### Returns

int A bitmask of currently enabled socket options, or negative value on failure

This static wrapper function retrieves the current state of socket options for the underlying socket associated with the [WebSocket](#) connection. It returns a bitmask where each bit represents a specific socket option's state (enabled/disabled).

The returned value can be tested against specific option flags to determine if particular options are currently enabled. This is useful for:

- Debugging socket configuration
- Verifying option settings
- Conditionally changing options based on current state
- Logging socket configuration



**Note**

The exact interpretation of the returned bitmask depends on the underlying socket implementation and available options.

**See also**

[ws\\_setopt\(\)](#) for setting socket options

[ws\\_clroption\(\)](#) for clearing socket options

**Example Usage****Expand for Example Usage****Examples****Query Current Options**

```
int ws_fd = GetWebSocketFd();

int options = ws_getoption(ws_fd);
if (options >= 0) {
 printf("Current socket options: 0x%08X\n", options);
} else {
 printf("Failed to get socket options\n");
}
```

**Check Specific Option**

```
int ws_fd = GetActiveWebSocketFd();
int options = ws_getoption(ws_fd);

if (options >= 0) {
 if (options & SO_KEEPALIVE) {
 printf("Keepalive is ENABLED\n");
 } else {
 printf("Keepalive is DISABLED\n");
 }

 if (options & TCP_NODELAY) {
 printf("Nagle's algorithm is DISABLED (low latency mode)\n");
 } else {
 printf("Nagle's algorithm is ENABLED (buffering mode)\n");
 }
}
```

**Conditional Option Setting**

```
void EnsureKeepaliveEnabled(int ws_fd) {
 int options = ws_getoption(ws_fd);

 if (options < 0) {
 printf("ERROR: Cannot read socket options\n");
 return;
 }

 // Only set keepalive if not already enabled
 if (!(options & SO_KEEPALIVE)) {
 ws_setopt(ws_fd, SO_KEEPALIVE);
 printf("Keepalive was disabled, now enabled\n");
 } else {
 printf("Keepalive already enabled\n");
 }
}
```

**Diagnostic Logging**

```
void LogWebSocketConfiguration(int ws_fd) {
 int options = ws_getoption(ws_fd);

 if (options < 0) {
 printf("ERROR: Could not retrieve socket options\n");
 return;
 }

 printf("=== WebSocket Configuration (fd %d) ===\n", ws_fd);
 printf("SO_KEEPALIVE: %s\n", (options & SO_KEEPALIVE) ? "ON" : "OFF");
 printf("TCP_NODELAY: %s\n", (options & TCP_NODELAY) ? "ON" : "OFF");
 printf("SO_REUSEADDR: %s\n", (options & SO_REUSEADDR) ? "ON" : "OFF");
 printf("Raw bitmask: 0x%08X\n", options);
 printf("=====\n");
}
```

### Option Comparison

```
void CompareWebSocketOptions(int ws_fd1, int ws_fd2) {
 int options1 = ws_getoption(ws_fd1);
 int options2 = ws_getoption(ws_fd2);

 if (options1 < 0 || options2 < 0) {
 printf("ERROR: Could not retrieve options for comparison\n");
 return;
 }

 if (options1 == options2) {
 printf("Both WebSockets have identical options\n");
 } else {
 printf("WebSocket options differ:\n");
 printf(" WS1: 0x%08X\n", options1);
 printf(" WS2: 0x%08X\n", options2);
 printf(" XOR: 0x%08X\n", options1 ^ options2);
 }
}
```

### 16.190.30.8 ws\_read()

```
static int NB::WebSocket::ws_read (
 int fd,
 char * buf,
 int nbytes) [static]
```

Read data from a [WebSocket](#) connection.

#### Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>fd</i>     | The <a href="#">WebSocket</a> file descriptor        |
| <i>buf</i>    | Pointer to buffer where received data will be stored |
| <i>nbytes</i> | Maximum number of bytes to read into the buffer      |

#### Returns

int Number of bytes actually read on success, 0 if no data available, negative value on error or connection closed

This static wrapper function reads data from a [WebSocket](#) connection into the provided buffer. It handles [WebSocket](#) frame decoding automatically, presenting the application data to the caller without protocol overhead.

The function behavior includes:

- Automatic [WebSocket](#) frame parsing and payload extraction
- Handling of fragmented messages
- Processing of control frames (ping, pong, close)
- Non-blocking read when no data is available

Return value interpretation:

- Positive value: Number of bytes read successfully
- Zero: No data currently available (non-blocking)
- Negative: [Error](#) occurred or connection closed

#### Note

The buffer must be large enough to hold the requested number of bytes. Reading less than the available data will leave remaining data for subsequent reads.

### Warning

Always check the return value. A return of 0 or negative indicates no data was read, which may require different handling than partial reads.

### See also

[ws\\_write\(\)](#) for sending data over the [WebSocket](#)

### Example Usage

#### Expand for Example Usage

##### Examples Basic Read

```
int ws_fd = GetActiveWebSocketFd();
char buffer[512];

int bytes_read = ws_read(ws_fd, buffer, sizeof(buffer) - 1);
if (bytes_read > 0) {
 buffer[bytes_read] = '\0'; // Null terminate for string operations
 printf("Received %d bytes: %s\n", bytes_read, buffer);
} else if (bytes_read == 0) {
 printf("No data available\n");
} else {
 printf("Error reading from WebSocket\n");
}
```

##### Read with Error Handling

```
int SafeWebSocketRead(int ws_fd, char *buffer, int buffer_size) {
 if (buffer == nullptr || buffer_size <= 0) {
 printf("ERROR: Invalid buffer parameters\n");
 return -1;
 }

 int bytes_read = ws_read(ws_fd, buffer, buffer_size - 1);

 if (bytes_read > 0) {
 buffer[bytes_read] = '\0';
 return bytes_read;
 } else if (bytes_read == 0) {
 // No data available, not an error
 return 0;
 } else {
 // Connection error or closed
 printf("WebSocket read error, closing connection\n");
 ws_externalclose(ws_fd);
 return -1;
 }
}
```

##### Continuous Reading Loop

```
void WebSocketReadTask(void *pd) {
 int ws_fd = GetWebSocketFd();
 char buffer[1024];

 while (1) {
 int bytes_read = ws_read(ws_fd, buffer, sizeof(buffer) - 1);

 if (bytes_read > 0) {
 buffer[bytes_read] = '\0';
 printf("Received: %s\n", buffer);

 // Process the received data
 ProcessWebSocketMessage(buffer, bytes_read);
 } else if (bytes_read < 0) {
 printf("Connection closed or error\n");
 break;
 }

 // Brief delay to prevent CPU spinning
 OSTimeDly(TICKS_PER_SECOND / 100);
 }

 printf("WebSocket read task exiting\n");
}
```

### JSON Message Reading

```
bool ReadJsonMessage(int ws_fd, char *json_buffer, int buffer_size) {
 int bytes_read = ws_read(ws_fd, json_buffer, buffer_size - 1);

 if (bytes_read > 0) {
 json_buffer[bytes_read] = '\0';

 // Validate it's valid JSON
 if (json_buffer[0] == '{' || json_buffer[0] == '[') {
 printf("Received JSON: %s\n", json_buffer);
 return true;
 } else {
 printf("WARNING: Non-JSON data received\n");
 return false;
 }
 }

 return false;
}
```

### Binary Data Reading

```
int ReadBinaryFrame(int ws_fd, uint8_t *data_buffer, int max_size) {
 int bytes_read = ws_read(ws_fd, (char *)data_buffer, max_size);

 if (bytes_read > 0) {
 printf("Received %d bytes of binary data\n", bytes_read);

 // Process binary data
 for (int i = 0; i < bytes_read; i++) {
 printf("%02X ", data_buffer[i]);
 }
 printf("\n");
 }

 return bytes_read;
}
```

### Multi-buffer Reading

```
void ReadLargeMessage(int ws_fd) {
 const int CHUNK_SIZE = 256;
 char chunk[CHUNK_SIZE];
 int total_bytes = 0;

 printf("Reading large message in chunks...\n");

 while (1) {
 int bytes_read = ws_read(ws_fd, chunk, CHUNK_SIZE - 1);

 if (bytes_read > 0) {
 chunk[bytes_read] = '\0';
 total_bytes += bytes_read;

 // Process this chunk
 ProcessChunk(chunk, bytes_read);
 } else if (bytes_read == 0) {
 // No more data available
 break;
 } else {
 printf("ERROR: Read failed\n");
 break;
 }
 }

 printf("Total bytes read: %d\n", total_bytes);
}
```

### Timeout-based Reading

```
int ReadWithTimeout(int ws_fd, char *buffer, int buffer_size,
 uint32_t timeout_ms) {
 uint32_t start_tick = TimeTick;
 uint32_t timeout_ticks = (timeout_ms * TICKS_PER_SECOND) / 1000;

 while (1) {
 int bytes_read = ws_read(ws_fd, buffer, buffer_size - 1);

 if (bytes_read > 0) {
 buffer[bytes_read] = '\0';
 return bytes_read;
 } else if (bytes_read < 0) {
 // Error
 }
 }
}
```

```
 return -1;
 }

 // Check timeout
 if ((TimeTick - start_tick) > timeout_ticks) {
 printf("Read timeout after %lu ms\n", timeout_ms);
 return 0; // Timeout
 }

 OSTimeDly(TICKS_PER_SECOND / 100);
}
}
```

### 16.190.30.9 ws\_read\_notify()

```
static void NB::WebSocket::ws_read_notify (
 int fd) [static]
```

TCP read notification callback invoked when data is available on the [WebSocket](#).

### 16.190.31 Description

This static callback function is invoked by the NetBurner TCP/IP stack when data becomes available to read on the [WebSocket](#) file descriptor. It handles incoming [WebSocket](#) frames, processes control frames (ping, pong, close), and delivers data frames to the application layer.

### 16.190.32 Parameters

#### Parameters

|           |                                                           |
|-----------|-----------------------------------------------------------|
| <i>fd</i> | The file descriptor of the TCP socket with available data |
|-----------|-----------------------------------------------------------|

### 16.190.33 Usage Notes

- Automatically called by the network stack, not invoked directly by user code
- Should process all available data to avoid notification storms
- Must handle partial frames and buffering appropriately
- Should be non-blocking to prevent stalling the network stack
- Typically registered via `SetSocketReadNotify()` or similar

### 16.190.34 WebSocket Frame Handling

This callback must handle:

- Data frames (text/binary)
- Control frames (ping, pong, close)
- Frame fragmentation
- Opcode validation

### 16.190.35 Thread Safety

Called from the network stack context. Ensure thread-safe access to shared resources.

## 16.190.36 See Also

- [ws\\_write\\_notify\(\)](#) - Companion write notification callback
- [WebSocket](#) frame processing documentation
- TCP notification system

---

## Expand for Example Usage

### Examples

#### Example 1: Basic Read Notification Setup

```
class WebSocketHandler {
private:
 static void ws_read_notify(int fd) {
 char buffer[1024];
 int bytesRead = read(fd, buffer, sizeof(buffer));

 if (bytesRead > 0) {
 printf("Received %d bytes on fd %d\r\n", bytesRead, fd);
 ProcessWebSocketFrame(buffer, bytesRead);
 }
 else if (bytesRead == 0) {
 printf("Connection closed on fd %d\r\n", fd);
 close(fd);
 }
 }

public:
 void RegisterNotifications(int sockfd) {
 // Register read callback with TCP stack
 SetSocketReadNotify(sockfd, ws_read_notify);
 }
};
```

#### Example 2: WebSocket Frame Processing

```
static void ws_read_notify(int fd) {
 uint8_t frameBuffer[2048];
 int bytesRead = read(fd, frameBuffer, sizeof(frameBuffer));

 if (bytesRead <= 0) {
 HandleSocketError(fd, bytesRead);
 return;
 }

 // Parse WebSocket frame header
 bool fin = (frameBuffer[0] & 0x80) != 0;
 uint8_t opcode = frameBuffer[0] & 0x0F;

 switch (opcode) {
 case WS_OPCODE_TEXT:
 ProcessTextFrame(frameBuffer, bytesRead);
 break;
 case WS_OPCODE_BINARY:
 ProcessBinaryFrame(frameBuffer, bytesRead);
 break;
 case WS_OPCODE_PING:
 SendPong(fd, frameBuffer, bytesRead);
 break;
 case WS_OPCODE_CLOSE:
 HandleCloseFrame(fd);
 break;
 }
}
```

#### Example 3: Buffered Read with Error Handling

```
class WebSocketConnection {
private:
 static std::map<int, FrameBuffer> readBuffers;

 static void ws_read_notify(int fd) {
 const int CHUNK_SIZE = 512;
 uint8_t chunk[CHUNK_SIZE];

 // Read all available data
 while (true) {
 int bytesRead = read(fd, chunk, CHUNK_SIZE);

 if (bytesRead > 0) {
 // Append to frame buffer
 readBuffers[fd].Append(chunk, bytesRead);
 }
 }
 }
};
```

```

 // Try to parse complete frames
 while (readBuffers[fd].HasCompleteFrame()) {
 WebSocketFrame frame = readBuffers[fd].ExtractFrame();
 ProcessFrame(fd, frame);
 }
 }
 else if (bytesRead == 0) {
 printf("Connection closed gracefully: fd=%d\r\n", fd);
 CleanupConnection(fd);
 break;
 }
 else {
 // EWOULDBLOCK means no more data available
 if (errno == EWOULDBLOCK) break;

 printf("Read error on fd=%d: %d\r\n", fd, errno);
 CleanupConnection(fd);
 break;
 }
}
};

```

### 16.190.36.1 ws\_readwto()

```

static int NB::WebSocket::ws_readwto (
 int fd,
 char * buf,
 int nbytes,
 int timeout) [static]

```

Read data from a [WebSocket](#) connection with timeout.

#### Parameters

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| <i>fd</i>      | The <a href="#">WebSocket</a> file descriptor                       |
| <i>buf</i>     | Pointer to buffer where received data will be stored                |
| <i>nbytes</i>  | Maximum number of bytes to read into the buffer                     |
| <i>timeout</i> | Timeout value in system ticks (use TICKS_PER_SECOND for conversion) |

#### Returns

int Number of bytes actually read on success (positive value), 0 if timeout occurred with no data, negative value on error or connection closed

This static wrapper function reads data from a [WebSocket](#) connection with a specified timeout period. Unlike [ws\\_read\(\)](#) which returns immediately if no data is available, this function will wait up to the specified timeout for data to arrive.

The timeout mechanism allows for:

- Blocking reads with maximum wait time
- Prevention of indefinite blocking
- Periodic polling with controlled delays
- Responsive handling of slow or intermittent connections

Return value interpretation:

- Positive value: Number of bytes read successfully
- Zero: Timeout expired with no data received
- Negative: [Error](#) occurred or connection closed

The timeout parameter is specified in system ticks. Use TICKS\_PER\_SECOND to convert from seconds:

- 1 second: TICKS\_PER\_SECOND
- 500 ms: TICKS\_PER\_SECOND / 2
- 100 ms: TICKS\_PER\_SECOND / 10

**Note**

The buffer must be large enough to hold the requested number of bytes.

This function may return before the timeout if data becomes available or if an error occurs.

**Warning**

A timeout of 0 is equivalent to non-blocking mode (immediate return). Very large timeout values may block the calling task for extended periods.

**See also**

[ws\\_read\(\)](#) for non-blocking [read](#) operation

[ws\\_write\(\)](#) for sending data over the [WebSocket](#)

**Example Usage****Expand for Example Usage****Examples****Basic Read with Timeout**

```
int ws_fd = GetActiveWebSocketFd();
char buffer[512];

// Wait up to 5 seconds for data
int bytes_read = ws_readwto(ws_fd, buffer, sizeof(buffer) - 1,
 TICKS_PER_SECOND * 5);

if (bytes_read > 0) {
 buffer[bytes_read] = '\0';
 printf("Received %d bytes: %s\n", bytes_read, buffer);
} else if (bytes_read == 0) {
 printf("Timeout: No data received within 5 seconds\n");
} else {
 printf("Error reading from WebSocket\n");
}
```

**Request-Response Pattern**

```
bool SendRequestAndWaitForResponse(int ws_fd, const char *request) {
 char response[256];

 // Send request
 if (ws_write(ws_fd, request, strlen(request)) <= 0) {
 printf("Failed to send request\n");
 return false;
 }
 ws_flush(ws_fd);

 // Wait up to 10 seconds for response
 int bytes_read = ws_readwto(ws_fd, response, sizeof(response) - 1,
 TICKS_PER_SECOND * 10);

 if (bytes_read > 0) {
 response[bytes_read] = '\0';
 printf("Response received: %s\n", response);
 return true;
 } else if (bytes_read == 0) {
 printf("ERROR: Response timeout\n");
 return false;
 } else {
 printf("ERROR: Connection error\n");
 return false;
 }
}
```

**Polling Loop with Timeout**

```
void WebSocketPollingTask(void *pd) {
 int ws_fd = GetWebSocketFd();
 char buffer[1024];

 while (1) {
 // Poll with 1 second timeout
 int bytes_read = ws_readwto(ws_fd, buffer, sizeof(buffer) - 1,
 TICKS_PER_SECOND);
 }
```



```

 if (bytes_read > 0) {
 buffer[bytes_read] = '\0';
 printf("Data received: %s\n", buffer);
 ProcessWebSocketData(buffer, bytes_read);
 } else if (bytes_read == 0) {
 // Timeout - no data, continue polling
 // Could perform other tasks here
 } else {
 // Error - connection likely closed
 printf("Connection error, exiting\n");
 break;
 }
 }
}

```

### Short Timeout for Responsive UI

```

void CheckWebSocketWithShortTimeout(int ws_fd) {
 char buffer[256];

 // Very short timeout (100ms) for responsive checking
 int bytes_read = ws_readwto(ws_fd, buffer, sizeof(buffer) - 1,
 TICKS_PER_SECOND / 10);

 if (bytes_read > 0) {
 buffer[bytes_read] = '\0';
 UpdateUI(buffer);
 } else if (bytes_read == 0) {
 // Quick timeout, continue normal operation
 } else {
 // Connection error
 HandleConnectionError();
 }
}

```

### Multi-attempt Read with Retry

```

int ReadWithRetry(int ws_fd, char *buffer, int buffer_size,
 int attempts, uint32_t timeout_per_attempt) {
 for (int i = 0; i < attempts; i++) {
 int bytes_read = ws_readwto(ws_fd, buffer, buffer_size - 1,
 timeout_per_attempt);

 if (bytes_read > 0) {
 // Success
 buffer[bytes_read] = '\0';
 printf("Read successful on attempt %d\n", i + 1);
 return bytes_read;
 } else if (bytes_read == 0) {
 // Timeout, try again
 printf("Attempt %d timed out, retrying...\n", i + 1);
 } else {
 // Error, don't retry
 printf("Read error on attempt %d\n", i + 1);
 return -1;
 }
 }

 printf("All %d attempts failed\n", attempts);
 return 0; // All attempts timed out
}

// Usage:
char data[512];
// Try 3 times, 2 seconds each
int result = ReadWithRetry(ws_fd, data, sizeof(data), 3,
 TICKS_PER_SECOND * 2);

```

### Blocking Read Until Complete Message

```

bool ReadCompleteMessage(int ws_fd, char *buffer, int expected_size) {
 int total_read = 0;
 const int TIMEOUT = TICKS_PER_SECOND * 30; // 30 second total timeout

 printf("Reading complete message (%d bytes expected)...\n",
 expected_size);

 while (total_read < expected_size) {
 int remaining = expected_size - total_read;
 }
}

```

```

 int bytes_read = ws_readwto(ws_fd, buffer + total_read,
 remaining, TIMEOUT);

 if (bytes_read > 0) {
 total_read += bytes_read;
 printf("Progress: %d/%d bytes\n", total_read, expected_size);
 } else if (bytes_read == 0) {
 printf("ERROR: Timeout waiting for data\n");
 return false;
 } else {
 printf("ERROR: Read error\n");
 return false;
 }
}

buffer[total_read] = '\0';
printf("Complete message received\n");
return true;
}

```

### Adaptive Timeout Based on Activity

```

void AdaptiveReadLoop(int ws_fd) {
 char buffer[512];
 uint32_t timeout = TICKS_PER_SECOND; // Start with 1 second
 int consecutive_timeouts = 0;

 while (1) {
 int bytes_read = ws_readwto(ws_fd, buffer, sizeof(buffer) - 1,
 timeout);

 if (bytes_read > 0) {
 // Data received - reset to short timeout
 buffer[bytes_read] = '\0';
 ProcessData(buffer, bytes_read);
 timeout = TICKS_PER_SECOND;
 consecutive_timeouts = 0;
 } else if (bytes_read == 0) {
 // Timeout - increase timeout for next attempt
 consecutive_timeouts++;

 if (consecutive_timeouts > 3) {
 // Multiple timeouts, extend timeout to 5 seconds
 timeout = TICKS_PER_SECOND * 5;
 printf("Extended timeout due to inactivity\n");
 }
 } else {
 // Error
 printf("Connection error\n");
 break;
 }
 }
}

```

#### 16.190.36.2 ws\_setopt()

```

static int NB::WebSocket::ws_setopt (
 int fd,
 int option) [static]

```

Set a socket option on a [WebSocket](#) connection.

#### Parameters

|               |                                                            |
|---------------|------------------------------------------------------------|
| <i>fd</i>     | The <a href="#">WebSocket</a> file descriptor              |
| <i>option</i> | The socket option to set (e.g., SO_KEEPALIVE, TCP_NODELAY) |

#### Returns

int Returns 0 on success, negative value on failure

This static wrapper function sets a specific socket option on the underlying socket associated with the [WebSocket](#) connection. It provides a convenient interface to modify socket behavior without directly accessing the underlying TCP socket.

Common socket options include:

- `SO_KEEPALIVE`: Enable TCP keepalive packets
- `TCP_NODELAY`: Disable Nagle's algorithm for low-latency
- `SO_REUSEADDR`: Allow address reuse
- `SO_SNDBUF/SO_RCVBUF`: Adjust send/receive buffer sizes

**Note**

This function operates on the underlying TCP socket of the [WebSocket](#). Some options may not be applicable or may have unexpected behavior in [WebSocket](#) mode.

**Warning**

Setting incompatible socket options may disrupt [WebSocket](#) communication. Use with caution and test thoroughly.

**Example Usage****Expand for Example Usage****Examples****Enable TCP Keepalive**

```
WebSocket *ws = new WebSocket("ws://example.com");
int fd = ws->GetFd();

// Enable keepalive to detect dead connections
if (ws_setopt(fd, SO_KEEPALIVE) == 0) {
 printf("Keepalive enabled for WebSocket fd %d\n", fd);
} else {
 printf("Failed to enable keepalive\n");
}
```

**Disable Nagle's Algorithm**

```
// For low-latency applications, disable Nagle's algorithm
int ws_fd = GetActiveWebSocketFd();

if (ws_setopt(ws_fd, TCP_NODELAY) == 0) {
 printf("TCP_NODELAY enabled - reduced latency mode\n");
} else {
 printf("ERROR: Could not set TCP_NODELAY option\n");
}
```

**Multiple Options Setup**

```
void ConfigureWebSocketConnection(int ws_fd) {
 // Configure multiple socket options for optimal performance

 // Enable keepalive
 if (ws_setopt(ws_fd, SO_KEEPALIVE) != 0) {
 printf("Warning: Could not enable keepalive\n");
 }

 // Disable Nagle for low latency
 if (ws_setopt(ws_fd, TCP_NODELAY) != 0) {
 printf("Warning: Could not disable Nagle's algorithm\n");
 }

 printf("WebSocket connection configured\n");
}
```

**16.190.36.3 ws\_write()**

```
static int NB::WebSocket::ws_write (
 int fd,
 const char * buf,
 int nbytes) [static]
```

Write data to a [WebSocket](#) connection.

### Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>fd</i>     | The <a href="#">WebSocket</a> file descriptor |
| <i>buf</i>    | Pointer to buffer containing data to send     |
| <i>nbytes</i> | Number of bytes to write from the buffer      |

### Returns

int Number of bytes actually written on success, negative value on error

This static wrapper function writes data to a [WebSocket](#) connection with automatic [WebSocket](#) frame encoding. The function handles the protocol overhead, allowing the application to send raw data that will be properly framed for [WebSocket](#) transmission.

The function behavior includes:

- Automatic [WebSocket](#) frame creation with proper headers
- Masking of data (if required by protocol direction)
- Handling of message fragmentation for large payloads
- Buffering and flow control

Return value interpretation:

- Positive value: Number of bytes successfully queued for transmission
- Negative: [Error](#) occurred (connection closed, buffer full, invalid parameters)

### Note

The function may return successfully even if data hasn't been physically transmitted yet. Use [ws\\_flush\(\)](#) to ensure immediate transmission.

For large messages, the function may fragment the data into multiple [WebSocket](#) frames automatically.

### Warning

Always check the return value. A negative return indicates the write failed and the connection may need to be closed.

### See also

[ws\\_read\(\)](#) for reading data from the [WebSocket](#)

[ws\\_flush\(\)](#) to force immediate transmission of buffered data

### Example Usage

---

### Expand for Example Usage

#### Examples

##### Basic Write

```
int ws_fd = GetActiveWebSocketFd();
const char *message = "Hello, WebSocket!";

int bytes_written = ws_write(ws_fd, message, strlen(message));
if (bytes_written > 0) {
 printf("Sent %d bytes\n", bytes_written);
} else {
 printf("Failed to send message\n");
}
```

### Write with Error Handling

```

int SafeWebSocketWrite(int ws_fd, const char *data, int length) {
 if (data == nullptr || length <= 0) {
 printf("ERROR: Invalid write parameters\n");
 return -1;
 }

 int bytes_written = ws_write(ws_fd, data, length);

 if (bytes_written == length) {
 // All data written successfully
 return bytes_written;
 } else if (bytes_written > 0) {
 // Partial write
 printf("WARNING: Partial write (%d of %d bytes)\n",
 bytes_written, length);
 return bytes_written;
 } else {
 // Write failed
 printf("ERROR: Write failed, closing connection\n");
 ws_externalclose(ws_fd);
 return -1;
 }
}

```

### Write with Flush

```

void SendImmediateMessage(int ws_fd, const char *message) {
 int length = strlen(message);
 int bytes_written = ws_write(ws_fd, message, length);

 if (bytes_written > 0) {
 // Flush to ensure immediate delivery
 ws_flush(ws_fd);
 printf("Message sent and flushed: %s\n", message);
 } else {
 printf("Failed to send message\n");
 }
}

```

### JSON Message Transmission

```

bool SendJsonMessage(int ws_fd, const char *json_str) {
 int length = strlen(json_str);

 printf("Sending JSON (%d bytes): %s\n", length, json_str);

 int bytes_written = ws_write(ws_fd, json_str, length);

 if (bytes_written == length) {
 ws_flush(ws_fd); // Ensure JSON is sent immediately
 return true;
 } else {
 printf("ERROR: JSON send failed\n");
 return false;
 }
}

```

### Binary Data Transmission

```

int SendBinaryData(int ws_fd, const uint8_t *data, int data_length) {
 printf("Sending %d bytes of binary data...\n", data_length);

 int bytes_written = ws_write(ws_fd, (const char *)data, data_length);

 if (bytes_written == data_length) {
 printf("Binary data sent successfully\n");
 return bytes_written;
 } else if (bytes_written > 0) {
 printf("WARNING: Only sent %d of %d bytes\n",
 bytes_written, data_length);
 return bytes_written;
 } else {
 printf("ERROR: Binary send failed\n");
 return -1;
 }
}

```

## Formatted Message Transmission

```
void SendFormattedMessage(int ws_fd, const char *format, ...) {
 char buffer[512];
 va_list args;

 va_start(args, format);
 int length = vsnprintf(buffer, sizeof(buffer), format, args);
 va_end(args);

 if (length > 0 && length < (int)sizeof(buffer)) {
 int bytes_written = ws_write(ws_fd, buffer, length);

 if (bytes_written > 0) {
 ws_flush(ws_fd);
 printf("Formatted message sent: %s\n", buffer);
 } else {
 printf("Failed to send formatted message\n");
 }
 } else {
 printf("ERROR: Message too large for buffer\n");
 }
}

// Usage:
SendFormattedMessage(ws_fd, "{\"sensor\":\"%s\",\"value\":%.2f}",
 "temperature", 23.45);
```

## Large Data Transmission

```
bool SendLargePayload(int ws_fd, const char *data, int total_length) {
 const int CHUNK_SIZE = 1024;
 int bytes_sent = 0;

 printf("Sending large payload: %d bytes\n", total_length);

 while (bytes_sent < total_length) {
 int remaining = total_length - bytes_sent;
 int to_send = (remaining > CHUNK_SIZE) ? CHUNK_SIZE : remaining;

 int bytes_written = ws_write(ws_fd, data + bytes_sent, to_send);

 if (bytes_written > 0) {
 bytes_sent += bytes_written;
 printf("Progress: %d/%d bytes\n", bytes_sent, total_length);
 } else {
 printf("ERROR: Write failed at %d bytes\n", bytes_sent);
 return false;
 }

 // Brief delay between chunks
 OSTimeDly(TICKS_PER_SECOND / 100);
 }

 ws_flush(ws_fd);
 printf("Large payload sent successfully\n");
 return true;
}
```

## Broadcast to Multiple WebSockets

```
void BroadcastMessage(const char *message) {
 int length = strlen(message);
 int success_count = 0;
 int fail_count = 0;

 printf("Broadcasting message to all WebSockets: %s\n", message);

 for (int i = 0; i < MAX_WEBSOCKETS; i++) {
 WebSocket *ws = GetWebSocketByIndex(i);
 if (ws != nullptr) {
 int fd = ws->GetFd();
 int bytes_written = ws_write(fd, message, length);

 if (bytes_written == length) {
 ws_flush(fd);
 success_count++;
 } else {
 fail_count++;
 printf("WARNING: Failed to send to WebSocket %d\n", i);
 }
 }
 }

 printf("Broadcast complete: %d success, %d failed\n",
 success_count, fail_count);
}
```

```
}
```

### Echo Server Pattern

```
void WebSocketEchoHandler(int ws_fd) {
 char read_buffer[512];

 while (1) {
 int bytes_read = ws_read(ws_fd, read_buffer, sizeof(read_buffer) - 1);

 if (bytes_read > 0) {
 read_buffer[bytes_read] = '\0';

 // Echo back what was received
 int bytes_written = ws_write(ws_fd, read_buffer, bytes_read);

 if (bytes_written > 0) {
 ws_flush(ws_fd);
 printf("Echoed %d bytes\n", bytes_written);
 } else {
 printf("Echo write failed\n");
 break;
 }
 } else if (bytes_read < 0) {
 printf("Connection closed\n");
 break;
 }

 OSTimeDly(TICKS_PER_SECOND / 100);
 }
}
```

#### 16.190.36.4 ws\_write\_notify()

```
static void NB::WebSocket::ws_write_notify (
 int fd) [static]
```

TCP write notification callback invoked when buffer space becomes available.

### 16.190.37 Description

This static callback function is invoked by the NetBurner TCP/IP stack when the TCP send buffer has space available for writing. It allows the [WebSocket](#) handler to resume sending data that was previously blocked due to a full send buffer. This implements flow control for outbound [WebSocket](#) traffic.

### 16.190.38 Parameters

#### Parameters

|           |                                                                   |
|-----------|-------------------------------------------------------------------|
| <i>fd</i> | The file descriptor of the TCP socket with available buffer space |
|-----------|-------------------------------------------------------------------|

### 16.190.39 Usage Notes

- Automatically called by the network stack when send buffer space is available
- Essential for handling EWOULDBLOCK conditions on non-blocking sockets
- Should attempt to flush pending outbound data
- Must not block or perform lengthy operations
- Typically registered via `SetSocketWriteNotify()` or similar
- May be called multiple times as buffer space becomes available

### 16.190.40 Flow Control

This callback enables proper flow control by:

- Resuming writes after EWOULDBLOCK
- Preventing busy-waiting on socket writability
- Managing outbound message queues efficiently

### 16.190.41 Thread Safety

Called from the network stack context. Ensure thread-safe access to write queues.

### 16.190.42 See Also

- [ws\\_read\\_notify\(\)](#) - Companion read notification callback
- [GetFlushTime\(\)](#) - Related flush timing management
- TCP notification system

---

#### Expand for Example Usage

##### Examples

##### Example 1: Basic Write Notification Setup

```
class WebSocketHandler {
private:
 static void ws_write_notify(int fd) {
 printf("Write space available on fd %d\r\n", fd);

 // Attempt to flush pending data
 if (HasPendingData(fd)) {
 FlushPendingData(fd);
 }
 }

public:
 void RegisterNotifications(int sockfd) {
 // Register write callback with TCP stack
 SetSocketWriteNotify(sockfd, ws_write_notify);
 }
};
```

##### Example 2: Message Queue Flushing

```
class WebSocketConnection {
private:
 static std::map<int, std::queue<WebSocketMessage> writeQueues;

 static void ws_write_notify(int fd) {
 auto& queue = writeQueues[fd];

 // Try to send queued messages
 while (!queue.empty()) {
 WebSocketMessage& msg = queue.front();

 int bytesSent = write(fd, msg.data, msg.length);

 if (bytesSent == msg.length) {
 // Complete message sent
 queue.pop();
 }
 else if (bytesSent > 0) {
 // Partial send - update message
 msg.data += bytesSent;
 msg.length -= bytesSent;
 break; // Wait for next notification
 }
 else {
 // EWOULDBLOCK - buffer full again
 if (errno == EWOULDBLOCK) break;

 // Error - close connection
 printf("Write error on fd=%d: %d\r\n", fd, errno);
 CleanupConnection(fd);
 break;
 }
 }
 }
};
```



```

 // Disable write notifications if queue empty
 if (queue.empty()) {
 SetSocketWriteNotify(fd, nullptr);
 }
 }
};

```

### Example 3: Advanced Flow Control with Backpressure

```

static void ws_write_notify(int fd) {
 ConnectionState* conn = GetConnection(fd);
 if (!conn) return;

 // Update connection state
 conn->writeBlocked = false;

 // Attempt to flush buffer
 while (conn->sendBuffer.HasData()) {
 const uint8_t* data = conn->sendBuffer.GetData();
 size_t length = conn->sendBuffer.GetLength();

 int bytesSent = write(fd, data, length);

 if (bytesSent > 0) {
 conn->sendBuffer.Consume(bytesSent);
 conn->totalBytesSent += bytesSent;

 if (bytesSent < (int)length) {
 // Partial write - buffer full again
 printf("Partial write: %d/%zu bytes\r\n", bytesSent, length);
 conn->writeBlocked = true;
 break;
 }
 }
 else if (errno == EWOULDBLOCK) {
 conn->writeBlocked = true;
 break;
 }
 else {
 // Error
 printf("Write failed on fd=%d: errno=%d\r\n", fd, errno);
 CloseConnection(fd);
 return;
 }
 }

 // Notify application that backpressure is relieved
 if (!conn->writeBlocked && conn->onWriteReady) {
 conn->onWriteReady(fd);
 }
}

```

## 16.190.43 Member Data Documentation

### 16.190.43.1 m\_fd\_tcp

int NB::WebSocket::m\_fd\_tcp

TCP socket file descriptor (underlying transport)

This member variable stores the file descriptor for the underlying TCP socket that provides the transport layer for the [WebSocket](#) connection. The TCP socket handles the actual network communication, while the [WebSocket](#) protocol operates on top of it.

Key characteristics:

- Created during initial connection establishment
- Used for all low-level socket operations (send, recv, setsockopt, etc.)
- Remains valid throughout the [WebSocket](#) connection lifetime
- Closed when the [WebSocket](#) connection is terminated

The TCP socket is typically established through one of two methods:

1. Client-initiated connection: Created via [connect\(\)](#) to a remote server
2. Server-accepted connection: Obtained from [accept\(\)](#) on a listening socket

After the [WebSocket](#) handshake completes, this socket is "promoted" to [WebSocket](#) mode, but the underlying TCP file descriptor remains the same and continues to handle the transport.

Common operations using this file descriptor:

- Reading raw data from the network
- Writing framed [WebSocket](#) data to the network
- Setting socket options (SO\_KEEPALIVE, TCP\_NODELAY, etc.)
- Checking socket status and errors
- Closing the connection

#### Note

This is the actual system socket file descriptor. Operations on this socket must be thread-safe when multiple tasks access the [WebSocket](#).

A value of -1 or similar invalid descriptor indicates no connection.

Direct manipulation of this file descriptor should be avoided in favor of using [WebSocket](#) member functions, which handle protocol requirements.

#### See also

[m\\_fd\\_ws](#) for the application-level [WebSocket](#) identifier

[socket\\_crit](#) for thread-safe access to socket operations

#### Example Usage

---

### Expand for Example Usage

#### Examples

##### Check if Socket is Valid

```
WebSocket *ws = GetActiveWebSocket();

if (ws->m_fd_tcp >= 0) {
 printf("TCP socket is valid (fd: %d)\n", ws->m_fd_tcp);
} else {
 printf("TCP socket is invalid\n");
}
```

##### Set Socket Options on TCP Layer

```
void ConfigureTcpSocket(WebSocket *ws) {
 if (ws->m_fd_tcp < 0) {
 printf("ERROR: Invalid TCP socket\n");
 return;
 }

 // Set TCP_NODELAY for low latency
 int flag = 1;
 if (setsockopt(ws->m_fd_tcp, IPPROTO_TCP, TCP_NODELAY,
 &flag, sizeof(flag)) == 0) {
 printf("TCP_NODELAY set on fd %d\n", ws->m_fd_tcp);
 }

 // Enable keepalive
 flag = 1;
 if (setsockopt(ws->m_fd_tcp, SOL_SOCKET, SO_KEEPALIVE,
 &flag, sizeof(flag)) == 0) {
 printf("SO_KEEPALIVE set on fd %d\n", ws->m_fd_tcp);
 }
}
```

### Get Socket Information

```

void DisplaySocketInfo(WebSocket *ws) {
 if (ws->m_fd_tcp < 0) {
 printf("No TCP socket\n");
 return;
 }

 // Get local address
 IPADDR local_addr;
 WORD local_port;
 GetSocketAddr(ws->m_fd_tcp, &local_addr, &local_port);

 // Get peer address
 IPADDR peer_addr;
 WORD peer_port;
 GetSocketPeer(ws->m_fd_tcp, &peer_addr, &peer_port);

 printf("TCP Socket Info (fd: %d):\n", ws->m_fd_tcp);
 printf(" Local: %s:%d\n", IPAddressToString(local_addr), local_port);
 printf(" Peer: %s:%d\n", IPAddressToString(peer_addr), peer_port);
}

```

### Check Socket Error Status

```

bool CheckSocketError(WebSocket *ws) {
 if (ws->m_fd_tcp < 0) {
 return true; // Invalid socket is an error
 }

 int error = 0;
 socklen_t len = sizeof(error);

 if (getsockopt(ws->m_fd_tcp, SOL_SOCKET, SO_ERROR,
 &error, &len) == 0) {
 if (error != 0) {
 printf("Socket error on fd %d: %d\n", ws->m_fd_tcp, error);
 return true;
 }
 }

 return false;
}

```

### Monitor Multiple WebSocket TCP Sockets

```

void MonitorAllTcpSockets() {
 printf("TCP Socket Status:\n");

 for (int i = 0; i < WS_MAX_SOCKETS; i++) {
 WebSocket *ws = &WebSocket::WSSockStructs[i];

 if (ws->GetState() != WS_STATE_CLOSED && ws->m_fd_tcp >= 0) {
 printf(" [%d] TCP fd=%d, WS fd=%d, State=%d\n",
 i, ws->m_fd_tcp, ws->m_fd_ws, ws->GetState());
 }
 }
}

```

### Get Socket Buffer Sizes

```

void DisplayBufferSizes(WebSocket *ws) {
 if (ws->m_fd_tcp < 0) {
 printf("Invalid socket\n");
 return;
 }

 int sndbuf, rcvbuf;
 socklen_t len = sizeof(int);

 getsockopt(ws->m_fd_tcp, SOL_SOCKET, SO_SNDBUF, &sndbuf, &len);
 getsockopt(ws->m_fd_tcp, SOL_SOCKET, SO_RCVBUF, &rcvbuf, &len);

 printf("TCP Socket %d Buffer Sizes:\n", ws->m_fd_tcp);
 printf(" Send buffer: %d bytes\n", sndbuf);
 printf(" Recv buffer: %d bytes\n", rcvbuf);
}

```

## 16.190.43.2 m\_fd\_ws

int NB::WebSocket::m\_fd\_ws

[WebSocket](#) file descriptor (application-level identifier)

This member variable stores an application-level file descriptor or identifier that represents this [WebSocket](#) connection. While `m_fd_tcp` represents the underlying TCP transport, `m_fd_ws` provides a WebSocket-specific identifier used by the [WebSocket](#) management system.

Key characteristics:

- Assigned when a TCP connection is promoted to a [WebSocket](#)
- Used to identify and look up [WebSocket](#) instances
- May differ from `m_fd_tcp` in systems with separate numbering schemes
- Used as a key in [WebSocket](#) registration/lookup operations

This identifier is used throughout the [WebSocket](#) API for operations such as:

- Finding a [WebSocket](#) instance via [GetWebSocketRecord\(\)](#)
- Static wrapper functions (`ws_read`, `ws_write`, etc.)
- Connection management and tracking
- Diagnostic and monitoring operations

In some implementations, `m_fd_ws` may equal `m_fd_tcp`, indicating the same file descriptor is used for both layers. In other implementations, they may be different to provide abstraction between layers.

#### Note

This identifier is set during [WebSocket](#) promotion (`promote_tcp_ws`).

A value of -1 or similar invalid identifier indicates no [WebSocket](#) has been established.

Application code should primarily use this identifier when calling WebSocket-specific functions rather than accessing `m_fd_tcp` directly.

#### See also

[m\\_fd\\_tcp](#) for the underlying TCP socket file descriptor

[GetWebSocketRecord\(\)](#) which uses this identifier to find [WebSocket](#) instances

#### Example Usage

---

### Expand for Example Usage

#### Examples

##### Get WebSocket Identifier

```
WebSocket *ws = GetActiveWebSocket();

if (ws->m_fd_ws >= 0) {
 printf("WebSocket identifier: %d\n", ws->m_fd_ws);
 printf("TCP socket: %d\n", ws->m_fd_tcp);
} else {
 printf("WebSocket not established\n");
}
```

##### Use WebSocket Identifier with Static Functions

```
void SendMessage(WebSocket *ws, const char *message) {
 int ws_fd = ws->m_fd_ws;

 if (ws_fd < 0) {
 printf("ERROR: Invalid WebSocket identifier\n");
 return;
 }

 // Use static wrapper functions with WebSocket identifier
 int result = ws_write(ws_fd, message, strlen(message));
 if (result > 0) {
 ws_flush(ws_fd);
 printf("Message sent via WebSocket %d\n", ws_fd);
 }
}
```

### Compare TCP and WebSocket Identifiers

```
void DisplayIdentifiers(WebSocket *ws) {
 printf("Connection Identifiers:\n");
 printf(" TCP fd: %d\n", ws->m_fd_tcp);
 printf(" WebSocket fd: %d\n", ws->m_fd_ws);

 if (ws->m_fd_tcp == ws->m_fd_ws) {
 printf(" (Same identifier used for both layers)\n");
 } else {
 printf(" (Separate identifiers for layers)\n");
 }
}
```

### Lookup WebSocket by Identifier

```
WebSocket* FindWebSocket(int ws_fd) {
 for (int i = 0; i < WS_MAX_SOCKETS; i++) {
 WebSocket *ws = &WebSocket::WSSockStructs[i];

 if (ws->m_fd_ws == ws_fd) {
 printf("Found WebSocket at index %d\n", i);
 return ws;
 }
 }

 printf("WebSocket %d not found\n", ws_fd);
 return nullptr;
}
```

### Track Active WebSocket Identifiers

```
void ListActiveWebSockets() {
 printf("Active WebSocket Identifiers:\n");
 int count = 0;

 for (int i = 0; i < WS_MAX_SOCKETS; i++) {
 WebSocket *ws = &WebSocket::WSSockStructs[i];

 if (ws->m_fd_ws >= 0 && ws->GetState() == WS_STATE_OPEN) {
 printf(" [%d] WS fd=%d (TCP fd=%d)\n",
 i, ws->m_fd_ws, ws->m_fd_tcp);
 count++;
 }
 }

 printf("Total active: %d\n", count);
}
```

### Validate WebSocket Identifier

```
bool IsValidWebSocketId(int ws_fd) {
 if (ws_fd < 0) {
 return false;
 }

 // Check if this identifier belongs to an active WebSocket
 for (int i = 0; i < WS_MAX_SOCKETS; i++) {
 WebSocket *ws = &WebSocket::WSSockStructs[i];
 if (ws->m_fd_ws == ws_fd) {
 return true;
 }
 }

 return false;
}
```

## 16.190.43.3 s\_openSocketCount

```
int NB::WebSocket::s_openSocketCount [static]
```

Number of currently open [WebSocket](#) connections.

This static member variable tracks the total number of active [WebSocket](#) connections currently managed by the [WebSocket](#) system. It is incremented when a new [WebSocket](#) connection is established and decremented when a connection is closed.

This counter is useful for:

- Monitoring system resource usage
- Enforcing connection limits

- Diagnostics and debugging
- Load balancing decisions

The value ranges from 0 to `WS_MAX_SOCKETS` (the maximum number of concurrent [WebSocket](#) connections supported by the system).

#### Note

This is a static variable shared across all [WebSocket](#) instances. Access should be synchronized in multi-threaded environments.

#### See also

[WS\\_MAX\\_SOCKETS](#) for the maximum connection limit

[WSSockStruts](#) for the array of [WebSocket](#) structures

#### Example Usage

---

### Expand for Example Usage

#### Examples

##### Check Connection Count

```
void DisplayWebSocketStatus() {
 int active_connections = WebSocket::s_openSocketCount;
 printf("Active WebSocket connections: %d/%d\n",
 active_connections, WS_MAX_SOCKETS);

 if (active_connections >= WS_MAX_SOCKETS) {
 printf("WARNING: Maximum WebSocket connections reached!\n");
 }
}
```

##### Enforce Connection Limit

```
bool CanAcceptNewConnection() {
 if (WebSocket::s_openSocketCount >= WS_MAX_SOCKETS) {
 printf("Cannot accept new connection: limit reached (%d)\n",
 WS_MAX_SOCKETS);
 return false;
 }

 printf("Can accept connection (%d/%d slots used)\n",
 WebSocket::s_openSocketCount, WS_MAX_SOCKETS);
 return true;
}
```

##### Resource Monitoring

```
void MonitorWebSocketResources() {
 int used = WebSocket::s_openSocketCount;
 int available = WS_MAX_SOCKETS - used;
 float usage_percent = (used * 100.0f) / WS_MAX_SOCKETS;

 printf("WebSocket Resource Usage:\n");
 printf(" Used: %d\n", used);
 printf(" Available: %d\n", available);
 printf(" Usage: %.1f%%\n", usage_percent);

 if (usage_percent > 80.0f) {
 printf(" WARNING: High resource usage!\n");
 }
}
```

##### Connection Acceptance Logic

```
int HandleNewWebSocketRequest(int tcp_fd) {
 // Check if we can accept more connections
 if (WebSocket::s_openSocketCount >= (WS_MAX_SOCKETS - 1)) {
 printf("Rejecting connection: only 1 slot remaining (reserved)\n");
 SendWebSocketError(tcp_fd, 503, "Service Unavailable");
 close(tcp_fd);
 return -1;
 }

 // Promote to WebSocket
```

```

 if (promote_tcp_ws(tcp_fd) == 0) {
 printf("Connection accepted (%d/%d)\n",
 WebSocket::s_openSocketCount, WS_MAX_SOCKS);
 return 0;
 }

 return -1;
}

```

### Periodic Status Reporting

```

void WebSocketStatusTask(void *pd) {
 while (1) {
 int count = WebSocket::s_openSocketCount;

 if (count > 0) {
 printf("[%lu] Active WebSocket connections: %d\n",
 Secs, count);
 }

 // Report every 60 seconds
 OSTimeDly(TICKS_PER_SECOND * 60);
 }
}

```

#### 16.190.43.4 socket\_crit

OS\_CRIT NB::WebSocket::socket\_crit

Critical section for thread-safe socket operations.

This member variable provides a critical section (mutual exclusion) mechanism to ensure thread-safe access to the [WebSocket](#)'s socket operations. In a multi-threaded RTOS environment, multiple tasks may attempt to read from, write to, or modify the same [WebSocket](#) connection simultaneously, which could lead to data corruption or protocol violations.

The critical section protects:

- Socket read and write operations
- Buffer access and manipulation
- State changes and transitions
- Socket option modifications
- Connection establishment and closure

Usage pattern:

1. Enter critical section before socket operations
2. Perform socket operations
3. Exit critical section after operations complete

NetBurner RTOS provides OS\_CRIT primitives for critical sections:

- USER\_ENTER\_CRITICAL(): Enter critical section (acquire lock)
- USER\_EXIT\_CRITICAL(): Exit critical section (release lock)

These operations disable task switching temporarily to ensure atomic execution of protected code sections. The critical section should be held for the minimum time necessary to avoid blocking other tasks.

#### Note

Critical sections should be kept as short as possible to minimize impact on system responsiveness.

Failing to properly enter/exit critical sections can lead to race conditions, data corruption, or deadlocks.

The OS\_CRIT type is typically a structure that tracks the critical section state and may include nesting counters.

## Warning

Never return from a function without exiting a critical section you entered. Always use balanced enter/exit calls.

Avoid performing blocking operations (delays, waits) inside critical sections as this defeats their purpose and can cause system performance issues.

## See also

[m\\_fd\\_tcp](#) for the socket being protected

[m\\_fd\\_ws](#) for the [WebSocket](#) identifier

## Example Usage

---

## Expand for Example Usage

### Examples

#### Protected Write Operation

```
int SafeWrite(WebSocket *ws, const char *data, int length) {
 USER_ENTER_CRITICAL(&ws->socket_crit);

 // Protected: Write operation is atomic
 int bytes_written = write(ws->m_fd_tcp, data, length);

 USER_EXIT_CRITICAL(&ws->socket_crit);

 return bytes_written;
}
```

#### Protected Read Operation

```
int SafeRead(WebSocket *ws, char *buffer, int size) {
 USER_ENTER_CRITICAL(&ws->socket_crit);

 // Protected: Read operation is atomic
 int bytes_read = read(ws->m_fd_tcp, buffer, size);

 USER_EXIT_CRITICAL(&ws->socket_crit);

 return bytes_read;
}
```

#### Protected State Change

```
void SetConnectionState(WebSocket *ws, WS_State new_state) {
 USER_ENTER_CRITICAL(&ws->socket_crit);

 // Protected: State change is atomic
 WS_State old_state = ws->m_state;
 ws->m_state = new_state;

 USER_EXIT_CRITICAL(&ws->socket_crit);

 printf("State changed: %d -> %d\n", old_state, new_state);
}
```

#### Protected Buffer Access

```
int GetBufferStats(WebSocket *ws, int *bytes_available, int *space_free) {
 USER_ENTER_CRITICAL(&ws->socket_crit);

 // Protected: Buffer state reads are consistent
 *bytes_available = ws->rxBuffer.BytesAvail();
 *space_free = ws->txBuffer.SpaceAvail();

 USER_EXIT_CRITICAL(&ws->socket_crit);

 return 0;
}
```



### Error Handling in Critical Section

```
int SafeOperation(WebSocket *ws, const char *data, int length) {
 USER_ENTER_CRITICAL(&ws->socket_crit);

 int result = -1;

 if (ws->m_fd_tcp >= 0) {
 result = write(ws->m_fd_tcp, data, length);

 if (result < 0) {
 // Handle error inside critical section
 ws->m_state = WS_STATE_ERROR;
 }
 }

 USER_EXIT_CRITICAL(&ws->socket_crit);

 return result;
}
```

### Nested Critical Section (if supported)

```
void ComplexOperation(WebSocket *ws) {
 USER_ENTER_CRITICAL(&ws->socket_crit);

 // First operation
 UpdateInternalState(ws);

 // Nested critical section (if OS_CRIT supports nesting)
 USER_ENTER_CRITICAL(&ws->socket_crit);
 UpdateSocketOptions(ws);
 USER_EXIT_CRITICAL(&ws->socket_crit);

 // Second operation
 FlushBuffers(ws);

 USER_EXIT_CRITICAL(&ws->socket_crit);
}
```

### Multiple Task Access Pattern

```
// Task 1: Writing to WebSocket
void WriterTask(void *pd) {
 WebSocket *ws = (WebSocket *)pd;

 while (1) {
 char data[64];
 GenerateData(data, sizeof(data));

 USER_ENTER_CRITICAL(&ws->socket_crit);
 ws->WriteData(data, sizeof(data));
 USER_EXIT_CRITICAL(&ws->socket_crit);

 OSTimeDly(TICKS_PER_SECOND);
 }
}

// Task 2: Reading from WebSocket
void ReaderTask(void *pd) {
 WebSocket *ws = (WebSocket *)pd;

 while (1) {
 char buffer[256];

 USER_ENTER_CRITICAL(&ws->socket_crit);
 int bytes = ws->ReadData(buffer, sizeof(buffer));
 USER_EXIT_CRITICAL(&ws->socket_crit);

 if (bytes > 0) {
 ProcessData(buffer, bytes);
 }

 OSTimeDly(TICKS_PER_SECOND / 10);
 }
}
```

### Critical Section Timing Measurement

```
void MeasureCriticalSectionTime(WebSocket *ws) {
 uint32_t start_tick = TimeTick;

 USER_ENTER_CRITICAL(&ws->socket_crit);

 // Perform operation
```

```

PerformSocketOperation(ws);

uint32_t end_tick = TimeTick;
USER_EXIT_CRITICAL(&ws->socket_crit);

uint32_t duration_ticks = end_tick - start_tick;
float duration_us = (duration_ticks * 1000000.0f) / TICKS_PER_SECOND;

printf("Critical section held for %.2f microseconds\n", duration_us);

if (duration_us > 1000.0f) {
 printf("WARNING: Critical section held too long!\n");
}
}

```

### Conditional Critical Section

```

void ConditionalProtectedOperation(WebSocket *ws, bool needs_protection) {
 if (needs_protection) {
 USER_ENTER_CRITICAL(&ws->socket_crit);
 }

 // Operation that may or may not need protection
 PerformOperation(ws);

 if (needs_protection) {
 USER_EXIT_CRITICAL(&ws->socket_crit);
 }
}

```

### Diagnostic: Check Critical Section Status

```

void DiagnoseCriticalSection(WebSocket *ws) {
 // Note: This is conceptual - actual implementation depends on OS_CRIT structure
 printf("Critical Section Diagnostics:\n");
 printf(" Address: %p\n", &ws->socket_crit);

 // If OS_CRIT has a lock counter or owner field:
 // printf(" Lock count: %d\n", ws->socket_crit.lock_count);
 // printf(" Owner task: %d\n", ws->socket_crit.owner_task);
}

```

## 16.190.43.5 WSSockStructs

`WebSocket` NB: `WebSocket::WSSockStructs[WS_MAX_SOCKS]` [static]

Static array of `WebSocket` connection structures.

This static array holds all `WebSocket` connection structures, providing storage for up to `WS_MAX_SOCKS` concurrent `WebSocket` connections. Each element represents a potential `WebSocket` connection slot.

The array is pre-allocated at compile time, eliminating the need for dynamic memory allocation during runtime. This approach:

- Provides predictable memory usage
- Eliminates allocation failures
- Improves performance by avoiding heap operations
- Simplifies resource management

When a new `WebSocket` connection is established, an unused slot in this array is assigned. When a connection closes, its slot becomes available for reuse.

#### Note

This is a static array shared across all `WebSocket` operations. Individual elements are accessed via helper functions like `GetWebSocketRecord()`.

#### See also

`WS_MAX_SOCKS` for the array size

`s_openSocketCount` for tracking active connections

`GetWebSocketRecord()` for finding `WebSocket` by file descriptor

## Example Usage

### Expand for Example Usage

#### Examples Iterate All WebSocket Slots

```
void DisplayAllWebSockets() {
 printf("WebSocket Connection List:\n");
 printf("=====\n");

 for (int i = 0; i < WS_MAX_SOCKETS; i++) {
 WebSocket *ws = &WebSocket::WSSockStructs[i];

 if (ws->IsActive()) {
 printf("[%d] fd=%d, state=%d\n",
 i, ws->GetFd(), ws->GetState());
 }
 }
}
```

#### Find Available Slot

```
WebSocket* FindAvailableWebSocketSlot() {
 for (int i = 0; i < WS_MAX_SOCKETS; i++) {
 WebSocket *ws = &WebSocket::WSSockStructs[i];

 if (!ws->IsActive()) {
 printf("Found available slot: %d\n", i);
 return ws;
 }
 }

 printf("No available WebSocket slots\n");
 return nullptr;
}
```

#### Close All Active Connections

```
void CloseAllWebSockets() {
 int closed_count = 0;

 printf("Closing all WebSocket connections...\n");

 for (int i = 0; i < WS_MAX_SOCKETS; i++) {
 WebSocket *ws = &WebSocket::WSSockStructs[i];

 if (ws->IsActive()) {
 int fd = ws->GetFd();
 ws_externalclose(fd);
 closed_count++;
 printf(" Closed slot %d (fd %d)\n", i, fd);
 }
 }

 printf("Closed %d WebSocket connections\n", closed_count);
}
```

#### Collect Connection Statistics

```
void CollectWebSocketStatistics() {
 int active = 0;
 int inactive = 0;
 uint32_t total_bytes_sent = 0;
 uint32_t total_bytes_received = 0;

 for (int i = 0; i < WS_MAX_SOCKETS; i++) {
 WebSocket *ws = &WebSocket::WSSockStructs[i];

 if (ws->IsActive()) {
 active++;
 total_bytes_sent += ws->GetBytesSent();
 total_bytes_received += ws->GetBytesReceived();
 } else {
 inactive++;
 }
 }

 printf("WebSocket Statistics:\n");
 printf(" Active connections: %d\n", active);
 printf(" Inactive slots: %d\n", inactive);
 printf(" Total bytes sent: %lu\n", total_bytes_sent);
 printf(" Total bytes received: %lu\n", total_bytes_received);
}
```

### Broadcast to All Active Connections

```
void BroadcastToAllWebSockets(const char *message) {
 int length = strlen(message);
 int sent_count = 0;

 for (int i = 0; i < WS_MAX_SOCKETS; i++) {
 WebSocket *ws = &WebSocket::WSocketStructs[i];

 if (ws->IsActive()) {
 int fd = ws->GetFd();
 int bytes_written = ws_write(fd, message, length);

 if (bytes_written > 0) {
 ws_flush(fd);
 sent_count++;
 }
 }
 }

 printf("Broadcast sent to %d WebSocket(s)\n", sent_count);
}
```

### Find Idle Connections

```
void CloseIdleConnections(uint32_t max_idle_seconds) {
 uint32_t current_time = Secs;

 for (int i = 0; i < WS_MAX_SOCKETS; i++) {
 WebSocket *ws = &WebSocket::WSocketStructs[i];

 if (ws->IsActive()) {
 uint32_t idle_time = current_time - ws->GetLastActivityTime();

 if (idle_time > max_idle_seconds) {
 int fd = ws->GetFd();
 printf("Closing idle connection slot %d (idle %lu sec)\n",
 i, idle_time);
 ws_externalclose(fd);
 }
 }
 }
}
```

---

The documentation for this class was generated from the following file:

- websockets.h

## 16.191 wifi\_init Struct Reference

```
#include <wifiDriver.h>
```

### 16.191.1 Detailed Description

This structure is used as a WiFi initialization structure for defining pins, IRQ, and type of WiFi module being used with a specific bus type. the GetDriver function pointer must be defined to one of the supported WiFi module option's static driver function. The options are GetNewWicedSPIDriver(), GetNewWicedSerialDriver(), or GetNewWicedWilcDriver(). Default WifiInit structures are defined using macros for each of the NBWIFI development kit options. If the user is using pin/IRQ settings that differ from the default values, the user must either define their own wifiInit structure in their project or pass in the custom values to the WiFi initialization functions as parameters.

The documentation for this struct was generated from the following file:

- wifiDriver.h

## 16.192 WifiInterface Class Reference

WiFi Interface clas.

```
#include <nbWifiApi.h>
```

### 16.192.1 Detailed Description

WiFi Interface clas.

The documentation for this class was generated from the following file:

- nbWifiApi.h

## 16.193 WireIntf Class Reference

Wire Interface Class for [I2C](#).

```
#include <i2c.h>
```

### Public Member Functions

- void **begin** ()  
*Initialize the [WireIntf](#) driver object as a master with a bus speed of 100 kHz. This normally only needs to be called once.*
- uint32\_t **requestFrom** (uint8\_t addr, uint32\_t len, bool stop=true)  
*Request a number of bytes from a slave device. The bytes can then be retrieved from the slave device with the [read\(\)](#) and [available\(\)](#) functions.*
- void **beginTransmission** (uint8\_t addr)  
*Begin a transmission to a [I2C](#) slave device at the provided address. Bytes can be queued for transmission with the [write\(\)](#) functions, and are transmitted by calling [endTransmission\(\)](#).*
- void **endTransmission** (bool stop=true)  
*Transmits the bytes of data queued by using the [write\(\)](#) functions, and ends a transmission to a slave device that was started using [beginTransmission\(\)](#).*
- uint32\_t **write** (uint8\_t dat)  
*Queues bytes of data to be transmitted to the slave device. This function can be called after a call to [beginTransmission\(\)](#). The data will not be transmitted on the [I2C](#) bus until a call to [endTransmission\(\)](#) or [flush\(\)](#) is performed.*
- uint32\_t **write** (char \*str)  
*Queues bytes of data to be transmitted to the slave device. This function can be called after a call to [beginTransmission\(\)](#). The data will not be transmitted on the [I2C](#) bus until a call to [endTransmission\(\)](#) or [flush\(\)](#) is performed.*
- uint32\_t **write** (uint8\_t \*buf, uint32\_t len)  
*Queues bytes of data to be transmitted to the slave device. This function can be called after a call to [beginTransmission\(\)](#). The data will not be transmitted on the [I2C](#) bus until a call to [endTransmission\(\)](#) or [flush\(\)](#) is performed.*
- uint32\_t **available** ()  
*Get the number of bytes available to be read from the slave device with [read\(\)](#). This function can be called after a call to [requestFrom\(\)](#).*
- uint8\_t **read** ()  
*Reads a byte of data that was transmitted from a slave [I2C](#) device after a call to [requestFrom\(\)](#).*
- void **flush** (bool bIssueStop=false)  
*Force the data that were queued to be transmitted using the [write\(\)](#) functions to be transmitted on the [I2C](#) bus.*

### 16.193.1 Detailed Description

Wire Interface Class for [I2C](#).

For [I2C](#) communication, you can choose the [WireIntf](#) class, or the [I2C](#) class. The [WireIntf](#) class is simpler to use, while the [I2C](#) class provides more low level control.

Note that the system automatically instantiates [I2C](#) and [WireIntf](#) objects for each of the [I2C](#) peripheral modules. The [WireIntf](#) objects can be accessed with: Wire, Wire1 and Wire2. The [I2C](#) objects can be accessed with [I2C\[0\]](#), [I2C\[1\]](#) and [I2C\[2\]](#).

### 16.193.2 Member Function Documentation

#### 16.193.2.1 available()

```
uint32_t WireIntf::available ()
```

Get the number of bytes available to be read from the slave device with [read\(\)](#). This function can be called after a call to [requestFrom\(\)](#).

**Returns**

The number of bytes available for reading from the slave device.

**16.193.2.2 beginTransmission()**

```
void WireIntf::beginTransmission (
 uint8_t addr)
```

Begin a transmission to a [I2C](#) slave device at the provided address. Bytes can be queued for transmission with the [write\(\)](#) functions, and are transmitted by calling [endTransmission\(\)](#).

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| <i>addr</i> | the address of the device to transmit data to. |
|-------------|------------------------------------------------|

**16.193.2.3 endTransmission()**

```
void WireIntf::endTransmission (
 bool stop = true)
```

Transmits the bytes of data queued by using the [write\(\)](#) functions, and ends a transmission to a slave device that was started using [beginTransmission\(\)](#).

**Parameters**

|             |                                                                                                                                                                                                                                                                                                                                             |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>stop</i> | If true, sends a stop message after the request, releasing the <a href="#">I2C</a> bus. If false, the driver will send a restart request, keeping the connection alive. This prevents other <a href="#">I2C</a> master devices from transmitting between messages. This allows one master to transmit multiple transmissions uninterrupted. |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**16.193.2.4 flush()**

```
void WireIntf::flush (
 bool bIssueStop = false)
```

Force the data that were queued to be transmitted using the [write\(\)](#) functions to be transmitted on the [I2C](#) bus. If you need to know exactly when a transaction will occur, call [flush\(\)](#). If accessing a register, be sure to write all address bytes before the call.

**Parameters**

|                   |                                                                                                                                                                                                                                                                                                                                             |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>bIssueStop</i> | If true, sends a stop message after the request, releasing the <a href="#">I2C</a> bus. If false, the driver will send a restart request, keeping the connection alive. This prevents other <a href="#">I2C</a> master devices from transmitting between messages. This allows one master to transmit multiple transmissions uninterrupted. |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**16.193.2.5 read()**

```
uint8_t WireIntf::read ()
```

Reads a byte of data that was transmitted from a slave [I2C](#) device after a call to [requestFrom\(\)](#).

**Returns**

The byte of data received from the slave [I2C](#) device.

### 16.193.2.6 requestFrom()

```
uint32_t WireIntf::requestFrom (
 uint8_t addr,
 uint32_t len,
 bool stop = true)
```

Request a number of bytes from a slave device. The bytes can then be retrieved from the slave device with the [read\(\)](#) and [available\(\)](#) functions.

#### Parameters

|             |                                                                                                                                                                                                                                                                                                                                             |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>addr</i> | the address of the slave to request data from.                                                                                                                                                                                                                                                                                              |
| <i>len</i>  | the number of bytes to request from the slave.                                                                                                                                                                                                                                                                                              |
| <i>stop</i> | If true, sends a stop message after the request, releasing the <a href="#">I2C</a> bus. If false, the driver will send a restart request, keeping the connection alive. This prevents other <a href="#">I2C</a> master devices from transmitting between messages. This allows one master to transmit multiple transmissions uninterrupted. |

#### Returns

The number of bytes returned from the slave [I2C](#) device.

### 16.193.2.7 write() [1/3]

```
uint32_t WireIntf::write (
 char * str)
```

Queues bytes of data to be transmitted to the slave device. This function can be called after a call to [beginTransaction\(\)](#). The data will not be transmitted on the [I2C](#) bus until a call to [endTransmission\(\)](#) or [flush\(\)](#) is performed.

#### Parameters

|            |                                                                                |
|------------|--------------------------------------------------------------------------------|
| <i>str</i> | A string to send as a series of bytes to the <a href="#">I2C</a> slave device. |
|------------|--------------------------------------------------------------------------------|

#### Returns

The number of of bytes written to the transmit queue buffer.

### 16.193.2.8 write() [2/3]

```
uint32_t WireIntf::write (
 uint8_t * buf,
 uint32_t len)
```

Queues bytes of data to be transmitted to the slave device. This function can be called after a call to [beginTransaction\(\)](#). The data will not be transmitted on the [I2C](#) bus until a call to [endTransmission\(\)](#) or [flush\(\)](#) is performed.

#### Parameters

|            |                                                                            |
|------------|----------------------------------------------------------------------------|
| <i>buf</i> | An array of data to send as bytes to the <a href="#">I2C</a> slave device. |
| <i>len</i> | The number of bytes to be written from the buf parameter.                  |

#### Returns

The number of of bytes written to the transmit queue buffer.

**16.193.2.9 write() [3/3]**

```
uint32_t WireIntf::write (
 uint8_t dat)
```

Queues bytes of data to be transmitted to the slave device. This function can be called after a call to [beginTransaction\(\)](#). The data will not be transmitted on the [I2C](#) bus until a call to [endTransmission\(\)](#) or [flush\(\)](#) is performed.

**Parameters**

|            |                                                                           |
|------------|---------------------------------------------------------------------------|
| <i>dat</i> | A byte of data to be transmitted to the <a href="#">I2C</a> slave device. |
|------------|---------------------------------------------------------------------------|

**Returns**

The number of of bytes written to the transmit queue buffer.

The documentation for this class was generated from the following file:

- [i2c.h](#)

**16.194 WM8904 Class Reference**

```
#include <wm8904.h>
```

**Public Member Functions**

- [WM8904](#) ([I2C](#) &module)  
*The constructor for the [WM8904](#) context.*
- void [Init](#) (const [cfg\\_t](#) &cfg, const [SSC\\_cfg\\_t](#) &ssc\_cfg)  
*Configures and initializes both the driver and the codec.*
- void [Shutdown](#) ()  
*Shuts down the [WM8904](#) codec driver.*
- void [WriteReg](#) ([Reg::RegAddr\\_t](#) reg, [uint16\\_t](#) dat)  
*Write a register on the [WM8904](#).*
- [uint16\\_t](#) [ReadReg](#) ([Reg::RegAddr\\_t](#) reg)  
*Read a register on the [WM8904](#).*
- void [SendCmd](#) ([Reg::cmd\\_t](#) cmd)  
*Execute codec Command. A command is: a register to write, a value to write, and a delay of time required for command execution.*
- void [SendCmdList](#) ([Reg::cmd\\_t](#) \*cmds, [uint32\\_t](#) len)  
*Execute an array of codec Commands. A command is: a register to write, a value to write, and a delay of time required for command execution.*
- void [UpdateCmd](#) ([Reg::cmd\\_t](#) cmd, [uint16\\_t](#) updateMask)  
*Execute a codec Command to update a register. A command is: a register to write, a value to write, and a delay of time required for command execution.*
- void [SetVolume](#) ([AudioOutSelect\\_t](#) out, [AudioChSelect\\_t](#) channel, [uint8\\_t](#) volume)  
*Set the volume of the selected audio output and channel.*
- [uint8\\_t](#) [GetVolume](#) ([AudioOutSelect\\_t](#) out, [AudioChSelect\\_t](#) channel)  
*Get the volume of the selected audio output and channel.*
- void [Mute](#) ([AudioOutSelect\\_t](#) out, [AudioChSelect\\_t](#) channel, bool mute)  
*Mute or unmute the selected audio output and channel.*
- void [SetMicGain](#) ([AudioChSelect\\_t](#) channel, [uint8\\_t](#) gain)  
*Set the microphone gain of the selected input channel.*
- [uint8\\_t](#) [GetMicGain](#) ([AudioChSelect\\_t](#) channel)  
*Get the microphone gain of the selected input channel.*



- void [MuteMic](#) (AudioChSelect\_t channel, bool mute)  
*Mute or unmute the selected input channel.*
- int [TransmitBuffer](#) (void \*buffer, uint32\_t bufferLen, bool waitIfNeeded)  
*Hands off a buffer to be transmitted to the codec.*
- int [ReadyReceiveBuffer](#) (void \*buffer, uint32\_t bufferLen, bool waitIfNeeded)  
*Hands off a buffer to be written to by the codec.*
- void **RegisterTxBufferDoneCB** (SSC\_BufferDoneFn\_t cb)  
*Registers a callback for when a transmit buffer is finished.*
- void **RegisterRxBufferDoneCB** (SSC\_BufferDoneFn\_t cb)  
*Registers a callback for when a receive buffer is finished.*

### 16.194.1 Detailed Description

[WM8904](#) is a driver for the [WM8904](#) Audio Codec.

### 16.194.2 Constructor & Destructor Documentation

#### 16.194.2.1 WM8904()

```
WM8904::WM8904 (
 I2C & module)
```

The constructor for the [WM8904](#) context.

##### Parameters

|               |                                                                                     |
|---------------|-------------------------------------------------------------------------------------|
| <i>module</i> | A reference to the <a href="#">I2C</a> module to be used for configuring the codec. |
|---------------|-------------------------------------------------------------------------------------|

### 16.194.3 Member Function Documentation

#### 16.194.3.1 GetMicGain()

```
uint8_t WM8904::GetMicGain (
 AudioChSelect_t channel)
```

Get the microphone gain of the selected input channel.

##### Parameters

|                |                          |
|----------------|--------------------------|
| <i>channel</i> | Input Channel to select. |
|----------------|--------------------------|

##### Returns

Gain the channel is set to.

#### 16.194.3.2 GetVolume()

```
uint8_t WM8904::GetVolume (
 AudioOutSelect_t out,
 AudioChSelect_t channel)
```

Get the volume of the selected audio output and channel.

##### Parameters

|                |                           |
|----------------|---------------------------|
| <i>out</i>     | Audio Output to select.   |
| <i>channel</i> | Output Channel to select. |

**Returns**

Volume the channel is set to.

**16.194.3.3 Init()**

```
void WM8904::Init (
 const cfg_t & cfg,
 const SSC_cfg_t & ssc_cfg)
```

Configures and initializes both the driver and the codec.

**Parameters**

|                |                                      |
|----------------|--------------------------------------|
| <i>cfg</i>     | The driver configuration to use.     |
| <i>ssc_cfg</i> | The SSC driver configuration to use. |

**16.194.3.4 Mute()**

```
void WM8904::Mute (
 AudioOutSelect_t out,
 AudioChSelect_t channel,
 bool mute)
```

Mute or unmute the selected audio output and channel.

**Parameters**

|                |                                     |
|----------------|-------------------------------------|
| <i>out</i>     | Audio Output to select.             |
| <i>channel</i> | Output Channel to select.           |
| <i>mute</i>    | Whether the channel is to be muted. |

**16.194.3.5 MuteMic()**

```
void WM8904::MuteMic (
 AudioChSelect_t channel,
 bool mute)
```

Mute or unmute the selected input channel.

**Parameters**

|                |                                     |
|----------------|-------------------------------------|
| <i>channel</i> | Input Channel to select.            |
| <i>mute</i>    | Whether the channel is to be muted. |

**16.194.3.6 ReadReg()**

```
uint16_t WM8904::ReadReg (
 Reg::RegAddr_t reg)
```

Read a register on the [WM8904](#).

**Parameters**

|            |                    |
|------------|--------------------|
| <i>reg</i> | Register to write. |
|------------|--------------------|

**Returns**

Data read.

**16.194.3.7 ReadyReceiveBuffer()**

```
int WM8904::ReadyReceiveBuffer (
 void * buffer,
 uint32_t bufferLen,
 bool waitIfNeeded)
```

Hands off a buffer to be written to by the codec.

**Parameters**

|                     |                                                                                                             |
|---------------------|-------------------------------------------------------------------------------------------------------------|
| <i>buffer</i>       | A pointer to the buffer to be written to.                                                                   |
| <i>bufferLen</i>    | The length of the buffer to be written. (Must be multiples of 1, 2, or 4 bytes depending on word bit width) |
| <i>waitIfNeeded</i> | Whether the driver should wait for space to receive or fail immediately upon exhausting the queue depth.    |

**Returns**

Negative on failure.

**16.194.3.8 SendCmd()**

```
void WM8904::SendCmd (
 Reg::cmd_t cmd)
```

Execute codec Command. A command is: a register to write, a value to write, and a delay of time required for command execution.

**Parameters**

|            |                    |
|------------|--------------------|
| <i>cmd</i> | Command to execute |
|------------|--------------------|

**16.194.3.9 SendCmdList()**

```
void WM8904::SendCmdList (
 Reg::cmd_t * cmds,
 uint32_t len)
```

Execute an array of codec Commands. A command is: a register to write, a value to write, and a delay of time required for command execution.

**Parameters**

|             |                                |
|-------------|--------------------------------|
| <i>cmds</i> | Commands to execute            |
| <i>len</i>  | Number of commands to execute. |

**16.194.3.10 SetMicGain()**

```
void WM8904::SetMicGain (
 WM8904::AudioChSelect_t channel,
 uint8_t gain)
```

Set the microphone gain of the selected input channel.

**Parameters**

|                |                          |
|----------------|--------------------------|
| <i>channel</i> | Input Channel to select. |
| <i>gain</i>    | Gain to set channel to.  |

#### 16.194.3.11 SetVolume()

```
void WM8904::SetVolume (
 WM8904::AudioOutSelect_t out,
 WM8904::AudioChSelect_t channel,
 uint8_t volume)
```

Set the volume of the selected audio output and channel.

##### Parameters

|                |                           |
|----------------|---------------------------|
| <i>out</i>     | Audio Output to select.   |
| <i>channel</i> | Output Channel to select. |
| <i>volume</i>  | Volume to set channel to. |

#### 16.194.3.12 TransmitBuffer()

```
int WM8904::TransmitBuffer (
 void * buffer,
 uint32_t bufferLen,
 bool waitIfNeeded)
```

Hands off a buffer to be transmitted to the codec.

##### Parameters

|                     |                                                                                                              |
|---------------------|--------------------------------------------------------------------------------------------------------------|
| <i>buffer</i>       | A pointer to the buffer to be transmit.                                                                      |
| <i>bufferLen</i>    | The length of the buffer to be transmit. (Must be multiples of 1, 2, or 4 bytes depending on word bit width) |
| <i>waitIfNeeded</i> | Whether the driver should wait for space to transmit or fail immediately upon exhausting the queue depth.    |

##### Returns

Negative on failure.

#### 16.194.3.13 UpdateCmd()

```
void WM8904::UpdateCmd (
 Reg::cmd_t cmd,
 uint16_t updateMask)
```

Execute a codec Command to update a register. A command is: a register to write, a value to write, and a delay of time required for command execution.

##### Parameters

|                   |                                                              |
|-------------------|--------------------------------------------------------------|
| <i>cmd</i>        | Command to execute                                           |
| <i>updateMask</i> | A Positive bit mask of bits to update in the target register |

#### 16.194.3.14 WriteReg()

```
void WM8904::WriteReg (
 Reg::RegAddr_t reg,
 uint16_t dat)
```

Write a register on the [WM8904](#).

**Parameters**

|            |                    |
|------------|--------------------|
| <i>reg</i> | Register to write. |
| <i>dat</i> | Data to write.     |

The documentation for this class was generated from the following files:

- wm8904.h
- wm8904.cpp

## 16.195 fifo\_buffer\_storage::WriteBufPtr Class Reference

RAII accessor for writing to buffer storage.

```
#include <buffers.h>
```

### 16.195.1 Detailed Description

RAII accessor for writing to buffer storage.

Provides safe access to internal buffer write pointer with automatic locking and cleanup. Not IRQ safe.

The documentation for this class was generated from the following file:

- buffers.h



# Index

- [\\_DeviceInterruptTrigger](#)
  - [Device Interrupts, 1186](#)
- [\\_EthernetFrameHeader, 1911](#)
- [\\_FlexSPIConfig, 1919](#)
  - [busyBitPolarity, 1920](#)
  - [columnAddressWidth, 1920](#)
  - [controllerMiscOption, 1920](#)
  - [deviceModeArg, 1921](#)
  - [deviceModeCfgEnable, 1921](#)
  - [deviceModeSeq, 1921](#)
  - [deviceModeType, 1921](#)
  - [deviceType, 1921](#)
  - [lookupTable, 1921](#)
  - [lutCustomSeqEnable, 1921](#)
  - [reserved3, 1921](#)
  - [serialClkFreq, 1921](#)
  - [waitTimeCfgCommands, 1921](#)
- [\\_IpHeaderIPv4, 1922](#)
- [\\_PinVector, 1937](#)
  - [config, 1937](#)
  - [operator uint32\\_t, 1938](#)
  - [operator=, 1938](#)
  - [operator\[\], 1938](#)
- [\\_\\_pad0\\_\\_](#)
  - [\\_flexcan\\_frame, 1913](#)
- [\\_edma\\_channel\\_Preemption\\_config, 1905](#)
  - [channelPriority, 1905](#)
  - [enableChannelPreemption, 1905](#)
  - [enablePreemptAbility, 1905](#)
- [\\_edma\\_config, 1905](#)
  - [enableContinuousLinkMode, 1906](#)
  - [enableDebugMode, 1906](#)
  - [enableHaltOnError, 1906](#)
  - [enableRoundRobinArbitration, 1906](#)
- [\\_edma\\_handle, 1906](#)
  - [base, 1907](#)
  - [callback, 1907](#)
  - [channel, 1907](#)
  - [flags, 1907](#)
  - [header, 1907](#)
  - [tail, 1907](#)
  - [tcdPool, 1907](#)
  - [tcdSize, 1907](#)
  - [tcdUsed, 1907](#)
  - [userData, 1907](#)
- [\\_edma\\_minor\\_offset\\_config, 1907](#)
  - [enableDestMinorOffset, 1908](#)
  - [enableSrcMinorOffset, 1908](#)
  - [minorOffset, 1908](#)
- [\\_edma\\_tcd, 1908](#)
  - [ATTR, 1909](#)
  - [BITER, 1909](#)
  - [CITER, 1909](#)
  - [CSR, 1909](#)
  - [DADDR, 1909](#)
  - [DLAST\\_SGA, 1909](#)
  - [DOFF, 1909](#)
  - [NBYTES, 1909](#)
  - [SADDR, 1909](#)
  - [SLAST, 1909](#)
  - [SOFF, 1909](#)
- [\\_edma\\_transfer\\_config, 1910](#)
  - [destAddr, 1910](#)
  - [destOffset, 1910](#)
  - [destTransferSize, 1910](#)
  - [majorLoopCounts, 1910](#)
  - [minorLoopBytes, 1910](#)
  - [srcAddr, 1910](#)
  - [srcOffset, 1910](#)
  - [srcTransferSize, 1910](#)
- [\\_exit](#)
  - [Utilities, 1782](#)
- [\\_fini](#)
  - [Utilities, 1782](#)
- [\\_flexcan\\_clock\\_source](#)
  - [FlexCAN SOMRT1061, 1138](#)
- [\\_flexcan\\_config, 1911](#)
  - [baudRate, 1912](#)
  - [bitRate, 1912](#)
  - [clkSrc, 1912](#)
  - [disableSelfReception, 1912](#)
  - [enableIndividMask, 1912](#)
  - [enableListenOnlyMode, 1912](#)
  - [enableLoopBack, 1912](#)
  - [enableSelfWakeup, 1912](#)
  - [enableSupervisorMode, 1912](#)
  - [enableTimerSync, 1912](#)
  - [maxMbNum, 1912](#)
  - [wakeupSrc, 1912](#)
- [\\_flexcan\\_error\\_flags](#)
  - [FlexCAN SOMRT1061, 1139](#)
- [\\_flexcan\\_fifo\\_transfer, 1913](#)
  - [frame, 1913](#)
  - [frameNum, 1913](#)
- [\\_flexcan\\_flags](#)
  - [FlexCAN SOMRT1061, 1139](#)
- [\\_flexcan\\_frame, 1913](#)
  - [\\_\\_pad0\\_\\_, 1913](#)

- dataByte0, [1913](#)
- dataByte1, [1913](#)
- dataByte2, [1914](#)
- dataByte3, [1914](#)
- dataByte4, [1914](#)
- dataByte5, [1914](#)
- dataByte6, [1914](#)
- dataByte7, [1914](#)
- dataWord0, [1914](#)
- dataWord1, [1914](#)
- format, [1914](#)
- id, [1914](#)
- idhit, [1914](#)
- length, [1914](#)
- timestamp, [1915](#)
- type, [1915](#)
- \_flexcan\_frame\_format
  - FlexCAN SOMRT1061, [1139](#)
- \_flexcan\_frame\_type
  - FlexCAN SOMRT1061, [1140](#)
- \_flexcan\_handle, [1915](#)
  - callback, [1915](#)
  - mbFrameBuf, [1915](#)
  - mbState, [1915](#)
  - rxFifoFrameBuf, [1915](#)
  - rxFifoFrameNum, [1915](#)
  - rxFifoState, [1916](#)
  - rxFifoTransferTotalNum, [1916](#)
  - timestamp, [1916](#)
  - userData, [1916](#)
- \_flexcan\_interrupt\_enable
  - FlexCAN SOMRT1061, [1140](#)
- \_flexcan\_mb\_transfer, [1916](#)
  - frame, [1916](#)
  - mbIdx, [1916](#)
- \_flexcan\_rx\_fifo\_config, [1917](#)
  - idFilterNum, [1917](#)
  - idFilterTable, [1917](#)
  - idFilterType, [1917](#)
  - priority, [1917](#)
- \_flexcan\_rx\_fifo\_filter\_type
  - FlexCAN SOMRT1061, [1140](#)
- \_flexcan\_rx\_fifo\_priority
  - FlexCAN SOMRT1061, [1140](#)
- \_flexcan\_rx\_mb\_config, [1917](#)
  - format, [1918](#)
  - id, [1918](#)
  - type, [1918](#)
- \_flexcan\_timing\_config, [1918](#)
  - phaseSeg1, [1918](#)
  - phaseSeg2, [1918](#)
  - preDivider, [1918](#)
  - propSeg, [1918](#)
  - rJumpwidth, [1918](#)
- \_flexcan\_wake\_up\_source
  - FlexCAN SOMRT1061, [1141](#)
- \_lpspi\_master\_config, [1923](#)
  - baudRate, [1923](#)
  - betweenTransferDelayInNanoSec, [1923](#)
  - bitsPerFrame, [1923](#)
  - cpha, [1923](#)
  - cpol, [1923](#)
  - dataOutConfig, [1923](#)
  - direction, [1923](#)
  - enableInputDelay, [1924](#)
  - lastSckToPcsDelayInNanoSec, [1924](#)
  - pcsActiveHighOrLow, [1924](#)
  - pcsToSckDelayInNanoSec, [1924](#)
  - pinCfg, [1924](#)
  - whichPcs, [1924](#)
- \_lpspi\_master\_edma\_handle, [1924](#)
  - bytesEachRead, [1925](#)
  - bytesEachWrite, [1925](#)
  - bytesLastRead, [1925](#)
  - callback, [1925](#)
  - edmaRxRegToRxDataHandle, [1925](#)
  - edmaTxDataToTxRegHandle, [1925](#)
  - fifoSize, [1925](#)
  - isByteSwap, [1925](#)
  - isPcsContinuous, [1925](#)
  - isThereExtraRxBytes, [1926](#)
  - lpspiSoftwareTCD, [1926](#)
  - nbytes, [1926](#)
  - readRegRemainingTimes, [1926](#)
  - rxBuffIfNull, [1926](#)
  - rxData, [1926](#)
  - rxRemainingByteCount, [1926](#)
  - rxWatermark, [1926](#)
  - state, [1926](#)
  - totalByteCount, [1926](#)
  - transmitCommand, [1926](#)
  - txBuffIfNull, [1926](#)
  - txData, [1927](#)
  - txRemainingByteCount, [1927](#)
  - userData, [1927](#)
  - writeRegRemainingTimes, [1927](#)
- \_lpspi\_master\_edma\_private\_handle, [1927](#)
  - base, [1927](#)
  - handle, [1927](#)
- \_lpspi\_master\_handle, [1928](#)
  - bytesEachRead, [1928](#)
  - bytesEachWrite, [1928](#)
  - bytesPerFrame, [1928](#)
  - callback, [1928](#)
  - fifoSize, [1928](#)
  - isByteSwap, [1928](#)
  - isPcsContinuous, [1929](#)
  - isTxMask, [1929](#)
  - readRegRemainingTimes, [1929](#)
  - rxData, [1929](#)
  - rxRemainingByteCount, [1929](#)
  - rxWatermark, [1929](#)
  - state, [1929](#)
  - totalByteCount, [1929](#)
  - txBuffIfNull, [1929](#)
  - txData, [1929](#)



- txRemainingByteCount, 1929
- userData, 1929
- writeRegRemainingTimes, 1930
- writeTcrInIsr, 1930
- \_lpspi\_slave\_config, 1930
  - bitsPerFrame, 1930
  - cpha, 1930
  - cpol, 1930
  - dataOutConfig, 1930
  - direction, 1930
  - pcsActiveHighOrLow, 1931
  - pinCfg, 1931
  - whichPcs, 1931
- \_lpspi\_slave\_edma\_handle, 1931
  - bytesEachRead, 1932
  - bytesEachWrite, 1932
  - bytesLastRead, 1932
  - callback, 1932
  - edmaRxRegToRxDataHandle, 1932
  - edmaTxDataToTxRegHandle, 1932
  - errorCount, 1932
  - fifoSize, 1932
  - isByteSwap, 1932
  - isThereExtraRxBytes, 1932
  - lpspiSoftwareTCD, 1932
  - nbytes, 1932
  - readRegRemainingTimes, 1932
  - rxBuffIfNull, 1933
  - rxData, 1933
  - rxRemainingByteCount, 1933
  - rxWatermark, 1933
  - state, 1933
  - totalByteCount, 1933
  - txBuffIfNull, 1933
  - txData, 1933
  - txRemainingByteCount, 1933
  - userData, 1933
  - writeRegRemainingTimes, 1933
- \_lpspi\_slave\_handle, 1934
  - bytesEachRead, 1934
  - bytesEachWrite, 1934
  - callback, 1934
  - errorCount, 1934
  - fifoSize, 1934
  - isByteSwap, 1934
  - readRegRemainingTimes, 1934
  - rxData, 1935
  - rxRemainingByteCount, 1935
  - rxWatermark, 1935
  - state, 1935
  - totalByteCount, 1935
  - txData, 1935
  - txRemainingByteCount, 1935
  - userData, 1935
  - writeRegRemainingTimes, 1935
- \_lpspi\_transfer, 1935
  - configFlags, 1936
  - dataSize, 1936
  - rxData, 1936
  - txData, 1936
- \_lut\_sequence, 1936
- \_pit\_chnl
  - Pit, 1369
- \_pit\_config, 1938
  - enableRunInDebug, 1939
- \_pit\_interrupt\_enable
  - Pit, 1369
- \_pit\_status\_flags
  - Pit, 1369
- \_qtmr\_channel\_selection
  - QTMR (Quad Timer), 1407
- \_qtmr\_config, 1939
  - debugMode, 1939
  - enableExternalForce, 1939
  - enableMasterMode, 1939
  - faultFilterCount, 1939
  - faultFilterPeriod, 1939
  - primarySource, 1940
  - secondarySource, 1940
- \_qtmr\_counting\_mode
  - QTMR (Quad Timer), 1407
- \_qtmr\_debug\_action
  - QTMR (Quad Timer), 1408
- \_qtmr\_dma\_enable
  - QTMR (Quad Timer), 1408
- \_qtmr\_input\_capture\_edge
  - QTMR (Quad Timer), 1408
- \_qtmr\_input\_source
  - QTMR (Quad Timer), 1408
- \_qtmr\_interrupt\_enable
  - QTMR (Quad Timer), 1409
- \_qtmr\_output\_mode
  - QTMR (Quad Timer), 1409
- \_qtmr\_preload\_control
  - QTMR (Quad Timer), 1409
- \_qtmr\_primary\_count\_source
  - QTMR (Quad Timer), 1410
- \_qtmr\_pwm\_out\_state
  - QTMR (Quad Timer), 1410
- \_qtmr\_status\_flags
  - QTMR (Quad Timer), 1410
- \_tempmon\_alarm\_mode
  - SOMRT1061 Temperature Monitor, 1432
- \_tempmon\_config, 1940
  - frequency, 1940
  - highAlarmTemp, 1940
  - lowAlarmTemp, 1940
  - panicAlarmTemp, 1940
- \_wake
  - DelayObject, 2051
- ~DelayObject
  - DelayObject, 2050
- ~UDPPacket
  - UDPPacket, 2386
- 1-Wire, 604
- abortsocket

- TCP, [1550](#)
- About This Guide, [133](#)
- accept
  - TCP, [1551](#), [1557](#)
- ACME Certificate Generation, [1509](#)
- ACME Certificate Generation (Easy Servlet), [839](#), [852](#)
- AcmeServletObject, [1941](#)
  - AcmeServletObject, [1941](#)
  - Delete\_Everything\_Restart, [1941](#)
  - GetDataForCertGen, [1942](#)
  - GetStateCC, [1941](#)
- Add
  - ParsedJsonDataSet, [2256–2258](#)
  - TFTP\_Server, [2361](#)
- Add a Library to a Project, [90](#)
- AddArrayElement
  - ParsedJsonDataSet, [2258–2260](#)
- AddArrayStart
  - ParsedJsonDataSet, [2260](#)
- AddChild
  - MQTT::mqtt\_leaf, [2202](#)
  - MQTT::mqtt\_obj, [2214](#)
- AddData
  - UDPPacket, [2389](#), [2390](#)
- AddDataByte
  - UDPPacket, [2390](#)
- AddDataWord
  - UDPPacket, [2391](#)
- AddDefaultGateway
  - IPv6Interface, [2134](#)
- AddEthernetInterfaces
  - Ethernet, [1208](#)
- AddInterface
  - IP, [1304](#)
  - Multihome and VLAN, [1335](#), [1336](#)
- AddMyMac
  - ParsedJsonDataSet, [2260](#)
- AddNull
  - ParsedJsonDataSet, [2260](#)
- AddObjectStart
  - ParsedJsonDataSet, [2261](#)
- addr\_hold
  - SEMC\_cfg\_t, [2312](#)
- addr\_setup
  - SEMC\_cfg\_t, [2313](#)
- addr\_wr\_hold
  - SEMC\_cfg\_t, [2313](#)
- AddStandardDHCPsServer
  - DHCP Server, [1176](#)
- AddStaticAddress
  - IPv6Interface, [2134](#)
- AddStaticDNS
  - IPv6Interface, [2135](#)
- AddStaticIPv6Address
  - DHCPv6 Client, [1179](#)
- AddToJsonOutObject
  - Dynamic\_IPADDR, [2073](#)
- AddUserAuth
  - UserAuthManager, [2422](#)
- AddVlanInterface
  - Multihome and VLAN, [1336](#), [1337](#)
- Advanced MQTT Echo Example, [581](#)
- Advanced MQTT Periodic Example, [576](#)
- Advanced MQTT Pub/Sub Example, [578](#)
- advHoldLvl
  - SEMC\_cfg\_t, [2313](#)
- AdvMux
  - SEMC, [1424](#)
- advPol
  - SEMC\_cfg\_t, [2313](#)
- AES, [373](#)
- AES - Advanced Encryption Standard, [1099](#)
  - aes\_cbc\_decrypt, [1099](#)
  - aes\_cbc\_encrypt, [1101](#)
  - aes\_decrypt, [1101](#)
  - aes\_encrypt, [1101](#)
  - aes\_self\_test, [1102](#)
  - aes\_set\_key, [1102](#)
- aes\_cbc\_decrypt
  - AES - Advanced Encryption Standard, [1099](#)
- aes\_cbc\_encrypt
  - AES - Advanced Encryption Standard, [1101](#)
- aes\_context, [1942](#)
  - drk, [1942](#)
  - erk, [1942](#)
  - nr, [1942](#)
- aes\_decrypt
  - AES - Advanced Encryption Standard, [1101](#)
- aes\_encrypt
  - AES - Advanced Encryption Standard, [1101](#)
- aes\_self\_test
  - AES - Advanced Encryption Standard, [1102](#)
- aes\_set\_key
  - AES - Advanced Encryption Standard, [1102](#)
- AgeStillPreferred
  - IPV6\_PREFIX, [2123](#)
- AgeStillValidTest
  - IPV6\_DEST, [2114](#)
  - IPV6\_DHCPD, [2115](#)
  - IPV6\_DNS, [2117](#)
  - IPV6\_IA\_ADDR, [2118](#)
  - IPV6\_NEIGHBOR, [2119](#)
  - IPV6\_PREFIX, [2123](#)
  - IPV6\_ROOT\_EL, [2125](#)
  - IPV6\_ROUTER, [2129](#)
- Ajax Graph, [985](#)
- ALLOC\_STRING
  - JSON Lexer, [1332](#)
- AlreadyConnected
  - NB::Error::Connect, [1902](#)
- AlreadyInit
  - NB::Error::Init, [1903](#)
- ALT\_NAME\_TYPE\_DNS
  - Onboard Self-Signed Certificate Generation, [1511](#)
- ALT\_NAME\_TYPE\_IP
  - Onboard Self-Signed Certificate Generation, [1511](#)

- ALT\_NAME\_TYPE\_NONE
  - Onboard Self-Signed Certificate Generation, 1511
- AltNameType\_t
  - Onboard Self-Signed Certificate Generation, 1511
- Analog to Digital, 607
- Analog to Digital (ADC), 721, 773
- AnyDNSInterFaceActive
  - DNS - Domain Name System, 1181
- Append
  - NBInterpolatedString, 2235
  - NBString, 2239
- Application Data, 387
- Application Image Checksum, 374
- Application Image Security, 374
- Appplication Update, 455
- ARM Cache Management, 126
- ARM\_MPU\_Region\_t, 1942
- ARP, 1943
- ARP - Address Resolution Protocol, 1102
  - fShowArp, 1107
  - GetArpMacFromIp, 1108
  - GetProperInterface4, 1110
  - hard\_size, 1117
  - hard\_Type, 1117
  - IsMyAddressUsedArpDetect, 1111
  - op\_code, 1117
  - PARP, 1107
  - prot\_size, 1117
  - prot\_Type, 1117
  - sender\_ip, 1117
  - sender\_phy, 1117
  - sendGratuitousArp, 1113
  - ShowArp, 1115
  - target\_ip, 1117
  - target\_phy, 1118
- ARP\_CONFLICT
  - DHCP, 1894
- AsciiTolp4
  - Utilities, 1782
- AsciiTolp6
  - IPADDR6, 2107
  - Utilities, 1782
- AsciiToMac
  - Utilities, 1782
- Assign
  - MQTT::mqtt\_leaf, 2202
- AssignUartNumber
  - SerialRecord, 2316
- Async
  - SEMC, 1426
- AT Commands, 1118
  - ProcessATcommandFunc, 1119
- Attach
  - I4Record, 2094
  - I6Record, 2095
  - IPV6\_ROOT\_EL\_CONTAINER, 2126
- ATTR
  - \_edma\_tcd, 1909
- Authorization Responses, 1778
  - AuthResponse, 1778
  - eAuthErrorAuthCheckFailed, 1778
  - eAuthErrorAuthTypeMismatch, 1778
  - eAuthErrorFailedRecordUpdate, 1778
  - eAuthErrorNoEmptyUserAuthRecords, 1778
  - eAuthErrorSaveFailed, 1778
  - eAuthErrorUnableToAddUser, 1778
  - eAuthErrorUnableToCreateHash, 1778
  - eAuthErrorUserDoesNotExist, 1778
  - eAuthErrorUserExists, 1778
  - eAuthSuccess, 1778
- Authorization Types, 1778
  - AuthType, 1779
  - eAuthTypeDefault, 1779
  - eAuthTypeKey, 1779
  - eAuthTypePassword, 1779
- AuthResponse
  - Authorization Responses, 1778
- AuthType
  - Authorization Types, 1779
- AuthWithGssApi
  - SOCKS, 1445
- automatic\_retransmission
  - mcanMODM7AE70::mcan\_config, 2158
- available
  - TwoWire, 2367
  - WireIntf, 2525
- BadRequestResponse
  - HTTP and HTML Functions, 1261
- base
  - \_edma\_handle, 1907
  - \_lpspi\_master\_edma\_private\_handle, 1927
- Base64, 1119
  - Base64Decode, 1120
  - Base64Encode, 1120
  - Base64StreamEncode, 1120
  - Base64UrlDecode, 1121
  - Base64UrlEncode, 1121
  - Base64UrlStreamEncode, 1122
- Base64Decode
  - Base64, 1120
- Base64Encode
  - Base64, 1120
- Base64StreamEncode
  - Base64, 1120
- Base64UrlDecode
  - Base64, 1121
- Base64UrlEncode
  - Base64, 1121
- Base64UrlStreamEncode
  - Base64, 1122
- Basic, 455
- Basic Config Variables, 387
- Basic Web Config, 417
- Basic Web Config with SSL/TLS, 417
- baudRate
  - \_flexcan\_config, 1912

- [\\_lpspi\\_master\\_config](#), 1923
- BBI2C, 1943
  - BBI2C, 1945
  - I2C\_RES\_ACK, 1944
  - I2C\_RES\_ARB\_LST, 1944
  - I2C\_RES\_ARG, 1944
  - I2C\_RES\_BUSY, 1944
  - I2C\_RES\_ERR, 1944
  - I2C\_RES\_IN\_PROGRESS, 1944
  - I2C\_RES\_NACK, 1944
  - readReg8, 1945
  - readRegN, 1945
  - Result\_t, 1944
  - SetMaxBusDelay, 1945
  - setNumAddressBytes, 1947
  - setup, 1947
  - writeReg8, 1947
  - writeRegN, 1947
- bDoFallBack
  - DhcpObject, 2055
- begin
  - TwoWire, 2367
- BEGIN\_ARRAY
  - JSON Lexer, 1331
- BEGIN\_OBJECT
  - JSON Lexer, 1331
- beginTransmission
  - TwoWire, 2367
  - WireIntf, 2526
- betweenTransferDelayInNanoSec
  - \_lpspi\_master\_config, 1923
- blsIPv6
  - UDPPacket, 2391
- Bit Bang I2C (GPIO), 1273
- BITER
  - \_edma\_tcd, 1909
- bitOrder
  - SAI\_rtx\_cfg\_t, 2301
  - SSC\_rtx\_cfg\_t, 2344
- bitOrder\_t
  - SAI Configuration Enumerations, 1529
- bitRate
  - \_flexcan\_config, 1912
- bitsPerFrame
  - \_lpspi\_master\_config, 1923
  - \_lpspi\_slave\_config, 1930
- bitsPerWord
  - SAI\_rtx\_cfg\_t, 2301
  - SSC\_rtx\_cfg\_t, 2344
- blocking\_send\_message
  - mcanMODM7AE70::mcan\_module, 2161
- Board Lock, 376
- Bootstrap Hello World, 987
- BSS Options, 1888
- BSSID\_NotFound
  - NB::Error::Connect, 1902
- Buffer Ownership States, 1126
- Buffer State Flags, 1126
- buffer\_list, 1948
  - GetCount, 1949
  - InsertAfter, 1949
  - InsertBefore, 1949
  - InsertHead, 1949
  - InsertTail, 1950
  - Remove, 1950
  - RemoveHead, 1950
  - RemoveTail, 1950
- buffer\_object, 1950
- BUFFER\_WORD16\_LEFT
  - SAI Configuration Enumerations, 1531
- BUFFER\_WORD16\_RIGHT
  - SAI Configuration Enumerations, 1531
- BUFFER\_WORD32\_LEFT
  - SAI Configuration Enumerations, 1531
- BUFFER\_WORD32\_RIGHT
  - SAI Configuration Enumerations, 1531
- BUFFER\_WORD8
  - SAI Configuration Enumerations, 1530
- bufferDepletionBehavior\_t
  - SAI Configuration Enumerations, 1529
- Buffers - System Buffer Pool, 1123
  - FreeBuffer, 1123
  - FreeBufferList, 1124
  - GetBuffer, 1124
  - GetFastBuffer, 1124
  - GetFreeCount, 1124
  - IncUsageCount, 1125
  - InitBuffers, 1125
  - ShowBuffer, 1125
- bufnstr
  - Utilities, 1783
- Build System, 7
- Burst\_1
  - SEMC, 1425
- Burst\_16
  - SEMC, 1425
- Burst\_2
  - SEMC, 1425
- Burst\_32
  - SEMC, 1425
- Burst\_4
  - SEMC, 1425
- Burst\_64
  - SEMC, 1425
- Burst\_8
  - SEMC, 1425
- burstLen
  - SEMC\_cfg\_t, 2313
- Bus
  - SEMC, 1424
- bUsCanada
  - TimeZoneRecord, 2363
- BusTimeout
  - NB::Error, 1902
- busWidth
  - EBI - External Bus Interface, 1401

- SEMC\_cfg\_t, [2313](#)
- busyBitPolarity
  - \_FlexSPIConfig, [1920](#)
- BuyPassAcmeServletObject, [1951](#)
- byteAccess
  - EBI - External Bus Interface, [1401](#)
- bytesEachRead
  - \_lpspi\_master\_edma\_handle, [1925](#)
  - \_lpspi\_master\_handle, [1928](#)
  - \_lpspi\_slave\_edma\_handle, [1932](#)
  - \_lpspi\_slave\_handle, [1934](#)
- bytesEachWrite
  - \_lpspi\_master\_edma\_handle, [1925](#)
  - \_lpspi\_master\_handle, [1928](#)
  - \_lpspi\_slave\_edma\_handle, [1932](#)
  - \_lpspi\_slave\_handle, [1934](#)
- bytesLastRead
  - \_lpspi\_master\_edma\_handle, [1925](#)
  - \_lpspi\_slave\_edma\_handle, [1932](#)
- bytesPerFrame
  - \_lpspi\_master\_handle, [1928](#)
- c\_str
  - config\_string, [2023](#)
  - NBString, [2239](#)
- callback
  - \_edma\_handle, [1907](#)
  - \_flexcan\_handle, [1915](#)
  - \_lpspi\_master\_edma\_handle, [1925](#)
  - \_lpspi\_master\_handle, [1928](#)
  - \_lpspi\_slave\_edma\_handle, [1932](#)
  - \_lpspi\_slave\_handle, [1934](#)
- CallbackFunctionPageHandler, [1951](#)
  - CallbackFunctionPageHandler, [1952](#), [1953](#)
  - ProcessRaw, [1953](#)
- CallbackFunctionPostHandler, [1954](#)
  - ProcessRaw, [1954](#)
- CAN, [726](#), [1127](#)
- CAN Send and Receive, [721](#)
- CAN to Serial, [617](#)
- canMCF5441x, [1893](#)
- canMCF5441x::CanRxMessage, [1954](#)
  - CanRxMessage, [1955](#)
  - GetData, [1956](#)
  - GetNewMessage, [1957](#)
  - IsValid, [1957](#)
- CanRxMessage
  - canMCF5441x::CanRxMessage, [1955](#)
  - mcanMODM7AE70::CanRxMessage, [1958](#)
- CERT\_GEN\_RETURN\_CERT\_CREATE\_ERROR
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CERT\_GEN\_RETURN\_CERT\_INIT\_ERROR
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CERT\_GEN\_RETURN\_CERT\_SAVING\_ERROR
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CERT\_GEN\_RETURN\_CERT\_SIGNING\_ERROR
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CERT\_GEN\_RETURN\_ECC\_CREATE\_ERROR
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CERT\_GEN\_RETURN\_ECC\_INIT\_ERROR
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CERT\_GEN\_RETURN\_INVALID\_RANDOM
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CERT\_GEN\_RETURN\_KEY\_CONVERT\_ERROR
  - Onboard Self-Signed Certificate Generation, [1512](#)
- CERT\_GEN\_RETURN\_KEY\_SAVING\_ERROR
  - Onboard Self-Signed Certificate Generation, [1512](#)
- CERT\_GEN\_RETURN\_MALLOC\_ERROR
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CERT\_GEN\_RETURN\_NOT\_REQUIRED
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CERT\_GEN\_RETURN\_RSA\_CREATE\_ERROR
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CERT\_GEN\_RETURN\_RSA\_INIT\_ERROR
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CERT\_GEN\_RETURN\_SET\_ISSUER\_ERROR
  - Onboard Self-Signed Certificate Generation, [1512](#)
- CERT\_GEN\_RETURN\_SUCCESS
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CertData, [1960](#)
- CertGenData, [1960](#)
- CertGenReturnCode
  - Onboard Self-Signed Certificate Generation, [1511](#)
- CF\_ACCESSERROR\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1375](#)
- CF\_ADDRESSERROR\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1376](#)
- CF\_AUTOVECTOR\_IRQ1
  - ColdFire Exception and Interrupt Vectors, [1376](#)
- CF\_AUTOVECTOR\_IRQ2
  - ColdFire Exception and Interrupt Vectors, [1376](#)
- CF\_AUTOVECTOR\_IRQ3
  - ColdFire Exception and Interrupt Vectors, [1376](#)
- CF\_AUTOVECTOR\_IRQ4
  - ColdFire Exception and Interrupt Vectors, [1377](#)
- CF\_AUTOVECTOR\_IRQ5
  - ColdFire Exception and Interrupt Vectors, [1377](#)
- CF\_AUTOVECTOR\_IRQ6
  - ColdFire Exception and Interrupt Vectors, [1377](#)
- CF\_AUTOVECTOR\_IRQ7
  - ColdFire Exception and Interrupt Vectors, [1377](#)
- CF\_DEBUG\_INTERRUPT
  - ColdFire Exception and Interrupt Vectors, [1377](#)
- CF\_FORMAT\_ERR\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1377](#)
- CF\_ILLEGAL\_INSTRUCTION\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1378](#)
- CF\_PRIVLEDGE\_VIOLATION\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1378](#)
- CF\_SPURIOUS\_INT\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1378](#)
- CF\_TRACE\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1379](#)
- CF\_TRAP0\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1379](#)
- CF\_TRAP10\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1380](#)

- CF\_TRAP11\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1380](#)
- CF\_TRAP12\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1380](#)
- CF\_TRAP13\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1380](#)
- CF\_TRAP14\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1380](#)
- CF\_TRAP15\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1380](#)
- CF\_TRAP1\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1381](#)
- CF\_TRAP2\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1381](#)
- CF\_TRAP3\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1381](#)
- CF\_TRAP4\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1381](#)
- CF\_TRAP5\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1381](#)
- CF\_TRAP6\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1381](#)
- CF\_TRAP7\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1381](#)
- CF\_TRAP8\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1381](#)
- CF\_TRAP9\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1381](#)
- CF\_UNIMPLMENTED\_A\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1382](#)
- CF\_UNIMPLMENTED\_F\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1382](#)
- CF\_UNINITIALIZED\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1382](#)
- CF\_USER\_BASE\_VECTOR
  - ColdFire Exception and Interrupt Vectors, [1382](#)
- Chaining Timers, [712](#)
- Change CompCode Settings for EDFS, [91](#)
- Change Notify, [432](#)
- Change The IP Targeted By A Project, [91](#)
- channel
  - \_edma\_handle, [1907](#)
- channelPriority
  - \_edma\_channel\_Preemption\_config, [1905](#)
- charavail
  - IOSYS - I/O System, [1283](#)
- CharOutputFn
  - JSON Lexer, [1331](#)
- Check Lock, [376](#)
- CheckDnsCache
  - DNS - Domain Name System, [1181](#)
- CheckHttpAccess
  - HTTP and HTML Functions, [1261](#)
- CheckUserAuth
  - UserAuthManager, [2422](#), [2423](#)
- CheckUserAuthLevel
  - UserAuthManager, [2423](#)
- CHIP\_SELECT\_0
  - MCF5441x (DSPI), [1453](#)
  - MIMX10xx (DSPI), [1464](#)
  - SAME70 (DSPI), [1473](#)
- CHIP\_SELECT\_1
  - MCF5441x (DSPI), [1453](#)
  - MIMX10xx (DSPI), [1464](#)
- CHIP\_SELECT\_2
  - MCF5441x (DSPI), [1453](#)
  - MIMX10xx (DSPI), [1464](#)
- CHIP\_SELECT\_3
  - MCF5441x (DSPI), [1453](#)
  - MIMX10xx (DSPI), [1464](#)
- CHIP\_SELECT\_DISABLED
  - MCF5441x (DSPI), [1453](#)
  - MIMX10xx (DSPI), [1464](#)
  - SAME70 (DSPI), [1473](#)
- Cipher Options, [1889](#)
- Cipher\_NotFound
  - NB::Error::Connect, [1902](#)
- CITER
  - \_edma\_tcd, [1909](#)
- Clear
  - StopWatch, [2349](#)
- clear
  - NBString, [2240](#)
- Clear Configuration Flash, [385](#)
- ClearCustomNetDoRX
  - Low Level Processing (NetDoRx), [1348](#)
- ClearFlags
  - MQTT::mqtt\_IO, [2189](#)
  - MQTT::mqtt\_leaf, [2203](#)
- Client
  - MQTT::Client, [1964](#)
- CLK\_ACTIVE\_HIGH
  - SAI Configuration Enumerations, [1530](#)
- CLK\_ACTIVE\_LOW
  - SAI Configuration Enumerations, [1530](#)
- CLK\_CH\_CH
  - SAI Configuration Enumerations, [1529](#)
- CLK\_CH\_RK
  - SAI Configuration Enumerations, [1529](#)
- CLK\_CH\_TK
  - SAI Configuration Enumerations, [1529](#)
- CLK\_GATE\_CONTINUOUS
  - SAI Configuration Enumerations, [1529](#)
- CLK\_GATE\_FRAME\_HIGH
  - SAI Configuration Enumerations, [1529](#)
- CLK\_GATE\_FRAME\_LOW
  - SAI Configuration Enumerations, [1529](#)
- CLK\_OUT\_CONTINUOUS
  - SAI Configuration Enumerations, [1530](#)
- CLK\_OUT\_INPUT
  - SAI Configuration Enumerations, [1530](#)
- CLK\_OUT\_TRANSFER
  - SAI Configuration Enumerations, [1530](#)
- CLK\_SRC\_EXT
  - SAI Configuration Enumerations, [1530](#)
- CLK\_SRC\_MCK



- SAI Configuration Enumerations, 1530
- clkCh
  - SAI\_rtx\_cfg\_t, 2301
- clkCh\_t
  - SAI Configuration Enumerations, 1529
- clkDiv
  - SSC\_cfg\_t, 2343
- clkGate
  - SAI\_rtx\_cfg\_t, 2302
  - SSC\_rtx\_cfg\_t, 2344
- clkGate\_t
  - SAI Configuration Enumerations, 1529
- clkMode
  - SEMC\_cfg\_t, 2313
- clkOut
  - SAI\_rtx\_cfg\_t, 2302
  - SSC\_rtx\_cfg\_t, 2345
- clkOut\_t
  - SAI Configuration Enumerations, 1529
- clkPin
  - SEMC\_cfg\_t, 2313
- clkPol
  - SAI\_rtx\_cfg\_t, 2302
- clkPol\_t
  - SAI Configuration Enumerations, 1530
- clkSrc
  - \_flexcan\_config, 1912
  - SAI\_rtx\_cfg\_t, 2302
  - SSC\_rtx\_cfg\_t, 2345
- clkSrc\_t
  - SAI Configuration Enumerations, 1530
- Clock Module Example, 729
- clock\_stop\_acknowledge
  - mcanMODM7AE70::mcan\_config, 2158
- clock\_stop\_request
  - mcanMODM7AE70::mcan\_config, 2158
- close
  - IOSYS - I/O System, 1283
- CloseListenPort
  - SerialRecord, 2316
- clroption
  - NB::WebSocket, 2428
- ClrSem
  - DSPIModule, 2066
  - SPIModule, 2335
- clrsockoption
  - TCP, 1560
- clrStickyRxError
  - SAICtx\_t, 2305
- clrStickyTxError
  - SAICtx\_t, 2305
- ClrUserAuthLevel
  - UserAuthManager, 2423
- CmdAddCommandFd
  - Command Processor, 1160
- CmdAuthenticateFunc
  - Command Processor, 1165
- CmdAuthenticateSshFunc
  - Command Processor, 1165
- CmdChar\_func
  - Command Processor, 1165
- CmdCmd\_func
  - Command Processor, 1166
- CmdConnect\_func
  - Command Processor, 1166
- CmdDisconnect\_func
  - Command Processor, 1167
- CmdIdleTimeout
  - Command Processor, 1167
- CmdListenOnSshPort
  - Command Processor, 1161
- CmdListenOnTcpPort
  - Command Processor, 1161
- CmdListenQuietOnSshPort
  - Command Processor, 1161
- CmdListenQuietOnTcpPort
  - Command Processor, 1162
- CmdPrompt\_func
  - Command Processor, 1167
- CmdRemoveCommandFd
  - Command Processor, 1162
- CmdStartCommandProcessor
  - Command Processor, 1162
- CmdStopListeningOnSshPort
  - Command Processor, 1164
- CmdStopListeningOnTcpPort
  - Command Processor, 1164
- colAddrWidth
  - SEMC\_cfg\_t, 2313
- ColdFire Exception and Interrupt Vectors, 1373
  - CF\_ACCESSERROR\_VECTOR, 1375
  - CF\_ADDRESSERROR\_VECTOR, 1376
  - CF\_AUTOVECTOR\_IRQ1, 1376
  - CF\_AUTOVECTOR\_IRQ2, 1376
  - CF\_AUTOVECTOR\_IRQ3, 1376
  - CF\_AUTOVECTOR\_IRQ4, 1377
  - CF\_AUTOVECTOR\_IRQ5, 1377
  - CF\_AUTOVECTOR\_IRQ6, 1377
  - CF\_AUTOVECTOR\_IRQ7, 1377
  - CF\_DEBUG\_INTERRUPT, 1377
  - CF\_FORMAT\_ERR\_VECTOR, 1377
  - CF\_ILLEGAL\_INSTRUCTION\_VECTOR, 1378
  - CF\_PRIVLEDGE\_VIOLATION\_VECTOR, 1378
  - CF\_SPURIOUS\_INT\_VECTOR, 1378
  - CF\_TRACE\_VECTOR, 1379
  - CF\_TRAP0\_VECTOR, 1379
  - CF\_TRAP10\_VECTOR, 1380
  - CF\_TRAP11\_VECTOR, 1380
  - CF\_TRAP12\_VECTOR, 1380
  - CF\_TRAP13\_VECTOR, 1380
  - CF\_TRAP14\_VECTOR, 1380
  - CF\_TRAP15\_VECTOR, 1380
  - CF\_TRAP1\_VECTOR, 1381
  - CF\_TRAP2\_VECTOR, 1381
  - CF\_TRAP3\_VECTOR, 1381
  - CF\_TRAP4\_VECTOR, 1381

- CF\_TRAP5\_VECTOR, 1381
- CF\_TRAP6\_VECTOR, 1381
- CF\_TRAP7\_VECTOR, 1381
- CF\_TRAP8\_VECTOR, 1381
- CF\_TRAP9\_VECTOR, 1381
- CF\_UNIMPLMENTED\_A\_VECTOR, 1382
- CF\_UNIMPLMENTED\_F\_VECTOR, 1382
- CF\_UNINITIALIZED\_VECTOR, 1382
- CF\_USER\_BASE\_VECTOR, 1382
- columnAddressWidth
  - \_FlexSPIConfig, 1920
- Command Line Tools, 15
- Command Processor, 1159
  - CmdAddCommandFd, 1160
  - CmdAuthenticateFunc, 1165
  - CmdAuthenticateSshFunc, 1165
  - CmdChar\_func, 1165
  - CmdCmd\_func, 1166
  - CmdConnect\_func, 1166
  - CmdDisconnect\_func, 1167
  - CmdIdleTimeout, 1167
  - CmdListenOnSshPort, 1161
  - CmdListenOnTcpPort, 1161
  - CmdListenQuietOnSshPort, 1161
  - CmdListenQuietOnTcpPort, 1162
  - CmdPrompt\_func, 1167
  - CmdRemoveCommandFd, 1162
  - CmdStartCommandProcessor, 1162
  - CmdStopListeningOnSshPort, 1164
  - CmdStopListeningOnTcpPort, 1164
  - SendToAll, 1164
- Command Processor Disconnect Causes, 1168
- Command Processor Listen Channels, 1168
  - eListenOnSsh, 1168
  - eListenOnTcp, 1168
  - ListenOn, 1168
- Command Processor Response Codes, 1168
- CommitTestedValue
  - CUR\_IPADDR4, 2047
  - Dynamic\_IPADDR, 2073
- compare
  - NBString, 2240
- config
  - \_PinVector, 1937
- Config and System Params, 402
- Config Class, 391
- Config Class Receive Notification, 434
- Config Server Programming Guide, 149
- Config Tags, 418
- Config Web Interfaces, 416
- config\_bool, 1976
  - config\_bool, 1977
  - GetTextValue, 1978
  - GetTypeValue, 1978
  - operator bool, 1978
  - operator=, 1978, 1979
- config\_chooser, 1979
  - config\_chooser, 1980
- GetChoices, 1981
- GetTypeValue, 1981
- IsInChoices, 1981, 1982
- IsSelected, 1982
- operator NBString, 1983
- operator=, 1983
- SetChoices, 1983
- config\_double, 1984
  - config\_double, 1985
  - GetTextValue, 1985
  - GetTypeValue, 1985
  - operator double, 1986
  - operator float, 1986
  - operator int, 1986
  - operator=, 1986
- config\_double\_limit, 1987
  - config\_double\_limit, 1988
  - GetTypeValue, 1989
  - operator=, 1989
- config\_int, 1989
  - config\_int, 1990
  - GetTextValue, 1991
  - GetTypeValue, 1991
  - operator int, 1991
  - operator=, 1991
- config\_int\_limit, 1993
  - config\_int\_limit, 1994
  - operator=, 1994
- config\_IPADDR, 1995
  - config\_IPADDR, 1996, 1997
  - GetTextValue, 1997
  - GetTypeValue, 1997
  - IsNull, 1997
  - NotNull, 1998
  - operator IPADDR, 1998
  - operator=, 1998
- config\_IPADDR4, 1999
  - config\_IPADDR4, 2000
  - GetTextValue, 2001
  - GetTypeValue, 2001
  - IsNull, 2001
  - NotNull, 2001
  - operator IPADDR4, 2003
  - operator=, 2003
  - SetNull, 2003
- config\_localname, 2003
  - config\_localname, 2005, 2006
  - operator NBString, 2006
  - operator=, 2006, 2007
- config\_MACADR, 2007
  - config\_MACADR, 2008, 2009
  - GetTextValue, 2009
  - GetTypeValue, 2009
  - operator MACADR, 2011
  - operator=, 2011
- config\_obj, 2011
  - config\_obj, 2012, 2013
  - GetTextValue, 2013



- GetTypeValue, 2013
- config\_pass, 2013
  - config\_pass, 2015, 2016
  - GetRawValue, 2016
  - GetTextValue, 2016
  - operator NBString, 2016
  - operator=, 2017
- config\_pass\_limit, 2018
  - config\_pass\_limit, 2019
  - operator=, 2020
- config\_string, 2020
  - c\_str, 2023
  - config\_string, 2022
  - GetTextValue, 2023
  - GetTypeValue, 2023
  - length, 2023
  - operator NBString, 2024
  - operator=, 2024
  - operator[], 2024
  - SetEnumList, 2024
- config\_string\_limit, 2025
  - config\_string\_limit, 2026
  - operator=, 2028
- config\_time\_t, 2028
  - config\_time\_t, 2030, 2031
  - operator time\_t, 2031
  - operator=, 2031, 2032
- config\_uint, 2032
  - config\_uint, 2033
  - GetTextValue, 2033
  - GetTypeValue, 2033
  - operator uint32\_t, 2034
  - operator=, 2034
- config\_uint\_limit, 2034
  - config\_uint\_limit, 2035, 2036
  - operator=, 2036
- config\_value, 2037
  - config\_value, 2037
- ConfigExposer
  - MQTT::ConfigExposer, 2041
- configFlags
  - \_lpspi\_transfer, 1936
- ConfigMaxSize
  - Configuration Server, 1170
- ConfigRenderFunc
  - HTTP and HTML Functions, 1262
- ConfigSize
  - Configuration Server, 1170
- Configuration, 387
- Configuration Errors, 1889
- Configuration Network Objects, 1169
- Configuration Server, 1170
  - ConfigMaxSize, 1170
  - ConfigSize, 1170
  - EnableConfigMirror, 1170
  - SaveConfigToStorage, 1171
- Configuration System, 1169
- Configuration Time Objects, 1171
- Configuration Variable Flags, 1173
- Configuration Variables, 1171
  - OverrideConfig, 1172
- Configure AP with JSON, 1084
- ConfiguredClient
  - MQTT::ConfiguredClient, 2046
- ConfigureEBI\_CS
  - EBI - External Bus Interface, 1391, 1393
- ConfigureEBI\_CSPin
  - EBI - External Bus Interface, 1397
- ConfigureEBI\_NRDPin
  - EBI - External Bus Interface, 1398
- ConfigureEBI\_NWRPin
  - EBI - External Bus Interface, 1400
- ConfigureSEMC
  - SEMC, 1427, 1429
- Connect
  - MQTT::Client, 1965
  - NB::WebSocket, 2430, 2435, 2438
- connect
  - TCP, 1563, 1570
- Connect Request Errors, 1890
- ConnectErrors
  - NB::Error::Connect, 1902
- ConnectFailed
  - NB::Error::Connect, 1902
- connectvia
  - TCP, 1577, 1585, 1594, 1600
- connectwlocal
  - TCP, 1608, 1616
- Constants (constants.h), 1173
  - TICKS\_PER\_SECOND, 1175
- CONTENT\_TYPE\_BINARY\_ATTACH
  - SMTP - Send Email, 1435
- CONTENT\_TYPE\_END
  - SMTP - Send Email, 1435
- CONTENT\_TYPE\_ENUM
  - SMTP - Send Email, 1435
- CONTENT\_TYPE\_HTML\_DECOMP
  - SMTP - Send Email, 1435
- CONTENT\_TYPE\_PLAIN\_TEXT
  - SMTP - Send Email, 1435
- CONTENT\_TYPE\_PLAIN\_TEXT\_ATTACH
  - SMTP - Send Email, 1435
- controllerMiscOption
  - \_FlexSPIConfig, 1920
- Convert
  - StopWatch, 2350
- CONVERT\_BUFFER\_LENGTH\_MAX
  - PEM Conversion Utilities, 1351
- CONVERT\_CERTIFICATE
  - PEM Data Type Identifiers, 1359
- CONVERT\_DSA\_PRIVATE\_KEY
  - PEM Data Type Identifiers, 1359
- CONVERT\_ECDSA\_PRIVATE\_KEY
  - PEM Data Type Identifiers, 1359
- CONVERT\_NONE
  - PEM Data Type Identifiers, 1359

- CONVERT\_RSA\_PRIVATE\_KEY
  - PEM Data Type Identifiers, [1359](#)
- convertBinaryToHexAscii
  - Utilities, [1783](#)
- convertHexAsciiToBinary
  - Utilities, [1787](#)
- convertLittleEndianDword
  - PEM Conversion Utilities, [1352](#)
- convertLittleEndianWord
  - PEM Conversion Utilities, [1353](#)
- ConvertPEMFormattedData
  - PEM Conversion Utilities, [1354](#)
- copy
  - NBString, [2240](#)
- CopyCurrentString
  - ParsedJsonDataSet, [2261](#)
- CopyData
  - mcanMODM7AE70::CanRxMessage, [1958](#)
- CopyObject
  - ParsedJsonDataSet, [2261](#)
- Core Watchdog Timer (CWT), [670](#)
- CoreSendPend
  - IPV6\_NEIGHBOR, [2119](#)
- CouldNotConfig
  - NB::Error::Connect, [1902](#)
- CountResolution
  - StopWatch, [2351](#)
- cpha
  - \_lpspi\_master\_config, [1923](#)
  - \_lpspi\_slave\_config, [1930](#)
- cpol
  - \_lpspi\_master\_config, [1923](#)
  - \_lpspi\_slave\_config, [1930](#)
- CR0
  - SEMC\_cfg\_t, [2313](#)
- CR1
  - SEMC\_cfg\_t, [2313](#)
- CR2
  - SEMC\_cfg\_t, [2313](#)
- CR\_CLEAR\_NON\_GRP\_PRIO
  - MCF5441x (DSPI), [1450](#)
- CR\_SET\_RR\_CH\_ARB
  - MCF5441x (DSPI), [1450](#)
- CreateRxTxUdpSocket
  - UDP Socket API, [1766](#)
- CreateRxUdpSocket
  - UDP Socket API, [1768](#)
- CreateTxUdpSocket
  - UDP Socket API, [1771](#)
- CS\_ASSERT\_HIGH
  - MCF5441x (DSPI), [1454](#)
  - MIMX10xx (DSPI), [1464](#)
  - SAME70 (DSPI), [1469](#)
- CS\_ASSERT\_LOW
  - MCF5441x (DSPI), [1454](#)
  - MIMX10xx (DSPI), [1464](#)
  - SAME70 (DSPI), [1469](#)
- cs\_hold\_min
  - SEMC\_cfg\_t, [2314](#)
- cs\_min\_intv
  - SEMC\_cfg\_t, [2314](#)
- cs\_setup
  - SEMC\_cfg\_t, [2314](#)
- csPin
  - SEMC\_cfg\_t, [2314](#)
- CSR
  - \_edma\_tcd, [1909](#)
- csReturnType
  - MCF5441x (DSPI), [1453](#)
  - MIMX10xx (DSPI), [1463](#)
  - SAME70 (DSPI), [1469](#)
- CTAR\_CLOCK\_PHASE
  - MCF5441x (DSPI), [1450](#)
- CTAR\_CLOCK\_POLARITY
  - MCF5441x (DSPI), [1450](#)
- CTAR\_FRAME\_16BIT
  - MCF5441x (DSPI), [1451](#)
- CTAR\_FRAME\_4BIT
  - MCF5441x (DSPI), [1451](#)
- CTX\_STATE\_INIT
  - SAICtx\_t, [2305](#)
- CTX\_STATE\_RX
  - SAICtx\_t, [2305](#)
- CTX\_STATE\_STOPPED
  - SAICtx\_t, [2305](#)
- CTX\_STATE\_TX
  - SAICtx\_t, [2305](#)
- CTX\_STATE\_TXRX
  - SAICtx\_t, [2305](#)
- ctxState\_t
  - SAICtx\_t, [2305](#)
- CUR\_IPADDR4, [2047](#)
  - CommitTestedValue, [2047](#)
  - CUR\_IPADDR4, [2047](#)
  - ExtendedSchema, [2047](#)
  - GetTextValue, [2048](#)
  - GetTypeValue, [2048](#)
  - IsAutoIP, [2048](#)
  - IsGlobalBroadCast, [2048](#)
  - IsLoopBack, [2048](#)
  - IsMultiCast, [2048](#)
  - IsNull, [2048](#)
  - NotNull, [2048](#)
  - operator IPADDR4, [2048](#)
  - SetNull, [2048](#)
  - TestNewValue, [2048](#)
- CurrentBool
  - ParsedJsonDataSet, [2261](#)
- CurrentName
  - ParsedJsonDataSet, [2262](#)
- CurrentNumber
  - ParsedJsonDataSet, [2262](#)
- CurrentStdioFD
  - IOSYS - I/O System, [1283](#)
- CurrentString
  - ParsedJsonDataSet, [2262](#)

- Custom Branding, [169](#)
- Custom Range Limit Config Variables, [393](#)
- Custom Web Config, [422](#)
- Custom Web System Config, [425](#)
- CustomNetDoRX
  - Low Level Processing (NetDoRx), [1350](#)
- DACC Tone Generator, [725](#)
- DADDR
  - \_edma\_tcd, [1909](#)
- dataavail
  - IOSYS - I/O System, [1284](#)
- dataByte0
  - \_flexcan\_frame, [1913](#)
- dataByte1
  - \_flexcan\_frame, [1913](#)
- dataByte2
  - \_flexcan\_frame, [1914](#)
- dataByte3
  - \_flexcan\_frame, [1914](#)
- dataByte4
  - \_flexcan\_frame, [1914](#)
- dataByte5
  - \_flexcan\_frame, [1914](#)
- dataByte6
  - \_flexcan\_frame, [1914](#)
- dataByte7
  - \_flexcan\_frame, [1914](#)
- dataOutConfig
  - \_lpspi\_master\_config, [1923](#)
  - \_lpspi\_slave\_config, [1930](#)
- dataPacking
  - SAI\_rtx\_cfg\_t, [2302](#)
- dataPacking\_t
  - SAI Configuration Enumerations, [1530](#)
- dataSize
  - \_lpspi\_transfer, [1936](#)
- dataValid
  - SSC\_rtx\_cfg\_t, [2345](#)
- dataWord0
  - \_flexcan\_frame, [1914](#)
- dataWord1
  - \_flexcan\_frame, [1914](#)
- DB\_TCPIP
  - Utilities, [1781](#)
- DEASSERT\_AFTER\_LAST
  - MCF5441x (DSPI), [1453](#)
  - MIMX10xx (DSPI), [1463](#)
  - SAME70 (DSPI), [1469](#)
- DEASSERT\_EVERY\_TRANSFER
  - MCF5441x (DSPI), [1453](#)
  - MIMX10xx (DSPI), [1463](#)
  - SAME70 (DSPI), [1469](#)
- DEASSERT\_NEVER
  - MCF5441x (DSPI), [1453](#)
  - MIMX10xx (DSPI), [1463](#)
  - SAME70 (DSPI), [1469](#)
- debugMode
  - \_qtmr\_config, [1939](#)
- DEFAULT\_DSPI\_MODULE
  - MCF5441x (DSPI), [1451](#)
- delay\_compensation\_filter\_window\_length
  - mcanMODM7AE70::mcan\_config, [2158](#)
- delay\_compensation\_offset
  - mcanMODM7AE70::mcan\_config, [2158](#)
- DelayObject, [2049](#)
  - \_wake, [2051](#)
  - ~DelayObject, [2050](#)
  - DelayObject, [2050](#)
  - DelayUsec, [2051](#)
  - valid, [2052](#)
- DelayUsec
  - DelayObject, [2051](#)
- Delete\_Everything\_Restart
  - AcmeServletObject, [1941](#)
- DEPLETED\_PAUSE
  - SAI Configuration Enumerations, [1529](#)
- DEPLETED\_REPEAT\_LAST
  - SAI Configuration Enumerations, [1529](#)
- depletionBehavior
  - SAI\_rtx\_cfg\_t, [2302](#)
  - SSC\_rtx\_cfg\_t, [2345](#)
- Deprecated List, [1097](#)
- Description
  - TimeZoneRecord, [2363](#)
- destAddr
  - \_edma\_transfer\_config, [1910](#)
- destOffset
  - \_edma\_transfer\_config, [1910](#)
- DestOptHasError
  - IPv6FrameProcessingStruct, [2131](#)
- destTransferSize
  - \_edma\_transfer\_config, [1910](#)
- Detached Config Object, [430](#)
- Dev Board Temperature Sensor and ADC, [776](#)
- DevFirmVer
  - NB::Error::Init, [1903](#)
- DevHwVer
  - NB::Error::Init, [1903](#)
- Device Discovery and Configuration, [23](#)
- Device Interrupts, [1186](#)
  - \_DeviceInterruptTrigger, [1186](#)
  - DeviceAckInterrupt, [1187](#)
  - DeviceDisableInterrupt, [1187](#)
  - DeviceEnableInterrupt, [1187](#)
  - DeviceGetI2cSem, [1187](#)
  - DeviceGetQspiSem, [1188](#)
  - DevicesValid, [1188](#)
  - DeviceSetupI2c, [1188](#)
  - InterruptTriggerBothEdge, [1187](#)
  - InterruptTriggerFallingEdge, [1187](#)
  - InterruptTriggerNone, [1187](#)
  - InterruptTriggerRisingEdge, [1187](#)
  - InterruptTriggerSensitive, [1187](#)
- DeviceAckInterrupt
  - Device Interrupts, [1187](#)
- DeviceDisableInterrupt

- Device Interrupts, 1187
- DeviceEnableInterrupt
  - Device Interrupts, 1187
- DeviceGetI2cSem
  - Device Interrupts, 1187
- DeviceGetQspiSem
  - Device Interrupts, 1188
- DevicesValid
  - Device Interrupts, 1188
- deviceModeArg
  - \_FlexSPIConfig, 1921
- deviceModeCfgEnable
  - \_FlexSPIConfig, 1921
- deviceModeSeq
  - \_FlexSPIConfig, 1921
- deviceModeType
  - \_FlexSPIConfig, 1921
- DeviceSetupIsr
  - Device Interrupts, 1188
- deviceType
  - \_FlexSPIConfig, 1921
- DHCP, 440, 1175, 1893
  - ARP\_CONFLICT, 1894
  - LEASE\_OFFERED, 1894
  - LEASE\_OPEN, 1894
  - LEASE\_STATIC, 1894
  - LEASE\_TAKEN, 1894
  - LeaseState, 1894
- DHCP Client - Change IP Address, 440
- DHCP Client - Change IP Address Via Webpage, 448
- DHCP Client - Test DHCP, 443
- DHCP Server, 1176
  - AddStandardDHCPsServer, 1176
- DHCP Server (DHCPD), 446
- DHCP State, 1178
- DHCP::BlockAllocator, 1948
- DHCP::DhcpInfo, 2053
- DHCP::DhcpLeaseData, 2053
- DHCP::DhcpLeaseRequest, 2054
- DHCP::LeaseAllocator, 2154
- DHCP::MacPrefixAllocator, 2156
- DHCP::Server, 2318
- DHCP::SingleAllocator, 2319
- DhcpObject, 2054
  - bDoFallback, 2055
  - GetDhcpExpirationTime, 2055
  - GetDhcpRebindTime, 2055
  - GetDhcpRenewTime, 2055
  - GetDHCPState, 2056
  - GetRemainingDhcpLeaseTime, 2056
  - RebindDHCP, 2056
  - RenewDHCP, 2056
  - RestartDHCP, 2056
  - StartDHCP, 2057
  - StopDHCP, 2057
  - ValidDhcpLease, 2057
- DHCPv4 Client, 1177
  - GetInterfaceDHCPState, 1177
  - WaitForDHCPInterface, 1177
- DHCPv6, 528
- DHCPv6 Client, 1178
  - AddStaticIPv6Address, 1179
  - InitIPv6, 1179
  - RemoveStaticIPv6Address, 1179
  - ShowAllIPv6Info, 1179
  - StartDHCPv6, 1180
  - StartDHCPv6\_InfoReq, 1180
  - StartDHCPv6\_Solicit, 1180
- DiagCallBack, 2057
  - DiagCallBack, 2058
- diagcallback
  - Diagnostics (DIAG), 1189
- DiagItemClass, 2058
- Diagnostics (DIAG), 1189
  - diagcallback, 1189
  - InitPlatformDiag, 1190
  - SendDiagReport, 1190
- DiagPtrMon, 2058
  - DiagPtrMon, 2059
- DiagStrMon, 2060
  - DiagStrMon, 2060
- DiagVarMon, 2061
  - DiagVarMon, 2061
- Digital to Analog (DAC), 624
- direction
  - \_lpspi\_master\_config, 1923
  - \_lpspi\_slave\_config, 1930
- Disable System Features, 402
- DisableDMA
  - DSPIModule, 2066
- DisableMulticast
  - Network Interfaces, 1340
- DisablePHY
  - Ethernet, 1208
- disableSelfReception
  - \_flexcan\_config, 1912
- Disconnect
  - MQTT::Client, 1967
- Display HTML Files (FTPD), 493
- DLAST\_SGA
  - \_edma\_tcd, 1909
- DNS - Domain Name System, 1180
  - AnyDNSInterFaceActive, 1181
  - CheckDnsCache, 1181
  - fd\_dns\_part1, 1182
  - fd\_dns\_processresult, 1182
  - fd\_outstanding\_Responses, 1183
  - GetHostByName, 1183
  - GetHostByNameViaIfNum, 1184
  - IsNameIPAddress, 1185
- DNS Client, 451
- DNS Device Name, 437
- DNS Record Types, 1185
- DNS Return Codes, 1185
- DoActualClientRequest
  - Web Client, 1806

- Document Index, [753](#)
- DOFF
  - \_edma\_tcd, [1909](#)
- DoGet
  - Web Client, [1806–1810](#)
- DoGetEx
  - Web Client, [1810](#), [1811](#), [1813–1815](#)
- DoGetUpdate
  - Web Client, [1816](#), [1817](#)
- DoHttp
  - Web Client, [1817–1821](#)
- DoHttpEx
  - Web Client, [1822–1826](#)
- DoJsonHttp
  - Web Client, [1827–1834](#)
- DoJsonHttpFile
  - Web Client, [1835](#), [1836](#), [1838](#)
- DoJsonPost
  - Web Client, [1839–1845](#)
- DoJsonPostHttpFile
  - Web Client, [1846–1848](#)
- DoMultipartBoundary
  - Web Client, [1849](#)
- DoMultipartFinished
  - Web Client, [1850](#), [1851](#)
- DoMultipartItem
  - Web Client, [1851](#)
- DoMultipartStartHttp
  - Web Client, [1852](#), [1853](#)
- DoMultipartStartPost
  - Web Client, [1854](#), [1855](#)
- Done
  - DSPIModule, [2066](#)
  - SPIModule, [2335](#)
- DoTransaction
  - I2C, [2089](#)
- DoUrlEncodedFormHttp
  - Web Client, [1856–1858](#)
- DoUrlEncodedFormPost
  - Web Client, [1859–1861](#)
- drk
  - aes\_context, [1942](#)
- DSPI Multiplex, [820](#)
- DSPI to Serial, [629](#)
- DSPI\_BUSY
  - MCF5441x (DSPI), [1451](#)
- DSPI\_ERROR
  - MCF5441x (DSPI), [1451](#)
- DSPI\_MODULE\_COUNT
  - MCF5441x (DSPI), [1451](#)
- DSPI\_OK
  - MCF5441x (DSPI), [1451](#)
- dspiDMAStruct, [2062](#)
- DSPIdone
  - MCF5441x (DSPI), [1454](#)
  - MIMX10xx (DSPI), [1464](#)
  - SAME70 (DSPI), [1470](#)
- dspiDriverStruct, [2063](#)
- DSPInit
  - DSPIModule, [2071](#)
  - MCF5441x (DSPI), [1454](#)
  - MIMX10xx (DSPI), [1464](#), [1465](#)
  - SAME70 (DSPI), [1470](#)
- DSPIModule, [2064](#)
  - ClrSem, [2066](#)
  - DisableDMA, [2066](#)
  - Done, [2066](#)
  - DSPInit, [2071](#)
  - DSPIModule, [2065](#)
  - EnableDMA, [2066](#)
  - GetActualBaudrate, [2068](#)
  - GetSem, [2068](#)
  - Init, [2068](#)
  - RegisterSem, [2069](#)
  - Rx, [2069](#)
  - SetCS, [2069](#)
  - Start, [2070](#)
  - Tx, [2070](#)
- DspiModuleNumber, [1467](#)
- DSPISart
  - MCF5441x (DSPI), [1455](#)
  - MIMX10xx (DSPI), [1466](#)
  - SAME70 (DSPI), [1471](#)
- DspiState, [1467](#)
- Dual TCP To Serial, [795](#)
- dump
  - SAICtx\_t, [2306](#)
- DumpSock
  - NB::WebSocket, [2439](#)
- DumpSockets
  - NB::WebSocket, [2442](#)
- DumpTcpDebug
  - TCP, [1623](#)
- Dynamic\_IPADDR, [2072](#)
  - AddToJsonOutObject, [2073](#)
  - CommitTestedValue, [2073](#)
  - Dynamic\_IPADDR, [2073](#)
  - ExtendedSchema, [2073](#)
  - GetTextValue, [2073](#)
  - GetTypeValue, [2073](#)
  - TestNewValue, [2073](#)
- E\_BAD\_EXPIRATION\_VALUE
  - SSL/TLS, [1489](#)
- E\_CERT\_EXPIRED
  - SSL/TLS, [1489](#)
- E\_CERT\_VALID
  - SSL/TLS, [1489](#)
- E\_NO\_SYSTEM\_TIME
  - SSL/TLS, [1489](#)
- eAnswering
  - PPP Connection State, [1367](#)
- Earthquake, [1053](#)
- Easy SSL/TLS Onboard Certificates, [839](#)
- eAuthErrorAuthCheckFailed
  - Authorization Responses, [1778](#)
- eAuthErrorAuthTypeMismatch

- Authorization Responses, 1778
- eAuthErrorFailedRecordUpdate
  - Authorization Responses, 1778
- eAuthErrorNoEmptyUserAuthRecords
  - Authorization Responses, 1778
- eAuthErrorSaveFailed
  - Authorization Responses, 1778
- eAuthErrorUnableToAddUser
  - Authorization Responses, 1778
- eAuthErrorUnableToCreateHash
  - Authorization Responses, 1778
- eAuthErrorUserDoesNotExist
  - Authorization Responses, 1778
- eAuthErrorUserExists
  - Authorization Responses, 1778
- eAuthSuccess
  - Authorization Responses, 1778
- eAuthTypeDefault
  - Authorization Types, 1779
- eAuthTypeKey
  - Authorization Types, 1779
- eAuthTypePassword
  - Authorization Types, 1779
- EBI - External Bus Interface, 1385
  - busWidth, 1401
  - byteAccess, 1401
  - ConfigureEBI\_CS, 1391, 1393
  - ConfigureEBI\_CSPin, 1397
  - ConfigureEBI\_NRDPin, 1398
  - ConfigureEBI\_NWRPin, 1400
  - EBI\_BUS\_WIDTH\_16, 1387
  - EBI\_BUS\_WIDTH\_8, 1387
  - EBI\_BYTE\_ACCESS\_SELECT, 1388
  - EBI\_BYTE\_ACCESS\_WRITE, 1388
  - EBI\_CS\_BusWidth\_t, 1387
  - EBI\_CS\_ByteAccess\_t, 1387
  - EBI\_CS\_NWait\_t, 1388
  - EBI\_CS\_RdMode\_t, 1389
  - EBI\_CS\_WrMode\_t, 1390
  - EBI\_NWAIT\_DISABLED, 1389
  - EBI\_NWAIT\_FROZEN, 1389
  - EBI\_NWAIT\_READY, 1389
  - EBI\_READ\_MODE\_NCS, 1390
  - EBI\_READ\_MODE\_NRD, 1390
  - EBI\_WRITE\_MODE\_NCS, 1391
  - EBI\_WRITE\_MODE\_NWE, 1391
  - ncs\_rd\_pulse, 1401
  - ncs\_rd\_setup, 1401
  - ncs\_wr\_pulse, 1401
  - ncs\_wr\_setup, 1401
  - nrd\_cycles, 1402
  - nrd\_pulse, 1402
  - nrd\_setup, 1402
  - nWait, 1402
  - nwe\_cycles, 1402
  - nwe\_pulse, 1403
  - nwe\_setup, 1403
  - rdMode, 1403
  - tdf\_cycles, 1403
  - wrMode, 1403
- EBI Page Control, 721
- EBI Page Control - Simple, 721
- EBI\_BUS\_WIDTH\_16
  - EBI - External Bus Interface, 1387
- EBI\_BUS\_WIDTH\_8
  - EBI - External Bus Interface, 1387
- EBI\_BYTE\_ACCESS\_SELECT
  - EBI - External Bus Interface, 1388
- EBI\_BYTE\_ACCESS\_WRITE
  - EBI - External Bus Interface, 1388
- EBI\_CS\_BusWidth\_t
  - EBI - External Bus Interface, 1387
- EBI\_CS\_ByteAccess\_t
  - EBI - External Bus Interface, 1387
- EBI\_CS\_cfg\_t, 2074
- EBI\_CS\_NWait\_t
  - EBI - External Bus Interface, 1388
- EBI\_CS\_RdMode\_t
  - EBI - External Bus Interface, 1389
- EBI\_CS\_WrMode\_t
  - EBI - External Bus Interface, 1390
- EBI\_NWAIT\_DISABLED
  - EBI - External Bus Interface, 1389
- EBI\_NWAIT\_FROZEN
  - EBI - External Bus Interface, 1389
- EBI\_NWAIT\_READY
  - EBI - External Bus Interface, 1389
- EBI\_READ\_MODE\_NCS
  - EBI - External Bus Interface, 1390
- EBI\_READ\_MODE\_NRD
  - EBI - External Bus Interface, 1390
- EBI\_WRITE\_MODE\_NCS
  - EBI - External Bus Interface, 1391
- EBI\_WRITE\_MODE\_NWE
  - EBI - External Bus Interface, 1391
- eCHAPAuthenticate
  - PPP Connection State, 1367
- eClosed
  - PPP Connection State, 1367
- eClosing
  - PPP Connection State, 1367
- eClosingNow
  - IOSYS - I/O System, 1283
- eConnectionStatus
  - MQTT::Client, 1963
- eConnStat\_Error
  - MQTT::Client, 1963
- eConnStat\_Established
  - MQTT::Client, 1963
- eConnStat\_InProgress
  - MQTT::Client, 1963
- eConnStat\_No\_Broker\_Configured
  - MQTT::Client, 1963
- eConnStat\_Refused
  - MQTT::Client, 1963
- eConnStat\_Resolving

- MQTT::Client, [1963](#)
- eDelay
  - IPv6 Neighbor Discovery, [1326](#)
- edge\_filtering
  - mcanMODM7AE70::mcan\_config, [2158](#)
- eDHCP
  - IPv6 Routing, [1327](#)
- eDialing
  - PPP Connection State, [1367](#)
- edmaRxRegToRxDataHandle
  - \_lpspi\_master\_edma\_handle, [1925](#)
  - \_lpspi\_slave\_edma\_handle, [1932](#)
- edmaTxDataToTxRegHandle
  - \_lpspi\_master\_edma\_handle, [1925](#)
  - \_lpspi\_slave\_edma\_handle, [1932](#)
- eDoingND
  - IPv6 Routing, [1327](#)
- eErrorSet
  - IOSYS - I/O System, [1283](#)
- EFFS - Embedded Flash File System, [1191](#)
- EFFS Multiple Partitions, [734](#)
- EFFS Programming Guide, [184](#)
- EFFS-STD Flash File System, [1191](#)
  - fs\_chdir, [1192](#)
  - fs\_close, [1192](#)
  - fs\_delete, [1193](#)
  - fs\_eof, [1193](#)
  - fs\_findfirst, [1193](#)
  - fs\_findnext, [1194](#)
  - fs\_getfreespace, [1194](#)
  - fs\_gettimedate, [1195](#)
  - fs\_mkdir, [1195](#)
  - fs\_open, [1195](#)
  - fs\_read, [1196](#)
  - fs\_rewind, [1196](#)
  - fs\_rmdir, [1197](#)
  - fs\_seek, [1197](#)
  - fs\_settimedate, [1198](#)
  - fs\_write, [1198](#)
- EFFS-STD HTTP, [467](#)
- Effs\_Cfg, [2078](#)
- eIncomplete
  - IPv6 Neighbor Discovery, [1326](#)
- eInitializingModem
  - PPP Connection State, [1367](#)
- eInvalid
  - IPv6 Routing, [1327](#)
- eLCPNegotiate
  - PPP Connection State, [1367](#)
- eLimit\_InFlight\_Rx
  - MQTT::Client, [1964](#)
- eLimit\_InFlight\_Tx
  - MQTT::Client, [1963](#)
- eLimit\_PublishAliases
  - MQTT::Client, [1964](#)
- eLimit\_QoS
  - MQTT::Client, [1964](#)
- eLimit\_SessionExpiry
  - MQTT::Client, [1964](#)
- eLimit\_Size\_Rx
  - MQTT::Client, [1963](#)
- eLimit\_Size\_Tx
  - MQTT::Client, [1963](#)
- eLimit\_SubAliases
  - MQTT::Client, [1964](#)
- eLimit\_Subscriptions
  - MQTT::Client, [1964](#)
- eLinkLocal
  - IPv6 Routing, [1327](#)
- eListenOnSsh
  - Command Processor Listen Channels, [1168](#)
- eListenOnTcp
  - Command Processor Listen Channels, [1168](#)
- Embedded Flash File System (EFFS), [454](#)
- EMI Emissions Reduction, [648](#)
- Empty
  - fifo\_buffer\_storage, [2079](#)
- empty
  - NBString, [2241](#)
- EmptyResponse
  - HTTP and HTML Functions, [1262](#)
- eMY\_ADDR\_STATE
  - IPv6 Routing, [1326](#)
- eMY\_NEIGHBOR\_STATE
  - IPv6 Neighbor Discovery, [1326](#)
- enable
  - SAI\_rtx\_cfg\_t, [2302](#)
  - SSC\_rtx\_cfg\_t, [2345](#)
- enableChannelPreemption
  - \_edma\_channel\_Preemption\_config, [1905](#)
- EnableConfigMirror
  - Configuration Server, [1170](#)
- enableContinuousLinkMode
  - \_edma\_config, [1906](#)
- enableDebugMode
  - \_edma\_config, [1906](#)
- enableDestMinorOffset
  - \_edma\_minor\_offset\_config, [1908](#)
- EnableDMA
  - DSPIModule, [2066](#)
- enableExternalForce
  - \_qtmr\_config, [1939](#)
- enableHaltOnError
  - \_edma\_config, [1906](#)
- enableIndividMask
  - \_flexcan\_config, [1912](#)
- enableInputDelay
  - \_lpspi\_master\_config, [1924](#)
- enableListenOnlyMode
  - \_flexcan\_config, [1912](#)
- enableLoopBack
  - \_flexcan\_config, [1912](#)
- enableMasterMode
  - \_qtmr\_config, [1939](#)
- EnableMulticast
  - InterfaceBlock, [2100](#)



- Network Interfaces, [1340](#)
- EnableOnboardCertificateCreation
  - Onboard Self-Signed Certificate Generation, [1512](#)
- EnablePHY
  - Ethernet, [1209](#)
- enablePPSCapture
  - IEEE1588Timer, [2096](#)
- enablePreemptAbility
  - \_edma\_channel\_Preemption\_config, [1905](#)
- EnableRemoteConsole
  - HTTP and HTML Functions, [1263](#)
- enableRoundRobinArbitration
  - \_edma\_config, [1906](#)
- enableRunInDebug
  - \_pit\_config, [1939](#)
- EnableSecureConfigServer
  - Initialization - System Initialization Functions, [1328](#)
- enableSelfWakeup
  - \_flexcan\_config, [1912](#)
- enableSrcMinorOffset
  - \_edma\_minor\_offset\_config, [1908](#)
- enableSupervisorMode
  - \_flexcan\_config, [1912](#)
- EnableSystemDiagnostics
  - Initialization - System Initialization Functions, [1328](#)
- EnableTcpDebug
  - TCP, [1624](#)
- enableTimerSync
  - \_flexcan\_config, [1912](#)
- eNCPNegotiate
  - PPP Connection State, [1367](#)
- END\_ARRAY
  - JSON Lexer, [1331](#)
- END\_OBJECT
  - JSON Lexer, [1331](#)
- endTransmission
  - TwoWire, [2367](#)
  - WireIntf, [2526](#)
- eNoRoute
  - IPv6 Routing, [1327](#)
- enum\_PPPState
  - PPP Connection State, [1367](#)
- eObj\_Flag\_DynTopicName
  - MQTT, [1898](#)
- eObj\_Flag\_PubOnWrite
  - MQTT, [1898](#)
- eObj\_Flag\_PubQoS\_AtLeastOnce
  - MQTT, [1898](#)
- eObj\_Flag\_PubQoS\_ExactlyOnce
  - MQTT, [1898](#)
- eObj\_Flag\_PubRetain
  - MQTT, [1898](#)
- eObj\_Flag\_Sub\_HandlerOnly
  - MQTT, [1898](#)
- eObj\_Flag\_Sub\_IncludeLocal
  - MQTT, [1898](#)
- eObj\_Flag\_Sub\_NoLocal
  - MQTT, [1898](#)
- eObj\_Flag\_Sub\_QoS\_AtLeastOnce
  - MQTT, [1898](#)
- eObj\_Flag\_Sub\_QoS\_ExactlyOnce
  - MQTT, [1898](#)
- eObj\_Flag\_Sub\_Retained\_IfNew
  - MQTT, [1898](#)
- eObj\_Flag\_Sub\_Retained\_Never
  - MQTT, [1898](#)
- eObj\_Flag\_Subscribe
  - MQTT, [1898](#)
- eObj\_Flag\_WaitForPub
  - MQTT, [1898](#)
- eObj\_Serialize\_BigEndian
  - MQTT, [1898](#)
- eObj\_Serialize\_CURRENT\_FORM
  - MQTT, [1899](#)
- eObj\_Serialize\_Disabled
  - MQTT, [1898](#)
- eObj\_Serialize\_JSON
  - MQTT, [1898](#)
- eObj\_Serialize\_JSON\_ValOnly
  - MQTT, [1898](#)
- eObj\_Serialize\_LittleEndian
  - MQTT, [1899](#)
- eObj\_Serialize\_MqttString
  - MQTT, [1898](#)
- eObj\_Serialize\_NativeEndian
  - MQTT, [1899](#)
- eObj\_Serialize\_String
  - MQTT, [1898](#)
- eObj\_Serialize\_t
  - MQTT, [1898](#)
- EOF\_EL
  - JSON Lexer, [1332](#)
- eOpen
  - PPP Connection State, [1367](#)
- ePAPAAuthenticate
  - PPP Connection State, [1367](#)
- eParityEven
  - Serial Interfaces, [1515](#)
- eParityMulti
  - Serial Interfaces, [1515](#)
- eParityMultiEven
  - Serial Interfaces, [1515](#)
- eParityMultiOdd
  - Serial Interfaces, [1515](#)
- eParityNone
  - Serial Interfaces, [1515](#)
- eParityOdd
  - Serial Interfaces, [1515](#)
- ePrefixSource
  - IPv6 Routing, [1327](#)
- eProbe
  - IPv6 Neighbor Discovery, [1326](#)
- eRateLimit\_Publish
  - MQTT::Client, [1963](#)
- eRateLimit\_Subscribe
  - MQTT::Client, [1963](#)



- eRateLimitAdj\_Add
  - MQTT::Client, [1963](#)
- eRateLimitAdj\_None
  - MQTT::Client, [1963](#)
- eRateLimitAdj\_Set
  - MQTT::Client, [1963](#)
- eRateLimitAdj\_Subtract
  - MQTT::Client, [1963](#)
- eRateLimitAdjustMethod
  - MQTT::Client, [1963](#)
- eRateLimitType
  - MQTT::Client, [1963](#)
- eReachable
  - IPv6 Neighbor Discovery, [1326](#)
- eReadSet
  - IOSYS - I/O System, [1282](#)
- eRender\_Hidden
  - MQTT::ConfigExposer, [2040](#)
- eRender\_ModifiedOnly
  - MQTT::ConfigExposer, [2040](#)
- eRender\_Pretty
  - MQTT::ConfigExposer, [2040](#)
- eRenderFlag\_t
  - MQTT::ConfigExposer, [2040](#)
- eResult\_Cont\_Auth
  - MQTT, [1900](#)
- eResult\_Error\_BadArg
  - MQTT, [1900](#)
- eResult\_Error\_BadProp
  - MQTT, [1900](#)
- eResult\_Error\_InvalidURI
  - MQTT, [1900](#)
- eResult\_Error\_Malformed
  - MQTT, [1900](#)
- eResult\_Error\_NetErr
  - MQTT, [1900](#)
- eResult\_Error\_NoneAvail
  - MQTT, [1900](#)
- eResult\_Error\_NoSuchConn
  - MQTT, [1900](#)
- eResult\_Error\_Timeout
  - MQTT, [1900](#)
- eResult\_Mqtt\_Admin
  - MQTT, [1899](#)
- eResult\_Mqtt\_BadAuthMethod
  - MQTT, [1899](#)
- eResult\_Mqtt\_Banned
  - MQTT, [1899](#)
- eResult\_Mqtt\_ClientIdInvalid
  - MQTT, [1899](#)
- eResult\_Mqtt\_ConnRate
  - MQTT, [1899](#)
- eResult\_Mqtt\_FilterInvalid
  - MQTT, [1899](#)
- eResult\_Mqtt\_Implementation
  - MQTT, [1899](#)
- eResult\_Mqtt\_KATimeout
  - MQTT, [1899](#)
- eResult\_Mqtt\_MalformedPkt
  - MQTT, [1899](#)
- eResult\_Mqtt\_MaxConnTime
  - MQTT, [1900](#)
- eResult\_Mqtt\_MsgRate
  - MQTT, [1899](#)
- eResult\_Mqtt\_NameInvalid
  - MQTT, [1899](#)
- eResult\_Mqtt\_NotAuthorized
  - MQTT, [1899](#)
- eResult\_Mqtt\_NotSupported\_QoS
  - MQTT, [1899](#)
- eResult\_Mqtt\_NotSupported\_Retain
  - MQTT, [1899](#)
- eResult\_Mqtt\_NotSupported\_SharedSubs
  - MQTT, [1899](#)
- eResult\_Mqtt\_NotSupported\_SubIds
  - MQTT, [1900](#)
- eResult\_Mqtt\_NotSupported\_WildSubs
  - MQTT, [1900](#)
- eResult\_Mqtt\_PayFmt\_Invalid
  - MQTT, [1899](#)
- eResult\_Mqtt\_PktId\_InUse
  - MQTT, [1899](#)
- eResult\_Mqtt\_PktId\_NotFound
  - MQTT, [1899](#)
- eResult\_Mqtt\_Protocol
  - MQTT, [1899](#)
- eResult\_Mqtt\_Quota
  - MQTT, [1899](#)
- eResult\_Mqtt\_RcvMax\_Exceeded
  - MQTT, [1899](#)
- eResult\_Mqtt\_ServerBusy
  - MQTT, [1899](#)
- eResult\_Mqtt\_ServerMoved
  - MQTT, [1899](#)
- eResult\_Mqtt\_ServerShutdown
  - MQTT, [1899](#)
- eResult\_Mqtt\_ServerUnavail
  - MQTT, [1899](#)
- eResult\_Mqtt\_SessTakeover
  - MQTT, [1899](#)
- eResult\_Mqtt\_TooLarge
  - MQTT, [1899](#)
- eResult\_Mqtt\_TopicAlias\_Invalid
  - MQTT, [1899](#)
- eResult\_Mqtt\_Transport\_Error
  - MQTT, [1900](#)
- eResult\_Mqtt\_Unspecified
  - MQTT, [1899](#)
- eResult\_Mqtt\_UnsupportedProtoVer
  - MQTT, [1899](#)
- eResult\_Mqtt\_UseOther
  - MQTT, [1899](#)
- eResult\_Mqtt\_UserPassword
  - MQTT, [1899](#)
- eResult\_Socket\_NeedDNS
  - MQTT, [1900](#)

- eResult\_Socket\_TryAgain
  - MQTT, [1900](#)
- eResult\_Success
  - MQTT, [1899](#)
- eResult\_t
  - MQTT, [1899](#)
- erk
  - aes\_context, [1942](#)
- eRouteOutResult
  - IPv6 Routing, [1327](#)
- eRouter
  - IPv6 Routing, [1327](#)
- errorCount
  - \_lpspi\_slave\_edma\_handle, [1932](#)
  - \_lpspi\_slave\_handle, [1934](#)
- eSent
  - IPv6 Routing, [1327](#)
- eSessionLimit
  - MQTT::Client, [1963](#)
- eSocksAdrTypeDomain
  - SOCKS, [1445](#)
- eSocksAdrTypeIpv4
  - SOCKS, [1445](#)
- eSocksAdrTypeIpv6
  - SOCKS, [1445](#)
- eSocksAdrTypeNone
  - SOCKS, [1445](#)
- eSocksAuthTypeGssApi
  - SOCKS, [1445](#)
- eSocksAuthTypeNoAuth
  - SOCKS, [1445](#)
- eSocksAuthTypeUnPw
  - SOCKS, [1445](#)
- eSocksClientCmdBind
  - SOCKS, [1445](#)
- eSocksClientCmdConnect
  - SOCKS, [1445](#)
- eSocksClientCmdUdpAssoc
  - SOCKS, [1445](#)
- eStale
  - IPv6 Neighbor Discovery, [1326](#)
- eStatic
  - IPv6 Routing, [1327](#)
- eTenative
  - IPv6 Routing, [1327](#)
- Ethernet, [470](#), [1207](#)
  - AddEthernetInterfaces, [1208](#)
  - DisablePHY, [1208](#)
  - EnablePHY, [1209](#)
  - ManualEthernetConfig, [1209](#)
  - NO\_AUTOMATIC\_2ND\_ETHERNET, [1208](#)
- Ethernet I/O Counters, [1210](#)
- eUnknown
  - IPv6 Routing, [1327](#)
- eValid\_Deprecated
  - IPv6 Routing, [1327](#)
- eValid\_Preferred
  - IPv6 Routing, [1327](#)
- eWait4Ring
  - PPP Connection State, [1367](#)
- eWaitForTrain
  - PPP Connection State, [1367](#)
- eWriteSet
  - IOSYS - I/O System, [1283](#)
- Example Applications, [371](#)
- Exception Try/Catch, [473](#)
- ExpireRetCodes
  - SSL/TLS, [1489](#)
- extended\_id\_mask
  - mcanMODM7AE70::mcan\_config, [2158](#)
- ExtendedSchema
  - CUR\_IPADDR4, [2047](#)
  - Dynamic\_IPADDR, [2073](#)
- External IRQ, [474](#)
- External NMI IRQ 7, [650](#)
- External Pin IRQ, [722](#)
- External Real Time Clock, [734](#)
- Extra FD Circular Buffer, [477](#), [480](#)
- Extra File Descriptors, [1211](#)
  - FreeExtraFd, [1211](#)
  - GetExtraData, [1211](#)
  - GetExtraFD, [1212](#)
  - GetFreeExtraFDCount, [1212](#)
  - GetFreeSocketCount, [1212](#)
  - GetIoExpandStruct, [1212](#)
- Extract4
  - IPADDR6, [2107](#)
- f\_chdir
  - FAT File System, [1200](#)
- f\_close
  - FAT File System, [1200](#)
- f\_delete
  - FAT File System, [1200](#)
- f\_delvolume
  - FAT File System, [1201](#)
- f\_enterFS
  - FAT File System, [1206](#)
- f\_eof
  - FAT File System, [1201](#)
- f\_findfirst
  - FAT File System, [1201](#)
- f\_findnext
  - FAT File System, [1202](#)
- f\_getfreespace
  - FAT File System, [1202](#)
- f\_gettimedate
  - FAT File System, [1202](#)
- f\_mkdir
  - FAT File System, [1203](#)
- f\_open
  - FAT File System, [1203](#)
- f\_read
  - FAT File System, [1204](#)
- f\_releaseFS
  - FAT File System, [1207](#)
- f\_rewind

- FAT File System, [1204](#)
- f\_rmdir
  - FAT File System, [1205](#)
- f\_seek
  - FAT File System, [1205](#)
- f\_settimedate
  - FAT File System, [1205](#)
- f\_write
  - FAT File System, [1206](#)
- Factory Application, [722](#)
- FALSE\_EL
  - JSON Lexer, [1332](#)
- FAT File System, [1199](#)
  - f\_chdir, [1200](#)
  - f\_close, [1200](#)
  - f\_delete, [1200](#)
  - f\_delvolume, [1201](#)
  - f\_enterFS, [1206](#)
  - f\_eof, [1201](#)
  - f\_findfirst, [1201](#)
  - f\_findnext, [1202](#)
  - f\_getfreespace, [1202](#)
  - f\_gettimedate, [1202](#)
  - f\_mkdir, [1203](#)
  - f\_open, [1203](#)
  - f\_read, [1204](#)
  - f\_releaseFS, [1207](#)
  - f\_rewind, [1204](#)
  - f\_rmdir, [1205](#)
  - f\_seek, [1205](#)
  - f\_settimedate, [1205](#)
  - f\_write, [1206](#)
- FAT File System (FFFS-FAT), [455](#)
- FAT File System Seek Codes, [1207](#)
- faultFilterCount
  - \_qtmr\_config, [1939](#)
- faultFilterPeriod
  - \_qtmr\_config, [1939](#)
- FD\_CLR
  - IOSYS - I/O System, [1284](#)
- FD\_CLRFROMSET
  - IOSYS - I/O System, [1284](#)
- FD\_COPY
  - IOSYS - I/O System, [1285](#)
- fd\_dns\_part1
  - DNS - Domain Name System, [1182](#)
- fd\_dns\_processresult
  - DNS - Domain Name System, [1182](#)
- FD\_ISSET
  - IOSYS - I/O System, [1285](#)
- fd\_outstanding\_Responses
  - DNS - Domain Name System, [1183](#)
- FD\_OVERLAP
  - IOSYS - I/O System, [1285](#)
- FD\_SET
  - IOSYS - I/O System, [1286](#)
- FD\_SETFROMSET
  - IOSYS - I/O System, [1286](#)
- FD\_ZERO
  - IOSYS - I/O System, [1286](#)
- FdAppend
  - NBInterpolatedString, [2236](#)
  - NBString, [2241](#)
- FDBuffer, [483](#)
- FDCallBack
  - IOSYS - I/O System, [1282](#)
- FDChangeType
  - IOSYS - I/O System, [1282](#)
- fdprintf
  - File Descriptor Printf - fdprintf, [1238](#)
- fdprint
  - IPADDR4, [2103](#)
  - IPADDR6, [2107](#)
- fdprintf
  - File Descriptor Printf - fdprintf, [1239](#)
- fdprintf - printf to file descriptor, [485](#)
- FdShowData
  - Utilities, [1791](#)
- fdShowMac
  - Utilities, [1792](#)
- FdShowRingData
  - Utilities, [1792](#)
- FdToNBString, [487](#)
- fifo\_buffer\_storage, [2078](#)
  - Empty, [2079](#)
  - fifo\_buffer\_storage, [2079](#)
  - Full, [2079](#)
  - Peek\_Begin, [2080](#)
  - Peek\_End, [2080](#)
  - PeekData, [2080](#)
  - ReadData, [2080](#)
  - Reset, [2080](#)
  - SpaceAval, [2081](#)
  - SpaceUsed, [2081](#)
  - WriteData, [2081](#)
- fifo\_buffer\_storage::PeekIterator, [2285](#)
- fifo\_buffer\_storage::ReadBufPtr, [2299](#)
- fifo\_buffer\_storage::WriteBufPtr, [2533](#)
- fifoSize
  - \_lpspi\_master\_edma\_handle, [1925](#)
  - \_lpspi\_master\_handle, [1928](#)
  - \_lpspi\_slave\_edma\_handle, [1932](#)
  - \_lpspi\_slave\_handle, [1934](#)
- File Descriptor Printf - fdprintf, [1235](#)
  - fdprintf, [1238](#)
  - fdprintf, [1239](#)
  - vfdprintf, [1241](#)
  - vfdprintf, [1243](#)
- File Descriptors, [196](#)
- FileDescriptor, [480](#)
- FillInAltNamesASN1
  - Onboard Self-Signed Certificate Generation, [1512](#)
- Find
  - IPV6\_ROOT\_EL\_CONTAINER, [2126](#)
- find
  - NBString, [2241](#), [2242](#)

- Find My IP, [1055](#)
- Find My IP Task, [1056](#)
- FindAndCopyFullNameString
  - ParsedJsonDataSet, [2262](#)
- FindAndCopyGlobalString
  - ParsedJsonDataSet, [2264](#)
- FindAndCopyString
  - ParsedJsonDataSet, [2264](#), [2265](#)
- FindBoolean
  - ParsedJsonDataSet, [2265](#)
- FindBooleanInCurentObject
  - ParsedJsonDataSet, [2265](#)
- FindChild
  - MQTT::mqtt\_obj, [2214](#)
- FindElementAfterName
  - ParsedJsonDataSet, [2266](#)
- FindElementAfterNameInCurrentArray
  - ParsedJsonDataSet, [2266](#)
- FindElementAfterNameInCurrentObject
  - ParsedJsonDataSet, [2266](#)
- FindFullAtName
  - ParsedJsonDataSet, [2267](#)
- FindFullName
  - ParsedJsonDataSet, [2267](#)
- FindFullNameBoolean
  - ParsedJsonDataSet, [2267](#)
- FindFullNameNumber
  - ParsedJsonDataSet, [2268](#)
- FindFullNamePermissiveBoolean
  - ParsedJsonDataSet, [2268](#)
- FindFullNameString
  - ParsedJsonDataSet, [2268](#)
- FindGlobalBoolean
  - ParsedJsonDataSet, [2268](#)
- FindGlobalElementAfterName
  - ParsedJsonDataSet, [2269](#)
- FindGlobalNumber
  - ParsedJsonDataSet, [2269](#)
- FindGlobalObject
  - ParsedJsonDataSet, [2269](#)
- FindGlobalPermissiveBoolean
  - ParsedJsonDataSet, [2270](#)
- FindGlobalString
  - ParsedJsonDataSet, [2270](#)
- FindHandler
  - HtmlPageHandler, [2083](#)
- FindIA\_Addr
  - IPV6\_DHCPD, [2115](#)
- FindNumber
  - ParsedJsonDataSet, [2270](#)
- FindNumberInCurentObject
  - ParsedJsonDataSet, [2271](#)
- FindObject
  - ParsedJsonDataSet, [2271](#)
- FindObjectInCurentObject
  - ParsedJsonDataSet, [2271](#)
- FindPermissiveBoolean
  - ParsedJsonDataSet, [2271](#)
- FindPermissiveBooleanInCurentObject
  - ParsedJsonDataSet, [2272](#)
- FindString
  - ParsedJsonDataSet, [2272](#)
- FindStringInCurentObject
  - ParsedJsonDataSet, [2272](#)
- First
  - IPV6\_ROOT\_EL\_CONTAINER, [2127](#)
- FixHeaderAndSend
  - IP, [1304](#)
- FixHeaderAndSendViaInterface
  - IP, [1305](#)
- flags
  - \_edma\_handle, [1907](#)
- Flash Form, [995](#)
- FlashErase
  - HAL - Hardware Abstraction Layer, [1248](#)
- FlashProgram
  - HAL - Hardware Abstraction Layer, [1248](#)
- FlashProgramApplImage
  - HAL - Hardware Abstraction Layer, [1249](#)
- FlexCan Interrupt Transfer, [726](#)
- FlexCan Loopback, [728](#)
- FlexCAN SOMRT1061, [1127](#)
  - \_flexcan\_clock\_source, [1138](#)
  - \_flexcan\_error\_flags, [1139](#)
  - \_flexcan\_flags, [1139](#)
  - \_flexcan\_frame\_format, [1139](#)
  - \_flexcan\_frame\_type, [1140](#)
  - \_flexcan\_interrupt\_enable, [1140](#)
  - \_flexcan\_rx\_fifo\_filter\_type, [1140](#)
  - \_flexcan\_rx\_fifo\_priority, [1140](#)
  - \_flexcan\_wake\_up\_source, [1141](#)
- FLEXCAN\_CalculateImprovedTimingValues, [1141](#)
- FLEXCAN\_CALLBACK, [1132](#)
- flexcan\_clock\_source\_t, [1137](#)
- flexcan\_config\_t, [1137](#)
- FLEXCAN\_Deinit, [1141](#)
- FLEXCAN\_EnterFreezeMode, [1143](#)
- FLEXCAN\_ExitFreezeMode, [1143](#)
- FLEXCAN\_GetDefaultConfig, [1143](#)
- FLEXCAN\_GetInstance, [1143](#)
- FLEXCAN\_GetTimeStamp, [1144](#)
- FLEXCAN\_ID\_EXT, [1132](#)
- FLEXCAN\_ID\_STD, [1132](#)
- FLEXCAN\_Init, [1144](#)
- FLEXCAN\_ReadRxFifo, [1144](#)
- FLEXCAN\_ReadRxMb, [1145](#)
- FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_A, [1132](#)
- FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_HIGH, [1132](#)
- FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_LOW, [1132](#)
- FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_HIGH, [1133](#)
- FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_LOW, [1133](#)

FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_HIGH, [FLEXCAN\\_TransferAbortSend, 1149](#)  
[1133](#)  
 FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_LOW, [FLEXCAN\\_TransferCreateHandle, 1149](#)  
[1133](#)  
 FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A, [FLEXCAN\\_TransferGetReceiveFifoCount, 1149](#)  
[1133](#)  
 FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_HIGH, [FLEXCAN\\_TransferHandleIRQ, 1150](#)  
[1133](#)  
 FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_LOW, [FLEXCAN\\_TransferReceiveBlocking, 1150](#)  
[1134](#)  
 FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_HIGH, [FLEXCAN\\_TransferReceiveFifoBlocking, 1150](#)  
[1134](#)  
 FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW, [FLEXCAN\\_TransferReceiveFifoNonBlocking, 1151](#)  
[1134](#)  
 FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_HIGH, [FLEXCAN\\_TransferReceiveNonBlocking, 1151](#)  
[1134](#)  
 FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_LOW, [FLEXCAN\\_TransferSendBlocking, 1152](#)  
[1134](#)  
 flexcan\_rx\_fifo\_priority\_t, [FLEXCAN\\_TransferSendNonBlocking, 1152](#)  
[1137](#)  
 FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_A, [FLEXCAN\\_WriteTxMb, 1153](#)  
[1134](#)  
 FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_HIGH, [kFLEXCAN\\_AckError, 1139](#)  
[1134](#)  
 FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_LOW, [kFLEXCAN\\_Bit0Error, 1139](#)  
[1135](#)  
 FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_HIGH, [kFLEXCAN\\_Bit1Error, 1139](#)  
[1135](#)  
 FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_LOW, [kFLEXCAN\\_BusOffInterruptEnable, 1140](#)  
[1135](#)  
 FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_HIGH, [kFLEXCAN\\_BusOffIntFlag, 1139](#)  
[1135](#)  
 FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_LOW, [kFLEXCAN\\_ClkSrc0, 1139](#)  
[1135](#)  
 FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A, [kFLEXCAN\\_ClkSrc1, 1139](#)  
[1135](#)  
 FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_HIGH, [kFLEXCAN\\_ClkSrcOsc, 1139](#)  
[1136](#)  
 FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_LOW, [kFLEXCAN\\_ClkSrcPeri, 1139](#)  
[1136](#)  
 FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_HIGH, [kFLEXCAN\\_CrcError, 1139](#)  
[1136](#)  
 FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW, [kFLEXCAN\\_ErrorInterruptEnable, 1140](#)  
[1136](#)  
 FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_HIGH, [kFLEXCAN\\_ErrorIntFlag, 1139](#)  
[1136](#)  
 FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_LOW, [kFLEXCAN\\_FaultConfinementFlag, 1139](#)  
[1136](#)  
 flexcan\_rx\_mb\_config\_t, [kFLEXCAN\\_FormError, 1139](#)  
[1137](#)  
 FLEXCAN\_RX\_MB\_EXT\_MASK, [kFLEXCAN\\_FrameFormatExtend, 1140](#)  
[1137](#)  
 FLEXCAN\_RX\_MB\_STD\_MASK, [kFLEXCAN\\_FrameFormatStandard, 1140](#)  
[1137](#)  
 FLEXCAN\_SetBitRate, [kFLEXCAN\\_FrameTypeData, 1140](#)  
[1145](#)  
 FLEXCAN\_SetRxFifoConfig, [kFLEXCAN\\_FrameTypeRemote, 1140](#)  
[1146](#)  
 FLEXCAN\_SetRxFifoGlobalMask, [kFLEXCAN\\_IdleFlag, 1139](#)  
[1146](#)  
 FLEXCAN\_SetRxIndividualMask, [kFLEXCAN\\_ReceivingFlag, 1139](#)  
[1146](#)  
 FLEXCAN\_SetRxMbConfig, [kFLEXCAN\\_RxErrorWarningFlag, 1139](#)  
[1147](#)  
 FLEXCAN\_SetRxMbGlobalMask, [kFLEXCAN\\_RxFifoFilterTypeA, 1140](#)  
[1147](#)  
 FLEXCAN\_SetTimingConfig, [kFLEXCAN\\_RxFifoFilterTypeB, 1140](#)  
[1147](#)  
 FLEXCAN\_SetTxMbConfig, [kFLEXCAN\\_RxFifoFilterTypeC, 1140](#)  
[1148](#)  
 FLEXCAN\_TransferAbortReceive, [kFLEXCAN\\_RxFifoFilterTypeD, 1140](#)  
[1148](#)  
 FLEXCAN\_TransferAbortReceiveFifo, [kFLEXCAN\\_RxFifoFrameAvlFlag, 1138](#)  
[1148](#)  
 FLEXCAN\_TransferAbortSend, [kFLEXCAN\\_RxFifoOverflowFlag, 1138](#)  
[1149](#)  
 FLEXCAN\_TransferCreateHandle, [kFLEXCAN\\_RxFifoPrioHigh, 1141](#)  
[1149](#)  
 FLEXCAN\_TransferGetReceiveFifoCount, [kFLEXCAN\\_RxFifoPrioLow, 1141](#)  
[1149](#)  
 FLEXCAN\_TransferHandleIRQ, [kFLEXCAN\\_RxFifoWarningFlag, 1138](#)  
[1150](#)  
 FLEXCAN\_TransferReceiveBlocking, [kFLEXCAN\\_RxWarningInterruptEnable, 1140](#)  
[1150](#)  
 FLEXCAN\_TransferReceiveFifoBlocking, [kFLEXCAN\\_RxWarningIntFlag, 1139](#)  
[1150](#)  
 FLEXCAN\_TransferReceiveFifoNonBlocking, [kFLEXCAN\\_StuffingError, 1139](#)  
[1151](#)  
 FLEXCAN\_TransferReceiveNonBlocking, [kFLEXCAN\\_SynchFlag, 1139](#)  
[1151](#)  
 FLEXCAN\_TransferSendBlocking, [kFLEXCAN\\_TransmittingFlag, 1139](#)  
[1152](#)  
 FLEXCAN\_TransferSendNonBlocking, [kFLEXCAN\\_TxErrorWarningFlag, 1139](#)  
[1152](#)  
 FLEXCAN\_WriteTxMb, [kFLEXCAN\\_TxWarningInterruptEnable, 1140](#)  
[1153](#)  
 kFLEXCAN\_AckError, [kFLEXCAN\\_TxWarningIntFlag, 1139](#)  
[1139](#)  
 kFLEXCAN\_Bit0Error, [kFLEXCAN\\_WakeUpInterruptEnable, 1140](#)  
[1139](#)  
 kFLEXCAN\_Bit1Error, [kFLEXCAN\\_WakeUpIntFlag, 1139](#)  
[1139](#)  
 kFLEXCAN\_BusOffInterruptEnable, [kFLEXCAN\\_WakeupSrcFiltered, 1141](#)  
[1140](#)  
 kFLEXCAN\_BusOffIntFlag, [kFLEXCAN\\_WakeupSrcUnfiltered, 1141](#)  
[1139](#)  
 kFLEXCAN\_ClkSrc0, [kStatus\\_FLEXCAN\\_ErrorStatus, 1138](#)  
[1139](#)  
 kFLEXCAN\_ClkSrc1, [kStatus\\_FLEXCAN\\_RxBusy, 1138](#)  
[1139](#)  
 kFLEXCAN\_ClkSrcOsc, [kStatus\\_FLEXCAN\\_RxFifoBusy, 1138](#)  
[1139](#)  
 kFLEXCAN\_ClkSrcPeri, [kStatus\\_FLEXCAN\\_RxFifoDisabled, 1138](#)  
[1139](#)  
 kFLEXCAN\_CrcError, [kStatus\\_FLEXCAN\\_RxFifoIdle, 1138](#)  
[1139](#)  
 kFLEXCAN\_ErrorInterruptEnable, [1138](#)  
[1140](#)  
 kFLEXCAN\_ErrorIntFlag, [1138](#)  
[1139](#)  
 kFLEXCAN\_FaultConfinementFlag, [1138](#)  
[1139](#)  
 kFLEXCAN\_FormError, [1138](#)  
[1139](#)  
 kFLEXCAN\_FrameFormatExtend, [1138](#)  
[1140](#)  
 kFLEXCAN\_FrameFormatStandard, [1138](#)  
[1140](#)  
 kFLEXCAN\_FrameTypeData, [1138](#)  
[1140](#)  
 kFLEXCAN\_FrameTypeRemote, [1138](#)  
[1140](#)  
 kFLEXCAN\_IdleFlag, [1138](#)  
[1139](#)  
 kFLEXCAN\_ReceivingFlag, [1138](#)  
[1139](#)  
 kFLEXCAN\_RxErrorWarningFlag, [1138](#)  
[1139](#)  
 kFLEXCAN\_RxFifoFilterTypeA, [1138](#)  
[1140](#)  
 kFLEXCAN\_RxFifoFilterTypeB, [1138](#)  
[1140](#)  
 kFLEXCAN\_RxFifoFilterTypeC, [1138](#)  
[1140](#)  
 kFLEXCAN\_RxFifoFilterTypeD, [1138](#)  
[1140](#)  
 kFLEXCAN\_RxFifoFrameAvlFlag, [1138](#)  
[1138](#)  
 kFLEXCAN\_RxFifoOverflowFlag, [1138](#)  
[1138](#)  
 kFLEXCAN\_RxFifoPrioHigh, [1138](#)  
[1141](#)  
 kFLEXCAN\_RxFifoPrioLow, [1138](#)  
[1141](#)  
 kFLEXCAN\_RxFifoWarningFlag, [1138](#)  
[1138](#)  
 kFLEXCAN\_RxWarningInterruptEnable, [1138](#)  
[1140](#)  
 kFLEXCAN\_RxWarningIntFlag, [1138](#)  
[1139](#)  
 kFLEXCAN\_StuffingError, [1138](#)  
[1139](#)  
 kFLEXCAN\_SynchFlag, [1138](#)  
[1139](#)  
 kFLEXCAN\_TransmittingFlag, [1138](#)  
[1139](#)  
 kFLEXCAN\_TxErrorWarningFlag, [1138](#)  
[1139](#)  
 kFLEXCAN\_TxWarningInterruptEnable, [1138](#)  
[1140](#)  
 kFLEXCAN\_TxWarningIntFlag, [1138](#)  
[1139](#)  
 kFLEXCAN\_WakeUpInterruptEnable, [1138](#)  
[1140](#)  
 kFLEXCAN\_WakeUpIntFlag, [1138](#)  
[1139](#)  
 kFLEXCAN\_WakeupSrcFiltered, [1138](#)  
[1141](#)  
 kFLEXCAN\_WakeupSrcUnfiltered, [1138](#)  
[1141](#)  
 kStatus\_FLEXCAN\_ErrorStatus, [1138](#)  
[1138](#)  
 kStatus\_FLEXCAN\_RxBusy, [1138](#)  
[1138](#)  
 kStatus\_FLEXCAN\_RxFifoBusy, [1138](#)  
[1138](#)  
 kStatus\_FLEXCAN\_RxFifoDisabled, [1138](#)  
[1138](#)  
 kStatus\_FLEXCAN\_RxFifoIdle, [1138](#)  
[1138](#)

- kStatus\_FLEXCAN\_RxFifoOverflow, [1138](#)
- kStatus\_FLEXCAN\_RxFifoWarning, [1138](#)
- kStatus\_FLEXCAN\_RxIdle, [1138](#)
- kStatus\_FLEXCAN\_RxOverflow, [1138](#)
- kStatus\_FLEXCAN\_RxRemote, [1138](#)
- kStatus\_FLEXCAN\_TxBusy, [1138](#)
- kStatus\_FLEXCAN\_TxIdle, [1138](#)
- kStatus\_FLEXCAN\_TxSwitchToRx, [1138](#)
- kStatus\_FLEXCAN\_UnHandled, [1138](#)
- kStatus\_FLEXCAN\_WakeUp, [1138](#)
- FLEXCAN\_CalculateImprovedTimingValues
  - FlexCAN SOMRT1061, [1141](#)
- FLEXCAN\_CALLBACK
  - FlexCAN SOMRT1061, [1132](#)
- flexcan\_clock\_source\_t
  - FlexCAN SOMRT1061, [1137](#)
- flexcan\_config\_t
  - FlexCAN SOMRT1061, [1137](#)
- FLEXCAN\_Deinit
  - FlexCAN SOMRT1061, [1141](#)
- FLEXCAN\_EnterFreezeMode
  - FlexCAN SOMRT1061, [1143](#)
- FLEXCAN\_ExitFreezeMode
  - FlexCAN SOMRT1061, [1143](#)
- FLEXCAN\_GetDefaultConfig
  - FlexCAN SOMRT1061, [1143](#)
- FLEXCAN\_GetInstance
  - FlexCAN SOMRT1061, [1143](#)
- FLEXCAN\_GetTimeStamp
  - FlexCAN SOMRT1061, [1144](#)
- FLEXCAN\_ID\_EXT
  - FlexCAN SOMRT1061, [1132](#)
- FLEXCAN\_ID\_STD
  - FlexCAN SOMRT1061, [1132](#)
- FLEXCAN\_Init
  - FlexCAN SOMRT1061, [1144](#)
- FLEXCAN\_ReadRxFifo
  - FlexCAN SOMRT1061, [1144](#)
- FLEXCAN\_ReadRxMb
  - FlexCAN SOMRT1061, [1145](#)
- FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_A
  - FlexCAN SOMRT1061, [1132](#)
- FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_HIGH
  - FlexCAN SOMRT1061, [1132](#)
- FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_LOW
  - FlexCAN SOMRT1061, [1132](#)
- FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_HIGH
  - FlexCAN SOMRT1061, [1133](#)
- FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_LOW
  - FlexCAN SOMRT1061, [1133](#)
- FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_HIGH
  - FlexCAN SOMRT1061, [1133](#)
- FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_LOW
  - FlexCAN SOMRT1061, [1133](#)
- FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A
  - FlexCAN SOMRT1061, [1133](#)
- FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_HIGH
  - FlexCAN SOMRT1061, [1133](#)
- FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_LOW
  - FlexCAN SOMRT1061, [1133](#)
- FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_HIGH
  - FlexCAN SOMRT1061, [1134](#)
- FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW
  - FlexCAN SOMRT1061, [1134](#)
- FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_HIGH
  - FlexCAN SOMRT1061, [1134](#)
- FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_LOW
  - FlexCAN SOMRT1061, [1134](#)
- flexcan\_rx\_fifo\_priority\_t
  - FlexCAN SOMRT1061, [1137](#)
- FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_A
  - FlexCAN SOMRT1061, [1134](#)
- FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_HIGH
  - FlexCAN SOMRT1061, [1134](#)
- FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_LOW
  - FlexCAN SOMRT1061, [1135](#)
- FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_HIGH
  - FlexCAN SOMRT1061, [1135](#)
- FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_LOW
  - FlexCAN SOMRT1061, [1135](#)
- FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_HIGH
  - FlexCAN SOMRT1061, [1135](#)
- FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_LOW
  - FlexCAN SOMRT1061, [1135](#)
- FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A
  - FlexCAN SOMRT1061, [1135](#)
- FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_HIGH
  - FlexCAN SOMRT1061, [1136](#)
- FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_LOW
  - FlexCAN SOMRT1061, [1136](#)
- FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_HIGH
  - FlexCAN SOMRT1061, [1136](#)
- FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW
  - FlexCAN SOMRT1061, [1136](#)
- FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_HIGH
  - FlexCAN SOMRT1061, [1136](#)
- FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_LOW
  - FlexCAN SOMRT1061, [1136](#)
- flexcan\_rx\_mb\_config\_t
  - FlexCAN SOMRT1061, [1137](#)
- FLEXCAN\_RX\_MB\_EXT\_MASK
  - FlexCAN SOMRT1061, [1137](#)
- FLEXCAN\_RX\_MB\_STD\_MASK
  - FlexCAN SOMRT1061, [1137](#)
- FLEXCAN\_SetBitRate
  - FlexCAN SOMRT1061, [1145](#)
- FLEXCAN\_SetRxFifoConfig
  - FlexCAN SOMRT1061, [1146](#)
- FLEXCAN\_SetRxFifoGlobalMask
  - FlexCAN SOMRT1061, [1146](#)
- FLEXCAN\_SetRxIndividualMask
  - FlexCAN SOMRT1061, [1146](#)
- FLEXCAN\_SetRxMbConfig
  - FlexCAN SOMRT1061, [1147](#)
- FLEXCAN\_SetRxMbGlobalMask
  - FlexCAN SOMRT1061, [1147](#)



- FLEXCAN\_SetTimingConfig
  - FlexCAN SOMRT1061, [1147](#)
- FLEXCAN\_SetTxMbConfig
  - FlexCAN SOMRT1061, [1148](#)
- FLEXCAN\_TransferAbortReceive
  - FlexCAN SOMRT1061, [1148](#)
- FLEXCAN\_TransferAbortReceiveFifo
  - FlexCAN SOMRT1061, [1148](#)
- FLEXCAN\_TransferAbortSend
  - FlexCAN SOMRT1061, [1149](#)
- FLEXCAN\_TransferCreateHandle
  - FlexCAN SOMRT1061, [1149](#)
- FLEXCAN\_TransferGetReceiveFifoCount
  - FlexCAN SOMRT1061, [1149](#)
- FLEXCAN\_TransferHandleIRQ
  - FlexCAN SOMRT1061, [1150](#)
- FLEXCAN\_TransferReceiveBlocking
  - FlexCAN SOMRT1061, [1150](#)
- FLEXCAN\_TransferReceiveFifoBlocking
  - FlexCAN SOMRT1061, [1150](#)
- FLEXCAN\_TransferReceiveFifoNonBlocking
  - FlexCAN SOMRT1061, [1151](#)
- FLEXCAN\_TransferReceiveNonBlocking
  - FlexCAN SOMRT1061, [1151](#)
- FLEXCAN\_TransferSendBlocking
  - FlexCAN SOMRT1061, [1152](#)
- FLEXCAN\_TransferSendNonBlocking
  - FlexCAN SOMRT1061, [1152](#)
- FLEXCAN\_WriteTxMb
  - FlexCAN SOMRT1061, [1153](#)
- FlexPWM 3-Channel, [738](#)
- Flush
  - NB::WebSocket, [2445](#)
- flush
  - WireIntf, [2526](#)
- ForbiddenResponse
  - HTTP and HTML Functions, [1263](#)
- ForceReboot
  - HAL - Hardware Abstraction Layer, [1249](#)
- ForEachLeaf
  - MQTT::mqtt\_leaf, [2203](#)
- format
  - \_flexcan\_frame, [1914](#)
  - \_flexcan\_rx\_mb\_config, [1918](#)
- FragmentProcessFunction
  - IP, [1302](#)
- frame
  - \_flexcan\_fifo\_transfer, [1913](#)
  - \_flexcan\_mb\_transfer, [1916](#)
- FRAME\_SYNC\_FALLING
  - SAI Configuration Enumerations, [1531](#)
- FRAME\_SYNC\_INPUT
  - SAI Configuration Enumerations, [1531](#)
- FRAME\_SYNC\_NEGATIVE
  - SAI Configuration Enumerations, [1531](#)
- FRAME\_SYNC\_POSITIVE
  - SAI Configuration Enumerations, [1531](#)
- FRAME\_SYNC\_RISING
  - SAI Configuration Enumerations, [1531](#)
- frameEdge\_t
  - SAI Configuration Enumerations, [1531](#)
- frameNum
  - \_flexcan\_fifo\_transfer, [1913](#)
- frameSyncCh\_t
  - SAI Configuration Enumerations, [1531](#)
- frameSyncOut\_t
  - SAI Configuration Enumerations, [1531](#)
- FreeBuffer
  - Buffers - System Buffer Pool, [1123](#)
- FreeBufferList
  - Buffers - System Buffer Pool, [1124](#)
- FreeExtraFd
  - Extra File Descriptors, [1211](#)
- FreeSSLSockets
  - SSL/TLS, [1490](#)
- frequency
  - \_tempmon\_config, [1940](#)
- FS\_BulkStore, [399](#)
- fs\_chdir
  - EFFS-STD Flash File System, [1192](#)
- fs\_close
  - EFFS-STD Flash File System, [1192](#)
- fs\_delete
  - EFFS-STD Flash File System, [1193](#)
- fs\_eof
  - EFFS-STD Flash File System, [1193](#)
- fs\_findfirst
  - EFFS-STD Flash File System, [1193](#)
- fs\_findnext
  - EFFS-STD Flash File System, [1194](#)
- fs\_getfreespace
  - EFFS-STD Flash File System, [1194](#)
- fs\_gettimedate
  - EFFS-STD Flash File System, [1195](#)
- fs\_mkdir
  - EFFS-STD Flash File System, [1195](#)
- fs\_open
  - EFFS-STD Flash File System, [1195](#)
- fs\_read
  - EFFS-STD Flash File System, [1196](#)
- fs\_rewind
  - EFFS-STD Flash File System, [1196](#)
- fs\_rmdir
  - EFFS-STD Flash File System, [1197](#)
- fs\_seek
  - EFFS-STD Flash File System, [1197](#)
- fs\_settimedate
  - EFFS-STD Flash File System, [1198](#)
- fs\_write
  - EFFS-STD Flash File System, [1198](#)
- fShowArp
  - ARP - Address Resolution Protocol, [1107](#)
- FSL\_QTMR\_DRIVER\_VERSION
  - QTMR (Quad Timer), [1407](#)
- FTP, [458](#), [489](#)
- FTP Client, [490](#), [1214](#)

- FTP\_CloseSession, [1215](#)
- FTP\_InitializeSession, [1215](#)
- FTPActiveMode, [1216](#)
- FTPDeleteDir, [1216](#)
- FTPDeleteFile, [1216](#)
- FTPGetCommandResult, [1217](#)
- FTPGetDir, [1217](#)
- FTPGetFile, [1218](#)
- FTPGetFileNames, [1218](#)
- FTPGetList, [1219](#)
- FTPMakeDir, [1219](#)
- FTPPassiveMode, [1219](#)
- FTPRawCommand, [1220](#)
- FTPRawStreamCommand, [1220](#)
- FTPRenameFile, [1221](#)
- FTPSendFile, [1221](#)
- FTPSetDir, [1222](#)
- FTPUpDir, [1222](#)
- FTP Server, [1222](#)
- FTP Server Callback, [1223](#)
  - FTPD\_AbleToCreateFile, [1225](#)
  - FTPD\_CreateSubDirectory, [1225](#)
  - FTPD\_DeleteFile, [1226](#)
  - FTPD\_DeleteSubDirectory, [1226](#)
  - FTPD\_DirectoryExists, [1228](#)
  - FTPD\_FileExists, [1228](#)
  - FTPD\_GetFileFromClient, [1229](#)
  - FTPD\_GetFileSize, [1229](#)
  - FTPD\_ListFile, [1229](#)
  - FTPD\_ListSubDirectories, [1230](#)
  - FTPD\_Rename, [1231](#)
  - FTPD\_SendFileToClient, [1231](#)
  - FTPDCallBackReportFunct, [1225](#)
  - FTPDSessionEnd, [1232](#)
  - FTPDSessionStart, [1232](#)
  - FTPDStart, [1233](#)
  - FTPDStartSSL, [1233](#)
  - FTPDStopReq, [1233](#)
- FTP Server Core Return Codes, [1234](#)
- FTP Server File Return Codes, [1234](#)
- FTP Server with EDFS-FAT, [1234](#)
- FTP Server with EDFS-STD, [1235](#)
- FTP\_CloseSession
  - FTP Client, [1215](#)
- FTP\_InitializeSession
  - FTP Client, [1215](#)
- FTPActiveMode
  - FTP Client, [1216](#)
- FTPD\_AbleToCreateFile
  - FTP Server Callback, [1225](#)
- FTPD\_CreateSubDirectory
  - FTP Server Callback, [1225](#)
- FTPD\_DeleteFile
  - FTP Server Callback, [1226](#)
- FTPD\_DeleteSubDirectory
  - FTP Server Callback, [1226](#)
- FTPD\_DirectoryExists
  - FTP Server Callback, [1228](#)
- FTPD\_FileExists
  - FTP Server Callback, [1228](#)
- FTPD\_GetFileFromClient
  - FTP Server Callback, [1229](#)
- FTPD\_GetFileSize
  - FTP Server Callback, [1229](#)
- FTPD\_ListFile
  - FTP Server Callback, [1229](#)
- FTPD\_ListSubDirectories
  - FTP Server Callback, [1230](#)
- FTPD\_Rename
  - FTP Server Callback, [1231](#)
- FTPD\_SendFileToClient
  - FTP Server Callback, [1231](#)
- FTPDCallBackReportFunct
  - FTP Server Callback, [1225](#)
- FTPDeleteDir
  - FTP Client, [1216](#)
- FTPDeleteFile
  - FTP Client, [1216](#)
- FTPDSessionEnd
  - FTP Server Callback, [1232](#)
- FTPDSessionStart
  - FTP Server Callback, [1232](#)
- FTPDStart
  - FTP Server Callback, [1233](#)
- FTPDStartSSL
  - FTP Server Callback, [1233](#)
- FTPDStopReq
  - FTP Server Callback, [1233](#)
- FTPGetCommandResult
  - FTP Client, [1217](#)
- FTPGetDir
  - FTP Client, [1217](#)
- FTPGetFile
  - FTP Client, [1218](#)
- FTPGetFileNames
  - FTP Client, [1218](#)
- FTPGetList
  - FTP Client, [1219](#)
- FTPMakeDir
  - FTP Client, [1219](#)
- FTPPassiveMode
  - FTP Client, [1219](#)
- FTPRawCommand
  - FTP Client, [1220](#)
- FTPRawStreamCommand
  - FTP Client, [1220](#)
- FTPRenameFile
  - FTP Client, [1221](#)
- FTPS Server, [855](#)
- FTPSendFile
  - FTP Client, [1221](#)
- FTPSetDir
  - FTP Client, [1222](#)
- FTPUpDir
  - FTP Client, [1222](#)
- Full



- fifo\_buffer\_storage, [2079](#)
- function
  - PinIO, [2292](#)
- GDB Debugger, [499](#)
- GDB Stub (Remote Debugging), [1244](#)
  - InitGDBStub, [1245](#)
  - InitGDBStubNoBreak, [1245](#)
- General Purpose I/O, [501](#)
- General Purpose I/O Service, [503](#)
- GeneralErrors
  - NB::Error, [1901](#)
- Get Application Image and Load it From a Web Server, [555](#)
- Get JSON Object From Server, [541](#)
- Get MOD54417 Switch Port, [687](#)
- Get Non-JSON Object From Server, [544](#)
- Get\_msec
  - Utilities, [1792](#)
- GetActualBaudrate
  - DSPIModule, [2068](#)
  - SPIModule, [2336](#)
- GetAddr
  - ParsedURI, [2282](#)
- GetArpMacFromIp
  - ARP - Address Resolution Protocol, [1108](#)
- GetBoundaryMarker
  - HTTP\_Request, [2086](#)
- GetBuffer
  - Buffers - System Buffer Pool, [1124](#)
  - UniquelIdentifier, [2420](#)
- GetCertificateLen
  - SSL/TLS, [1490](#)
- GetCertificatePEM
  - SSL/TLS, [1490](#)
- getchar
  - IOSYS - I/O System, [1288](#)
- GetChildPrintSize
  - JsonRef, [2149](#)
- GetChildren
  - MQTT::mqtt\_leaf, [2203](#)
  - MQTT::mqtt\_obj, [2214](#)
- GetChoices
  - config\_chooser, [1981](#)
- GetClientCert
  - MQTT::Client, [1967](#)
- GetConnectionStatus
  - MQTT::Client, [1967](#)
- GetConnectionStatusString
  - MQTT, [1900](#)
- getConsecutiveValidSeconds
  - IEEE1588Timer, [2096](#)
- GetCount
  - buffer\_list, [1949](#)
- GetCurrent
  - ParsedJsonDataSet, [2273](#)
- GetCurrentChannelStatus
  - SerialRecord, [2316](#)
- getCurrentConfig
  - SAICtx\_t, [2306](#)
  - SSCCtx\_t, [2346](#)
- getCurrentTime
  - IEEE1588Timer, [2096](#)
- GetData
  - canMCF5441x::CanRxMessage, [1956](#)
  - mcanMODM7AE70::CanRxMessage, [1958](#)
- GetDataBuffer
  - UDPPacket, [2392](#)
- GetDataForCertGen
  - AcmeServletObject, [1942](#)
  - Onboard Self-Signed Certificate Generation, [1513](#)
- GetDataPointer
  - IPv6FrameProcessingStruct, [2131](#)
- GetDataSize
  - UDPPacket, [2392](#)
- GetDestinationAddress
  - UDPPacket, [2393](#)
- GetDestinationMTU
  - IPv6Interface, [2135](#)
- GetDestinationPort
  - UDPPacket, [2393](#)
- GetDhcpExpirationTime
  - DhcpObject, [2055](#)
- GetDhcpRebindTime
  - DhcpObject, [2055](#)
  - IPV6\_DHCPD, [2115](#)
- GetDhcpRenewTime
  - DhcpObject, [2055](#)
  - IPV6\_DHCPD, [2115](#)
- GetDHCPState
  - DhcpObject, [2056](#)
- GetDSCP
  - UDPPacket, [2394](#)
- GetEframe
  - IPv6FrameProcessingStruct, [2131](#)
- GetEmbeddedIP4Uint
  - IPADDR6, [2107](#)
- GetEphemeralPort
  - IP, [1305](#)
- GetExtraData
  - Extra File Descriptors, [1211](#)
- GetExtraFD
  - Extra File Descriptors, [1212](#)
- GetFastBuffer
  - Buffers - System Buffer Pool, [1124](#)
- GetFirst
  - ParsedJsonDataSet, [2273](#)
- GetFirst\_IP6\_Interface
  - IPv6Interface, [2135](#)
- GetFirstClient
  - MQTT::Client, [1967](#)
- GetFirstInterface
  - Network Interfaces, [1340](#)
- GetFlags
  - MQTT::mqtt\_leaf, [2203](#)
  - MQTT::PacketInfo::Msg, [2231](#)
- GetFlushTime

- NB::WebSocket, [2448](#)
- getFn
  - PinIO, [2292](#)
- GetFreeCount
  - Buffers - System Buffer Pool, [1124](#)
- GetFreeExtraFDCount
  - Extra File Descriptors, [1212](#)
- GetFreeSocketCount
  - Extra File Descriptors, [1212](#)
- GetFreeWebSocketCount
  - NB::WebSocket, [2450](#)
- GetGroup
  - HtmlPageHandler, [2083](#)
- GetHost
  - ParsedURI, [2282](#)
- GetHostByName
  - DNS - Domain Name System, [1183](#)
- GetHostByNameViaIfNum
  - DNS - Domain Name System, [1184](#)
- getI2CAddress
  - I2CDevice, [2092](#)
- GetId
  - mcanMODM7AE70::CanRxMessage, [1958](#)
- GetInterfaceDHCPState
  - DHCPv4 Client, [1177](#)
- GetInterfaceBlock
  - Network Interfaces, [1340](#)
- GetInterfaceForDestination
  - IPv6Interface, [2135](#)
- GetInterfaceForMyAddress4
  - Network Interfaces, [1341](#)
- GetInterfaceForSource
  - IPv6Interface, [2136](#)
- GetInterfaceLink
  - Network Interfaces, [1341](#)
- GetInterfaceN
  - IPv6Interface, [2136](#)
- GetInterfaceName
  - InterfaceBlock, [2100](#)
- GetInterfaceNumber
  - InterfaceBlock, [2100](#)
  - IPv6Interface, [2136](#)
  - Network Interfaces, [1341](#)
- GetIoExpandStruct
  - Extra File Descriptors, [1212](#)
- GetIpPkt
  - IP, [1306](#)
- GetLength
  - mcanMODM7AE70::CanRxMessage, [1958](#)
- GetLocalIPInterface
  - IP, [1307](#)
- GetLocalIPInterface4
  - IP, [1307](#)
- GetLocalIPInterface6
  - IP, [1307](#)
- GetMacSource
  - UDPPacket, [2395](#)
- GetMaxRxBuffers
  - SAICtx\_t, [2306](#)
- GetMaxTxBuffers
  - SAICtx\_t, [2306](#)
- GetMicGain
  - WM8904, [2529](#)
- GetMultiHomeInterface
  - IP, [1308](#)
- GetMyFirstAddress
  - IPv6Interface, [2136](#)
- GetMyNextAddress
  - IPv6Interface, [2137](#)
- GetNewMessage
  - canMCF5441x::CanRxMessage, [1957](#)
  - mcanMODM7AE70::CanRxMessage, [1959](#)
- GetNext
  - IPV6\_DEST, [2114](#)
  - IPV6\_DNS, [2117](#)
  - IPV6\_NEIGHBOR, [2120](#)
  - IPV6\_PREFIX, [2123](#)
  - IPV6\_ROUTER, [2129](#)
  - ParsedJsonDataSet, [2273](#)
- GetNext\_IP6\_Interface
  - IPv6Interface, [2137](#)
- GetNextArray
  - ParsedJsonDataSet, [2274](#)
- GetNextBoolInCurrentArray
  - ParsedJsonDataSet, [2274](#)
- GetNextInterface
  - Network Interfaces, [1341](#)
- GetNextName
  - ParsedJsonDataSet, [2274](#)
- GetNextNameInCurrentArray
  - ParsedJsonDataSet, [2275](#)
- GetNextNameInCurrentObject
  - ParsedJsonDataSet, [2275](#)
- GetNextNumberInCurrentArray
  - ParsedJsonDataSet, [2275](#)
- GetNextObject
  - ParsedJsonDataSet, [2276](#)
- GetNextObjectInCurrentArray
  - ParsedJsonDataSet, [2276](#)
- GetNextStringInCurrentArray
  - ParsedJsonDataSet, [2276](#)
- GetNextValid
  - IPV6\_ROUTER, [2129](#)
- GetnIf
  - I6Record, [2095](#)
- getoption
  - NB::WebSocket, [2453](#)
- GetPacketId
  - UDPPacket, [2397](#)
- GetPacketTypeString
  - MQTT, [1900](#)
- GetParsePosition
  - ParsedJsonDataSet, [2277](#)
- GetPath
  - ParsedURI, [2283](#)
- GetPingReplyTick

- NB::WebSocket, [2457](#)
- GetPOPErrString
  - POP3 - Post Office Protocol, [1361](#)
- GetPort
  - ParsedURI, [2283](#)
- getPort
  - NTPServer, [2248](#)
- getPPSCapture
  - IEEE1588Timer, [2096](#)
- getPPSLoop
  - IEEE1588Timer, [2096](#)
- GetPreciseTime
  - Utilities, [1793](#)
- GetPrintSize
  - ParsedJsonDataSet, [2277](#)
- getPriority
  - MQTT::mqtt\_leaf, [2203](#)
- GetPrivateKeyLen
  - SSL/TLS, [1490](#)
- GetPrivateKeyPEM
  - SSL/TLS, [1490](#)
- GetProperInterface4
  - ARP - Address Resolution Protocol, [1110](#)
  - IP, [1308](#)
- GetPubTopicName
  - MQTT::ConfigExposer, [2041](#)
  - MQTT::mqtt\_leaf, [2203](#)
- GetRawCurrent
  - ParsedJsonDataSet, [2277](#)
- GetRawValue
  - config\_pass, [2016](#)
- GetRecordFromName
  - HTTP and HTML Functions, [1263](#)
- GetReleaseTag
  - System Functions, [1541](#)
- GetRemainingDhcpLeasePreferredTime
  - IPV6\_PREFIX, [2123](#)
- GetRemainingDhcpLeaseTime
  - DhcpObject, [2056](#)
  - IPV6\_PREFIX, [2123](#)
- GetRenderLength
  - MQTT::ConfigExposer, [2041](#)
  - MQTT::mqtt\_leaf, [2204](#)
  - MQTT::mqtt\_obj, [2214](#)
- GetReportItem
  - I4Record, [2094](#)
- GetReturnCodeString
  - MQTT, [1900](#)
- GetSem
  - DSPIModule, [2068](#)
  - SPIModule, [2336](#)
- GetSerializationForm
  - MQTT::mqtt\_leaf, [2204](#)
- getsocketerror
  - TCP, [1626](#)
- GetSocketLocalAddr
  - TCP, [1631](#)
- GetSocketLocalPort
  - TCP, [1632](#)
- GetSocketRemoteAddr
  - TCP, [1632](#)
- GetSocketRemotePort
  - TCP, [1633](#)
- getsockopt
  - TCP, [1634](#)
- GetSocksProxySettings
  - SOCKS, [1445](#)
- GetSourceAddress
  - UDPPacket, [2397](#)
- GetSourcePort
  - UDPPacket, [2398](#)
- GetSR\_IntLevel
  - Interrupt Macro: MOD5441x, NANO54415, SB800EX, [1371](#)
- GetSrcIp
  - IP, [1309](#)
- GetSrcIp4
  - IP, [1309](#)
- GetSrcIp6
  - IP, [1310](#)
- GetSrcIp6wlf
  - IP, [1310](#)
- GetSrcIpwlf
  - IP, [1310](#)
- GetSrcIpwlf4
  - IP, [1311](#)
- GetState
  - IPV6\_NEIGHBOR, [2120](#)
  - IPV6\_PREFIX, [2123](#)
  - NB::WebSocket, [2460](#)
- getState
  - SAICtx\_t, [2306](#)
  - SSCCtx\_t, [2347](#)
- GetStateCC
  - AcmeServletObject, [1941](#)
- getStickyRxError
  - SAICtx\_t, [2307](#)
- getStickyTxError
  - SAICtx\_t, [2307](#)
- GetSubTopicFilter
  - MQTT::ConfigExposer, [2042](#)
  - MQTT::mqtt\_leaf, [2204](#)
  - MQTT::mqtt\_obj, [2215](#)
- GetSum
  - IP, [1311](#)
- GetSum20
  - IP, [1312](#)
- GetSumHdr
  - IP, [1312](#)
- GetTcpRtxCount
  - TCP, [1638](#)
- GetTextValue
  - config\_bool, [1978](#)
  - config\_double, [1985](#)
  - config\_int, [1991](#)
  - config\_IPADDR, [1997](#)

- config\_IPADDR4, [2001](#)
- config\_MACADR, [2009](#)
- config\_obj, [2013](#)
- config\_pass, [2016](#)
- config\_string, [2023](#)
- config\_uint, [2033](#)
- CUR\_IPADDR4, [2048](#)
- Dynamic\_IPADDR, [2073](#)
- GetTFTP
  - TFTP Client, [1716](#)
- GetTime
  - StopWatch, [2352](#)
- GetTimeStamp
  - mcanMODM7AE70::CanRxMessage, [1959](#)
- Getting Started Guide, [55](#)
- GetType
  - MQTT::PacketInfo::Msg, [2231](#)
- GetTypeValue
  - config\_bool, [1978](#)
  - config\_chooser, [1981](#)
  - config\_double, [1985](#)
  - config\_double\_limit, [1989](#)
  - config\_int, [1991](#)
  - config\_IPADDR, [1997](#)
  - config\_IPADDR4, [2001](#)
  - config\_MACADR, [2009](#)
  - config\_obj, [2013](#)
  - config\_string, [2023](#)
  - config\_uint, [2033](#)
  - CUR\_IPADDR4, [2048](#)
  - Dynamic\_IPADDR, [2073](#)
- GetUartErrorReg
  - Serial Interfaces, [1515](#)
- GetUdpPkt
  - IP, [1313](#)
- GetUniquelIdentifier
  - UniquelIdentifier, [2420](#), [2421](#)
- GetUserParameters
  - System Functions, [1541](#)
- GetVolume
  - WM8904, [2529](#)
- GetWebSocketRecord
  - NB::WebSocket, [2463](#)
- GetWriteSpace
  - NB::WebSocket, [2463](#)
- GIF Canvas, [997](#)
- GPIO, [1246](#)
- GPIO - Simple, [722](#), [725](#)
- GPIO Blink, [501](#)
- GPIO Command Server, [704](#)
- HadLink
  - IPv6Interface, [2137](#)
- HAL - Hardware Abstraction Layer, [1247](#)
  - FlashErase, [1248](#)
  - FlashProgram, [1248](#)
  - FlashProgramApplImage, [1249](#)
  - ForceReboot, [1249](#)
  - HalDeviceCertValid, [1250](#)
  - HalEraseDeviceCertAndKey, [1250](#)
  - HalGetDeviceCert, [1250](#)
  - HalGetDeviceCertLen, [1250](#)
  - HalGetDeviceKey, [1251](#)
  - HalGetDeviceKeyLen, [1251](#)
  - HalGetTickFraction, [1251](#)
  - HalSaveNewDeviceCert, [1251](#)
  - HalSaveNewDeviceKey, [1251](#)
  - HalStorage\_AddressOffset, [1252](#)
  - HalStorage\_Erase, [1252](#)
  - HalStorage\_Finalize, [1253](#)
  - HalStorage\_GetAllocated, [1253](#)
  - HalStorage\_GetMaxAllocation, [1253](#)
  - HalStorage\_Prepare, [1253](#)
  - HalStorage\_RemainingSpace, [1254](#)
  - HalStorage\_Save, [1254](#)
  - HalStorage\_SavePartial, [1254](#)
  - HalStorage\_WriteOffset, [1255](#)
  - HalTickMaxCount, [1257](#)
  - HardwareSetup, [1255](#)
  - PostConfigHardwareInit, [1255](#)
  - spaceleft, [1256](#)
  - StdioCheckIntc, [1256](#)
  - SysLogCheckIntc, [1256](#)
  - watchdog\_service\_function, [1257](#)
- HalDeviceCertValid
  - HAL - Hardware Abstraction Layer, [1250](#)
- HalEraseDeviceCertAndKey
  - HAL - Hardware Abstraction Layer, [1250](#)
- HalGetDeviceCert
  - HAL - Hardware Abstraction Layer, [1250](#)
- HalGetDeviceCertLen
  - HAL - Hardware Abstraction Layer, [1250](#)
- HalGetDeviceKey
  - HAL - Hardware Abstraction Layer, [1251](#)
- HalGetDeviceKeyLen
  - HAL - Hardware Abstraction Layer, [1251](#)
- HalGetTickFraction
  - HAL - Hardware Abstraction Layer, [1251](#)
- HalSaveNewDeviceCert
  - HAL - Hardware Abstraction Layer, [1251](#)
- HalSaveNewDeviceKey
  - HAL - Hardware Abstraction Layer, [1251](#)
- HalStorage\_AddressOffset
  - HAL - Hardware Abstraction Layer, [1252](#)
- HalStorage\_Erase
  - HAL - Hardware Abstraction Layer, [1252](#)
- HalStorage\_Finalize
  - HAL - Hardware Abstraction Layer, [1253](#)
- HalStorage\_GetAllocated
  - HAL - Hardware Abstraction Layer, [1253](#)
- HalStorage\_GetMaxAllocation
  - HAL - Hardware Abstraction Layer, [1253](#)
- HalStorage\_Prepare
  - HAL - Hardware Abstraction Layer, [1253](#)
- HalStorage\_RemainingSpace
  - HAL - Hardware Abstraction Layer, [1254](#)
- HalStorage\_Save

- HAL - Hardware Abstraction Layer, [1254](#)
- HalStorage\_SavePartial
  - HAL - Hardware Abstraction Layer, [1254](#)
- HalStorage\_WriteOffset
  - HAL - Hardware Abstraction Layer, [1255](#)
- HalTickMaxCount
  - HAL - Hardware Abstraction Layer, [1257](#)
- handle
  - \_lpspi\_master\_edma\_private\_handle, [1927](#)
- HandlePubAck
  - MQTT::mqtt\_leaf, [2204](#)
- HandleSubAck
  - MQTT::mqtt\_leaf, [2205](#)
- HandleUnsubAck
  - MQTT::mqtt\_leaf, [2205](#)
- hard\_size
  - ARP - Address Resolution Protocol, [1117](#)
- hard\_Type
  - ARP - Address Resolution Protocol, [1117](#)
- Hardware Connections, [763](#)
- HardwareSetup
  - HAL - Hardware Abstraction Layer, [1255](#)
- haserror
  - IOSYS - I/O System, [1288](#)
- HasRoute
  - IPv6Interface, [2137](#)
- HaveActiveNetwork
  - IPADDR4 Functions, [1344](#)
- hCSum
  - IPPKT, [2112](#)
- header
  - \_edma\_handle, [1907](#)
- High Resolution Delay Timer, [1722](#)
- highAlarmTemp
  - \_tempmon\_config, [1940](#)
- HopByHopParseHasError
  - IPv6FrameProcessingStruct, [2131](#)
- HTML Application File Post, [1000](#)
- HTML Cookie, [1002](#)
- HTML File Post, [859](#), [990](#)
- HTML File Post Task Handoff, [992](#)
- HTML Form Post, [862](#), [1005](#)
- HTML Form Post 2.X Compatible, [1014](#)
- HTML Only Configuration, [427](#)
- HTML Password, [1017](#)
- HTML Processing, [206](#)
- HTML Server GET Request, [1021](#)
- HTML Variables, [465](#)
- HtmlPageHandler, [2081](#)
  - FindHandler, [2083](#)
  - GetGroup, [2083](#)
  - HtmlPageHandler, [2083](#)
  - MoveHandlerToLast, [2083](#)
  - ProcessRaw, [2084](#)
- HtmlPostDateTime, [1018](#)
- HtmlPostVariableListCallback, [2084](#)
  - HtmlPostVariableListCallback, [2084](#), [2085](#)
- HTMLVariables, [1030](#)

- HTTP, [461](#)
- HTTP and HTML Functions, [1257](#)
  - BadRequestResponse, [1261](#)
  - CheckHttpAccess, [1261](#)
  - ConfigRenderFunc, [1262](#)
  - EmptyResponse, [1262](#)
  - EnableRemoteConsole, [1263](#)
  - ForbiddenResponse, [1263](#)
  - GetRecordFromName, [1263](#)
  - HTTP\_ACCESS, [1260](#)
  - HTTP\_FORBIDEN, [1260](#)
  - http\_gethandlerfunc, [1259](#)
  - HTTP\_NEED\_PASSWORD, [1260](#)
  - HTTP\_NOTFOUND, [1260](#)
  - HTTP\_OK\_TO\_SERVE, [1260](#)
  - http\_posthandler, [1259](#)
  - HTTP\_REQ\_RD\_RESULT, [1260](#)
  - HTTP\_RequestTypes, [1260](#)
  - HTTP\_RESPONSE\_HANDLED, [1260](#)
  - HTTP\_RRR\_BUF\_FULL, [1260](#)
  - HTTP\_RRR\_END\_FOUND, [1260](#)
  - HTTP\_RRR\_FINISHED, [1260](#)
  - HTTP\_RRR SOCK\_ERR, [1260](#)
  - httpstricmp, [1263](#)
  - NoContentResponse, [1264](#)
  - NotAvailableResponse, [1264](#)
  - NotFoundResponse, [1265](#)
  - RedirectResponse, [1265](#)
  - SendEmailResponse, [1265](#)
  - SendFileFragment, [1266](#)
  - SendFullResponse, [1266](#), [1267](#)
  - SendGifHeader, [1267](#)
  - SendHeaderResponse, [1267](#), [1269](#)
  - SendHTMLHeader, [1269](#)
  - SendHTMLHeaderWCookie, [1269](#)
  - SendTextHeader, [1270](#)
  - StartHttp, [1270](#)
  - StartHttps, [1270](#)
  - StopHttp, [1271](#)
  - tDelete, [1261](#)
  - tGet, [1261](#)
  - tHead, [1261](#)
  - tPost, [1261](#)
  - tPut, [1261](#)
  - tUnknown, [1261](#)
  - writeallsafestring, [1271](#)
  - WriteHtmlVariable, [1271](#)
  - writesafestring, [1272](#)
- HTTP\_ACCESS
  - HTTP and HTML Functions, [1260](#)
- HTTP\_FORBIDEN
  - HTTP and HTML Functions, [1260](#)
- http\_gethandlerfunc
  - HTTP and HTML Functions, [1259](#)
- HTTP\_NEED\_PASSWORD
  - HTTP and HTML Functions, [1260](#)
- HTTP\_NOTFOUND
  - HTTP and HTML Functions, [1260](#)

- HTTP\_OK\_TO\_SERVE
  - HTTP and HTML Functions, [1260](#)
- http\_posthandler
  - HTTP and HTML Functions, [1259](#)
- HTTP\_REQ\_RD\_RESULT
  - HTTP and HTML Functions, [1260](#)
- HTTP\_Request, [2085](#)
  - GetBoundaryMarker, [2086](#)
  - ReadSimpleBody, [2086](#)
- HTTP\_RequestTypes
  - HTTP and HTML Functions, [1260](#)
- HTTP\_RESPONSE\_HANDLED
  - HTTP and HTML Functions, [1260](#)
- HTTP\_RRR\_BUF\_FULL
  - HTTP and HTML Functions, [1260](#)
- HTTP\_RRR\_END\_FOUND
  - HTTP and HTML Functions, [1260](#)
- HTTP\_RRR\_FINISHED
  - HTTP and HTML Functions, [1260](#)
- HTTP\_RRR SOCK\_ERR
  - HTTP and HTML Functions, [1260](#)
- HTTPS Configuration Mirror, [907](#)
- HTTPS Dual Cert, [865](#)
- HTTPS GET Request, [878](#)
- HTTPS Upload Cert, [869](#)
- HTTPS Web Server Demo, [875](#)
- HTTPS WebSocket Demo, [1078](#)
- httpstricmp
  - HTTP and HTML Functions, [1263](#)
- I2C, [506](#), [655](#), [704](#), [1272](#), [2087](#)
  - DoTransaction, [2089](#)
  - I2C, [2089](#)
  - I2C\_RES\_ACK, [2088](#)
  - I2C\_RES\_ARB\_LST, [2088](#)
  - I2C\_RES\_ARG, [2088](#)
  - I2C\_RES\_BUSY, [2088](#)
  - I2C\_RES\_NACK, [2088](#)
  - readReg8, [2089](#)
  - readRegN, [2089](#)
  - Result\_t, [2088](#)
  - setNumAddressBytes, [2090](#)
  - setup, [2090](#)
  - writeReg8, [2090](#)
  - writeRegN, [2090](#)
- I2C Address Scan, [656](#), [704](#)
- I2C PicKit Demo Board, [661](#), [709](#)
- I2C\_RES\_ACK
  - BBI2C, [1944](#)
  - I2C, [2088](#)
- I2C\_RES\_ARB\_LST
  - BBI2C, [1944](#)
  - I2C, [2088](#)
- I2C\_RES\_ARG
  - BBI2C, [1944](#)
  - I2C, [2088](#)
- I2C\_RES\_BUSY
  - BBI2C, [1944](#)
  - I2C, [2088](#)
- I2C\_RES\_ERR
  - BBI2C, [1944](#)
- I2C\_RES\_IN\_PROGRESS
  - BBI2C, [1944](#)
- I2C\_RES\_NACK
  - BBI2C, [1944](#)
  - I2C, [2088](#)
- I2CDevice, [2091](#)
  - getI2CAddress, [2092](#)
  - I2CDevice, [2092](#)
  - readReg8, [2092](#)
  - readRegN, [2092](#)
  - setup, [2093](#)
  - writeReg8, [2093](#)
  - writeRegN, [2093](#)
- I2CMultiChannelResetPeripheral
  - MOD5441x, NANO54415, SB800EX, [1275](#)
- I4Record, [2094](#)
  - Attach, [2094](#)
  - GetReportItem, [2094](#)
  - I4Record, [2094](#)
- I6Record, [2094](#)
  - Attach, [2095](#)
  - GetIf, [2095](#)
  - I6Record, [2095](#)
  - ReportGetTextValue, [2095](#)
- id
  - \_flexcan\_frame, [1914](#)
  - \_flexcan\_rx\_mb\_config, [1918](#)
- idFilterNum
  - \_flexcan\_rx\_fifo\_config, [1917](#)
- idFilterTable
  - \_flexcan\_rx\_fifo\_config, [1917](#)
- idFilterType
  - \_flexcan\_rx\_fifo\_config, [1917](#)
- idhit
  - \_flexcan\_frame, [1914](#)
- IEEE1588Timer, [2095](#)
  - enablePPSCapture, [2096](#)
  - getConsecutiveValidSeconds, [2096](#)
  - getCurrentTime, [2096](#)
  - getPPSCapture, [2096](#)
  - getPPSLoop, [2096](#)
  - init, [2096](#)
  - markValidTime, [2097](#)
  - PPS\_1588\_TimeCaplSr, [2097](#)
  - resetConsecutiveSeconds, [2097](#)
  - ring\_diff\_forward, [2097](#)
  - updateConsecutiveSeconds, [2097](#)
- IncUsageCount
  - Buffers - System Buffer Pool, [1125](#)
- Init
  - DSPIModule, [2068](#)
  - SAICtx\_t, [2307](#)
  - SPI\_QSPI, [2322](#)
  - SPI\_USART, [2327](#)
  - SPIModule, [2336](#), [2337](#)
  - SSCCtx\_t, [2347](#)

- UserAuthManager, [2424](#)
- WM8904, [2530](#)
- init
  - IEEE1588Timer, [2096](#)
  - Initialization - System Initialization Functions, [1328](#)
  - mcanMODM7AE70::mcan\_module, [2163](#)
- InitAP\_SPI
  - Wifi, [1884](#)
- InitBuffers
  - Buffers - System Buffer Pool, [1125](#)
- InitGDBStub
  - GDB Stub (Remote Debugging), [1245](#)
- InitGDBStubNoBreak
  - GDB Stub (Remote Debugging), [1245](#)
- Initialization - System Initialization Functions, [1327](#)
  - EnableSecureConfigServer, [1328](#)
  - EnableSystemDiagnostics, [1328](#)
  - init, [1328](#)
  - StartHttp, [1329](#)
  - StartHttpCustom, [1329](#)
  - StartHttps, [1329](#)
  - WaitForActiveNetwork, [1330](#)
- InitializationErrors
  - NB::Error::Init, [1903](#)
- InitializeStack
  - IP, [1313](#)
- InitIPv6
  - DHCPv6 Client, [1179](#)
  - IPv6Interface, [2141](#)
- InitPlatformDiag
  - Diagnostics (DIAG), [1190](#)
- InitWifi\_Serial
  - Wifi, [1885](#)
- InitWifi\_SPI
  - Wifi, [1885](#)
- InProgress
  - NB::Error::Scan, [1903](#)
- Insert
  - IPV6\_ROOT\_EL\_CONTAINER, [2127](#)
- InsertAfter
  - buffer\_list, [1949](#)
- InsertBack
  - IPV6\_ROOT\_EL\_CONTAINER, [2127](#)
- InsertBefore
  - buffer\_list, [1949](#)
- InsertHead
  - buffer\_list, [1949](#)
- InsertTail
  - buffer\_list, [1950](#)
- InterfaceAutoIP
  - IPADDR4 Functions, [1345](#)
- InterfaceBlock, [2097](#)
  - EnableMulticast, [2100](#)
  - GetInterfaceName, [2100](#)
  - GetInterfaceNumber, [2100](#)
  - InterfaceBlock, [2100](#)
  - InterfaceLinkChange, [2101](#)
  - IsRootInterface, [2101](#)
- LinkActive, [2101](#)
- LinkDuplex, [2101](#)
- LinkSpeed, [2101](#)
- RegisterInterface, [2102](#)
- InterfaceDNS
  - IPADDR4 Functions, [1345](#)
- InterfaceDNS2
  - IPADDR4 Functions, [1345](#)
- InterfaceGate
  - IPADDR4 Functions, [1345](#)
- InterfaceIP
  - IPADDR4 Functions, [1346](#)
- InterfaceLinkActive
  - Network Interfaces, [1342](#)
- InterfaceLinkChange
  - InterfaceBlock, [2101](#)
- InterfaceLinkDuplex
  - Network Interfaces, [1342](#)
- InterfaceLinkSpeed
  - Network Interfaces, [1342](#)
- InterfaceMAC
  - Network Interfaces, [1343](#)
- InterfaceMASK
  - IPADDR4 Functions, [1346](#)
- InterfaceName
  - Network Interfaces, [1343](#)
- Interpolate
  - NBInterpolatedString, [2236](#)
  - NBString, [2242](#), [2243](#)
- Interrupt Handling, [220](#)
- Interrupt Macro: MOD5441x,NANO54415, SB800EX, [1371](#)
  - GetSR\_IntLevel, [1371](#)
  - SetSR\_IntLevel, [1372](#)
- Interrupts, [722](#)
- InterruptTriggerBothEdge
  - Device Interrupts, [1187](#)
- InterruptTriggerFallingEdge
  - Device Interrupts, [1187](#)
- InterruptTriggerNone
  - Device Interrupts, [1187](#)
- InterruptTriggerRisingEdge
  - Device Interrupts, [1187](#)
- InterruptTriggerSensitive
  - Device Interrupts, [1187](#)
- Interval Timer, [961](#), [1723](#)
  - IntervalInterruptCallback, [1727](#)
  - IntervalOSFlag, [1730](#)
  - IntervalOSSem, [1734](#)
  - IntervalStop, [1736](#)
- IntervalInterruptCallback
  - Interval Timer, [1727](#)
- IntervalOSFlag
  - Interval Timer, [1730](#)
- IntervalOSSem
  - Interval Timer, [1734](#)
- IntervalStop
  - Interval Timer, [1736](#)



- Introduction, [145](#)
- InvalidArgument
  - NB::Error, [1902](#)
- InvalidInfo
  - NB::Error::Init, [1903](#)
- InvalidRequest
  - NB::Error, [1902](#)
- ioctl
  - IOSYS - I/O System, [1288](#)
- IOSYS - I/O System, [1279](#)
  - charavail, [1283](#)
  - close, [1283](#)
  - CurrentStdioFD, [1283](#)
  - dataavail, [1284](#)
  - eClosingNow, [1283](#)
  - eErrorSet, [1283](#)
  - eReadSet, [1282](#)
  - eWriteSet, [1283](#)
  - FD\_CLR, [1284](#)
  - FD\_CLRFROMSET, [1284](#)
  - FD\_COPY, [1285](#)
  - FD\_ISSET, [1285](#)
  - FD\_OVERLAP, [1285](#)
  - FD\_SET, [1286](#)
  - FD\_SETFROMSET, [1286](#)
  - FD\_ZERO, [1286](#)
  - FDCallBack, [1282](#)
  - FDChangeType, [1282](#)
  - getchar, [1288](#)
  - haserror, [1288](#)
  - ioctl, [1288](#)
  - peek, [1289](#)
  - read, [1289](#)
  - readall, [1290](#)
  - ReadAllWithTickTimeout, [1290](#)
  - ReadAllWithTimeout, [1291](#)
  - ReadWithTickTimeout, [1291](#)
  - ReadWithTimeout, [1292](#)
  - RegisterFDCallBack, [1293](#)
  - ReplaceStdio, [1293](#)
  - select, [1294](#)
  - SockPeekWithTimeout, [1294](#)
  - write, [1295](#)
  - writeall, [1295](#)
  - writeavail, [1296](#)
  - writestring, [1296](#)
  - ZeroWaitSelect, [1297](#)
- IP, [1298](#)
  - AddInterface, [1304](#)
  - FixHeaderAndSend, [1304](#)
  - FixHeaderAndSendViaInterface, [1305](#)
  - FragmentProcessFunction, [1302](#)
  - GetEphemeralPort, [1305](#)
  - GetIpPkt, [1306](#)
  - GetLocalIPInterface, [1307](#)
  - GetLocalIPInterface4, [1307](#)
  - GetLocalIPInterface6, [1307](#)
  - GetMultiHomeInterface, [1308](#)
  - GetProperInterface4, [1308](#)
  - GetSrcIp, [1309](#)
  - GetSrcIp4, [1309](#)
  - GetSrcIp6, [1310](#)
  - GetSrcIp6wlf, [1310](#)
  - GetSrcIpwlf, [1310](#)
  - GetSrcIpwlf4, [1311](#)
  - GetSum, [1311](#)
  - GetSum20, [1312](#)
  - GetSumHdr, [1312](#)
  - GetUdpPkt, [1313](#)
  - InitializeStack, [1313](#)
  - IpHeaderIPv4, [1302](#)
  - IpProcessEthernetPacket, [1314](#)
  - IsLocal4, [1314](#)
  - IsMyIp4, [1315](#)
  - IsMyIp6, [1315](#)
  - pFragFunc, [1321](#)
  - Ping, [1316](#)
  - Ping4, [1316](#)
  - Ping4ViaInterface, [1317](#)
  - Ping6, [1318](#)
  - Ping6ViaInterface, [1318](#)
  - PingViaInterface, [1319](#)
  - pPromiscuousPacketFunc, [1321](#)
  - PromiscuousPacketFunc, [1303](#)
  - pTaskPacketFunc, [1321](#)
  - SendICMPError, [1319](#)
  - ShowIPBuffer, [1320](#)
- IP Address Object (IPADDR), [526](#)
- IPADDR4, [2102](#)
  - fdprint, [2103](#)
  - IPADDR4, [2103](#)
  - IsAutoIP, [2103](#)
  - IsGlobalBroadCast, [2104](#)
  - IsLoopBack, [2104](#)
  - IsmDns, [2104](#)
  - IsMultiCast, [2104](#)
  - IsNull, [2104](#)
  - NotNull, [2105](#)
  - SetFromAscii, [2105](#)
  - SetNull, [2105](#)
  - sprintf, [2105](#)
- IPADDR4 Class, [1321](#)
- IPADDR4 Functions, [1344](#)
  - HaveActiveNetwork, [1344](#)
  - InterfaceAutoIP, [1345](#)
  - InterfaceDNS, [1345](#)
  - InterfaceDNS2, [1345](#)
  - InterfaceGate, [1345](#)
  - InterfaceIP, [1346](#)
  - InterfaceMASK, [1346](#)
- IPADDR6, [2106](#)
  - AsciiToIp6, [2107](#)
  - Extract4, [2107](#)
  - fdprint, [2107](#)
  - GetEmbeddedIP4Uint, [2107](#)
  - IsEmbeddedIPv4, [2108](#)



- IsLinkLocal, [2108](#)
- IsLoopBack, [2108](#)
- IsMultiCast, [2108](#)
- IsNull, [2109](#)
- McastMac, [2109](#)
- NotNull, [2109](#)
- NullIP, [2109](#)
- print, [2110](#)
- SetFromAscii, [2110](#)
- SetFromIP4, [2110](#)
- SetFromUInt32Shortcut, [2110](#)
- SetNull, [2111](#)
- sprintf, [2111](#)
- IPADDR6 Class, [1322](#)
- IpHeaderIPv4
  - IP, [1302](#)
- IPPKT, [2112](#)
  - hCSum, [2112](#)
- IpProcessEthernetPacket
  - IP, [1314](#)
- IPv4/IPv6 Dual Stack Guide, [176](#)
- IPv6, [528](#)
- IPv6 Data Structures, [1323](#)
  - IPV6\_ROOT\_EL\_TYPE, [1324](#)
  - IPV6\_ROOT\_TYPE\_DEST, [1324](#)
  - IPV6\_ROOT\_TYPE\_DHCPD, [1324](#)
  - IPV6\_ROOT\_TYPE\_DNS, [1324](#)
  - IPV6\_ROOT\_TYPE\_EL, [1324](#)
  - IPV6\_ROOT\_TYPE\_EL\_CONTAINER, [1324](#)
  - IPV6\_ROOT\_TYPE\_NEIGHBOR, [1324](#)
  - IPV6\_ROOT\_TYPE\_PREFIX, [1324](#)
  - IPV6\_ROOT\_TYPE\_ROUTER, [1324](#)
- IPv6 DNS, [1323](#)
- IPv6 Frame Processing, [1324](#)
- IPv6 Interface Management, [1323](#)
  - ShowIP6Counters, [1323](#)
- IPv6 Multicast Constants, [1325](#)
- IPv6 Neighbor Discovery, [1325](#)
  - eDelay, [1326](#)
  - eIncomplete, [1326](#)
  - eMY\_NEIGHBOR\_STATE, [1326](#)
  - eProbe, [1326](#)
  - eReachable, [1326](#)
  - eStale, [1326](#)
- IPv6 Routing, [1326](#)
  - eDHCP, [1327](#)
  - eDoingND, [1327](#)
  - eInvalid, [1327](#)
  - eLinkLocal, [1327](#)
  - eMY\_ADDR\_STATE, [1326](#)
  - eNoRoute, [1327](#)
  - ePrefixSource, [1327](#)
  - eRouteOutResult, [1327](#)
  - eRouter, [1327](#)
  - eSent, [1327](#)
  - eStatic, [1327](#)
  - eTentative, [1327](#)
  - eUnknown, [1327](#)
  - eValid\_Depricated, [1327](#)
  - eValid\_Prefered, [1327](#)
- IPV6\_DEST, [2113](#)
  - AgeStillValidTest, [2114](#)
  - GetNext, [2114](#)
- IPV6\_DHCPD, [2114](#)
  - AgeStillValidTest, [2115](#)
  - FindIA\_Addr, [2115](#)
  - GetDhcpRebindTime, [2115](#)
  - GetDhcpRenewTime, [2115](#)
- IPV6\_DNS, [2116](#)
  - AgeStillValidTest, [2117](#)
  - GetNext, [2117](#)
- IPV6\_IA\_ADDR, [2117](#)
  - AgeStillValidTest, [2118](#)
- IPV6\_NEIGHBOR, [2118](#)
  - AgeStillValidTest, [2119](#)
  - CoreSendPend, [2119](#)
  - GetNext, [2120](#)
  - GetState, [2120](#)
  - ProcessAdvert, [2120](#)
  - Send, [2120](#)
  - Send\_ND\_Solicit, [2120](#)
  - SendPending, [2121](#)
  - SetState, [2121](#)
  - StillValidToSend, [2121](#)
- IPV6\_PREFIX, [2121](#)
  - AgeStillPreferred, [2123](#)
  - AgeStillValidTest, [2123](#)
  - GetNext, [2123](#)
  - GetRemainingDhcpLeasePreferredTime, [2123](#)
  - GetRemainingDhcpLeaseTime, [2123](#)
  - GetState, [2123](#)
  - OnLink, [2124](#)
  - Preferred, [2124](#)
  - RemainingValidTime, [2124](#)
  - Source, [2124](#)
- IPV6\_ROOT\_EL, [2124](#)
  - AgeStillValidTest, [2125](#)
- IPV6\_ROOT\_EL\_CONTAINER, [2125](#)
  - Attach, [2126](#)
  - Find, [2126](#)
  - First, [2127](#)
  - Insert, [2127](#)
  - InsertBack, [2127](#)
  - MoveToTop, [2127](#)
  - Remove, [2127](#)
- IPV6\_ROOT\_EL\_TYPE
  - IPv6 Data Structures, [1324](#)
- IPV6\_ROOT\_TYPE\_DEST
  - IPv6 Data Structures, [1324](#)
- IPV6\_ROOT\_TYPE\_DHCPD
  - IPv6 Data Structures, [1324](#)
- IPV6\_ROOT\_TYPE\_DNS
  - IPv6 Data Structures, [1324](#)
- IPV6\_ROOT\_TYPE\_EL
  - IPv6 Data Structures, [1324](#)
- IPV6\_ROOT\_TYPE\_EL\_CONTAINER

- IPv6 Data Structures, [1324](#)
- IPV6\_ROOT\_TYPE\_NEIGHBOR
  - IPv6 Data Structures, [1324](#)
- IPV6\_ROOT\_TYPE\_PREFIX
  - IPv6 Data Structures, [1324](#)
- IPV6\_ROOT\_TYPE\_ROUTER
  - IPv6 Data Structures, [1324](#)
- IPV6\_ROUTER, [2128](#)
  - AgeStillValidTest, [2129](#)
  - GetNext, [2129](#)
  - GetNextValid, [2129](#)
- IPv6FrameProcessingStruct, [2129](#)
  - DestOptHasError, [2131](#)
  - GetDataPointer, [2131](#)
  - GetEframe, [2131](#)
  - HopByHopParseHasError, [2131](#)
  - IPv6FrameProcessingStruct, [2130](#)
  - RouterHeaderParseHasError, [2131](#)
  - WalkNextHeader, [2132](#)
- IPv6Interface, [2132](#)
  - AddDefaultGateway, [2134](#)
  - AddStaticAddress, [2134](#)
  - AddStaticDNS, [2135](#)
  - GetDestinationMTU, [2135](#)
  - GetFirst\_IP6\_Interface, [2135](#)
  - GetInterfaceForDestination, [2135](#)
  - GetInterfaceForSource, [2136](#)
  - GetInterfaceN, [2136](#)
  - GetInterfaceNumber, [2136](#)
  - GetMyFirstAddress, [2136](#)
  - GetMyNextAddress, [2137](#)
  - GetNext\_IP6\_Interface, [2137](#)
  - HadLink, [2137](#)
  - HasRoute, [2137](#)
  - InitIPv6, [2141](#)
  - IPv6Interface, [2134](#)
  - IsMyAddress, [2137](#)
  - MyDNSAddress, [2138](#)
  - MySourceAddress, [2138](#)
  - NotifyReachable, [2138](#)
  - NotifyUnreachable, [2138](#)
  - OnLink, [2139](#)
  - ping, [2139](#)
  - Ping6, [2141](#)
  - RemoveDefaultGateway, [2139](#)
  - RemoveStaticAddress, [2140](#)
  - RemoveStaticDNS, [2140](#)
  - SendMLDRegistration, [2140](#)
  - SetStaticDNS, [2140](#)
  - ShowAllInfo, [2141](#)
  - ShowInfo, [2141](#)
- IsArray
  - JsonRef, [2149](#)
- IsAutoIP
  - CUR\_IPADDR4, [2048](#)
  - IPADDR4, [2103](#)
- IsBool
  - JsonRef, [2150](#)
- isByteSwap
  - \_lpspi\_master\_edma\_handle, [1925](#)
  - \_lpspi\_master\_handle, [1928](#)
  - \_lpspi\_slave\_edma\_handle, [1932](#)
  - \_lpspi\_slave\_handle, [1934](#)
- IsEmbeddedIPv4
  - IPADDR6, [2108](#)
- IsGlobalBroadCast
  - CUR\_IPADDR4, [2048](#)
  - IPADDR4, [2104](#)
- IsInChoices
  - config\_choser, [1981](#), [1982](#)
- IsLinkLocal
  - IPADDR6, [2108](#)
- IsLocal4
  - IP, [1314](#)
- IsLoopBack
  - CUR\_IPADDR4, [2048](#)
  - IPADDR4, [2104](#)
  - IPADDR6, [2108](#)
- IsMailError
  - SMTP - Send Email, [1435](#)
- IsMDns
  - IPADDR4, [2104](#)
- IsMultiCast
  - CUR\_IPADDR4, [2048](#)
  - IPADDR4, [2104](#)
  - IPADDR6, [2108](#)
- IsMyAddress
  - IPv6Interface, [2137](#)
- IsMyAddressUsedArpDetect
  - ARP - Address Resolution Protocol, [1111](#)
- IsMyIpp4
  - IP, [1315](#)
- IsMyIpp6
  - IP, [1315](#)
- IsNameIPAddress
  - DNS - Domain Name System, [1185](#)
- IsNBIdExt
  - MODM7AE70, [1155](#)
- IsNull
  - config\_IPADDR, [1997](#)
  - config\_IPADDR4, [2001](#)
  - CUR\_IPADDR4, [2048](#)
  - IPADDR4, [2104](#)
  - IPADDR6, [2109](#)
  - JsonRef, [2150](#)
- IsNumber
  - JsonRef, [2150](#)
- IsObject
  - JsonRef, [2150](#)
- isPcsContinuous
  - \_lpspi\_master\_edma\_handle, [1925](#)
  - \_lpspi\_master\_handle, [1929](#)
- IsRootInterface
  - InterfaceBlock, [2101](#)
- isRunning
  - NTPServer, [2248](#)

- IsSecure
  - ParsedURI, [2283](#)
- IsSelected
  - config\_chooser, [1982](#)
- IsSSL\_CertNKeyValid
  - SSL/TLS, [1491](#)
- IsSSLfd
  - SSL/TLS, [1507](#)
- IsString
  - JsonRef, [2150](#)
- isThereExtraRxBytes
  - \_lpspi\_master\_edma\_handle, [1926](#)
  - \_lpspi\_slave\_edma\_handle, [1932](#)
- isTxMask
  - \_lpspi\_master\_handle, [1929](#)
- IsValid
  - canMCF5441x::CanRxMessage, [1957](#)
  - mcanMODM7AE70::CanRxMessage, [1959](#)
- JSON, [531](#)
- JSON Array, [532](#)
- JSON Array with Objects, [534](#)
- JSON Lexer, [232](#), [532](#), [1330](#)
  - ALLOC\_STRING, [1332](#)
  - BEGIN\_ARRAY, [1331](#)
  - BEGIN\_OBJECT, [1331](#)
  - CharOutputFn, [1331](#)
  - END\_ARRAY, [1331](#)
  - END\_OBJECT, [1331](#)
  - EOF\_EL, [1332](#)
  - FALSE\_EL, [1332](#)
  - json\_primitive\_type, [1331](#)
  - NAME, [1331](#)
  - NOTFOUND, [1332](#)
  - NULL\_EL, [1332](#)
  - NUMBER, [1332](#)
  - STRING, [1331](#)
  - STRING\_CHUNK, [1332](#)
  - STRING\_TOO\_BIG, [1332](#)
  - TRUE\_EL, [1332](#)
  - UNDEFINED, [1331](#)
  - VALUE\_SEPERATOR, [1332](#)
- JSON Linter Test, [546](#)
- json\_primitive\_type
  - JSON Lexer, [1331](#)
- JsonAllocString, [2142](#)
- JsonLexerFDAdapter, [2142](#)
- JsonRef, [2147](#)
  - GetChildPrintSize, [2149](#)
  - IsArray, [2149](#)
  - IsBool, [2150](#)
  - IsNull, [2150](#)
  - IsNumber, [2150](#)
  - IsObject, [2150](#)
  - IsString, [2150](#)
  - MakeInvalid, [2150](#)
  - name, [2150](#)
  - object, [2151](#)
  - operator bool, [2151](#)
  - operator const char \*, [2151](#)
  - operator double, [2151](#)
  - operator float, [2152](#)
  - operator int, [2152](#)
  - operator int16\_t, [2152](#)
  - operator int32\_t, [2152](#)
  - operator int8\_t, [2152](#)
  - operator NBString, [2152](#)
  - operator time\_t, [2152](#)
  - operator uint16\_t, [2153](#)
  - operator uint32\_t, [2153](#)
  - operator uint8\_t, [2153](#)
  - operator(), [2153](#)
  - operator[], [2153](#)
  - Valid, [2154](#)
- Key Blob, [378](#)
- kFLEXCAN\_AckError
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_Bit0Error
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_Bit1Error
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_BusOffInterruptEnable
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_BusOffIntFlag
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_ClkSrc0
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_ClkSrc1
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_ClkSrcOsc
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_ClkSrcPeri
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_CrcError
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_ErrorInterruptEnable
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_ErrorIntFlag
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_FaultConfinementFlag
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_FormError
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_FrameFormatExtend
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_FrameFormatStandard
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_FrameTypeData
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_FrameTypeRemote
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_IdleFlag
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_ReceivingFlag
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_RxErrorWarningFlag
  - FlexCAN SOMRT1061, [1139](#)

- kFLEXCAN\_RxFifoFilterTypeA
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_RxFifoFilterTypeB
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_RxFifoFilterTypeC
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_RxFifoFilterTypeD
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_RxFifoFrameAvlFlag
  - FlexCAN SOMRT1061, [1138](#)
- kFLEXCAN\_RxFifoOverflowFlag
  - FlexCAN SOMRT1061, [1138](#)
- kFLEXCAN\_RxFifoPrioHigh
  - FlexCAN SOMRT1061, [1141](#)
- kFLEXCAN\_RxFifoPrioLow
  - FlexCAN SOMRT1061, [1141](#)
- kFLEXCAN\_RxFifoWarningFlag
  - FlexCAN SOMRT1061, [1138](#)
- kFLEXCAN\_RxWarningInterruptEnable
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_RxWarningIntFlag
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_StuffingError
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_SynchFlag
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_TransmittingFlag
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_TxErrorWarningFlag
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_TxWarningInterruptEnable
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_TxWarningIntFlag
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_WakeUpInterruptEnable
  - FlexCAN SOMRT1061, [1140](#)
- kFLEXCAN\_WakeUpIntFlag
  - FlexCAN SOMRT1061, [1139](#)
- kFLEXCAN\_WakeupSrcFiltered
  - FlexCAN SOMRT1061, [1141](#)
- kFLEXCAN\_WakeupSrcUnfiltered
  - FlexCAN SOMRT1061, [1141](#)
- kill
  - Utilities, [1793](#)
- kPIT\_Chnl\_0
  - Pit, [1369](#)
- kPIT\_Chnl\_1
  - Pit, [1369](#)
- kPIT\_Chnl\_2
  - Pit, [1369](#)
- kPIT\_Chnl\_3
  - Pit, [1369](#)
- kPIT\_TimerFlag
  - Pit, [1369](#)
- kPIT\_TimerInterruptEnable
  - Pit, [1369](#)
- kQTMR\_AssertWhenCountActive
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_CascadeCount
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_Channel\_0
  - QTMR (Quad Timer), [1407](#)
- kQTMR\_Channel\_1
  - QTMR (Quad Timer), [1407](#)
- kQTMR\_Channel\_2
  - QTMR (Quad Timer), [1407](#)
- kQTMR\_Channel\_3
  - QTMR (Quad Timer), [1407](#)
- kQTMR\_ClearOnCompare
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_ClockCounter0InputPin
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockCounter0Output
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockCounter1InputPin
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockCounter1Output
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockCounter2InputPin
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockCounter2Output
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockCounter3InputPin
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockCounter3Output
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockDivide\_1
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockDivide\_128
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockDivide\_16
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockDivide\_2
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockDivide\_32
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockDivide\_4
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockDivide\_64
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ClockDivide\_8
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_ComparatorPreload1DmaEnable
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_ComparatorPreload2DmaEnable
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_Compare1Flag
  - QTMR (Quad Timer), [1411](#)
- kQTMR\_Compare1InterruptEnable
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_Compare2Flag
  - QTMR (Quad Timer), [1411](#)
- kQTMR\_Compare2InterruptEnable
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_CompareFlag
  - QTMR (Quad Timer), [1410](#)

- kQTMR\_CompareInterruptEnable
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_Counter0InputPin
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_Counter1InputPin
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_Counter2InputPin
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_Counter3InputPin
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_EdgeFlag
  - QTMR (Quad Timer), [1411](#)
- kQTMR\_EdgeInterruptEnable
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_EnableGateClock
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_FallingEdge
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_ForceOutToZero
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_HaltCounter
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_HaltCountForceOutZero
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_InputEdgeFlagDmaEnable
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_LoadOnComp1
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_LoadOnComp2
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_NoCapture
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_NoOperation
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_NoPreload
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_OverflowFlag
  - QTMR (Quad Timer), [1411](#)
- kQTMR\_OverflowInterruptEnable
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_PriSrcRiseAndFallEdge
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_PriSrcRiseEdge
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_PriSrcRiseEdgeSecDir
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_PriSrcRiseEdgeSecInpHigh
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_PwmHigh
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_PwmLow
  - QTMR (Quad Timer), [1410](#)
- kQTMR\_QuadCountMode
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_RisingAndFallingEdge
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_RisingEdge
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_RunNormalInDebug
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_SecSrcTrigPriCnt
  - QTMR (Quad Timer), [1408](#)
- kQTMR\_SetOnCompare
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_SetOnCompareClearOnCountRoll
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_SetOnCompareClearOnSecSrcInp
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_ToggleOnAltCompareReg
  - QTMR (Quad Timer), [1409](#)
- kQTMR\_ToggleOnCompare
  - QTMR (Quad Timer), [1409](#)
- kStatus\_FLEXCAN\_ErrorStatus
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_RxBusy
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_RxFifoBusy
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_RxFifoDisabled
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_RxFifoldle
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_RxFifoOverflow
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_RxFifoWarning
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_RxIdle
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_RxOverflow
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_RxRemote
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_TxBusy
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_TxIdle
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_TxSwitchToRx
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_UnHandled
  - FlexCAN SOMRT1061, [1138](#)
- kStatus\_FLEXCAN\_WakeUp
  - FlexCAN SOMRT1061, [1138](#)
- kTEMPMON\_HighAlarmMode
  - SOMRT1061 Temperature Monitor, [1433](#)
- kTEMPMON\_LowAlarmMode
  - SOMRT1061 Temperature Monitor, [1433](#)
- kTEMPMON\_PanicAlarmMode
  - SOMRT1061 Temperature Monitor, [1433](#)
- lastSckToPcsDelayInNanoSec
  - \_lpspi\_master\_config, [1924](#)
- latency
  - SEMC\_cfg\_t, [2314](#)
- LEASE\_OFFERED
  - DHCP, [1894](#)
- LEASE\_OPEN
  - DHCP, [1894](#)

- LEASE\_STATIC
  - DHCP, [1894](#)
- LEASE\_TAKEN
  - DHCP, [1894](#)
- LeaseState
  - DHCP, [1894](#)
- LEAST\_SIG\_FIRST
  - SAI Configuration Enumerations, [1529](#)
- length
  - \_flexcan\_frame, [1914](#)
  - config\_string, [2023](#)
  - NBString, [2243](#)
- LetsEncryptAcmeServletObject, [2154](#)
  - LetsEncryptAcmeServletObject, [2155](#)
- linIdleState
  - SAI\_rtx\_cfg\_t, [2302](#)
  - SSC\_rtx\_cfg\_t, [2345](#)
- LinkActive
  - InterfaceBlock, [2101](#)
- LinkDuplex
  - InterfaceBlock, [2101](#)
- LinkSpeed
  - InterfaceBlock, [2101](#)
- listen
  - TCP, [1643](#)
- listen\_list\_item, [2155](#)
- ListenOn
  - Command Processor Listen Channels, [1168](#)
- listenvia
  - TCP, [1650](#), [1652](#)
- Load Application From Flash Card, [635](#)
- LoadAuthRecordsFn
  - User Authorization Manager, [1777](#)
- lookupTable
  - \_FlexSPIConfig, [1921](#)
- Low Level Processing (NetDoRx), [1346](#)
  - ClearCustomNetDoRX, [1348](#)
  - CustomNetDoRX, [1350](#)
  - NetDoRX, [1348](#)
  - netDoRXFunc, [1347](#)
  - SetCustomNetDoRX, [1349](#)
- lowAlarmTemp
  - \_tempmon\_config, [1940](#)
- lpspiSoftwareTCD
  - \_lpspi\_master\_edma\_handle, [1926](#)
  - \_lpspi\_slave\_edma\_handle, [1932](#)
- lutCustomSeqEnable
  - \_FlexSPIConfig, [1921](#)
- m\_authLevel
  - UserAuthRecord, [2426](#)
- m\_fd\_tcp
  - NB::WebSocket, [2513](#)
- m\_fd\_ws
  - NB::WebSocket, [2515](#)
- MACADR, [2155](#)
  - operator+, [2156](#)
- MacToID
  - Utilities, [1793](#)
- majorLoopCounts
  - \_edma\_transfer\_config, [1910](#)
- MakeInvalid
  - JsonRef, [2150](#)
- MakeTcpConnection
  - SerialRecord, [2316](#)
- MakeUdpConnection
  - SerialRecord, [2316](#)
- malloc, [563](#)
- Manual Configuration, [471](#)
- ManualEthernetConfig
  - Ethernet, [1209](#)
- markValidTime
  - IEEE1588Timer, [2097](#)
- mask
  - PinIO, [1246](#)
- MatchId
  - MQTT::mqtt\_leaf, [2205](#)
- maxMbNum
  - \_flexcan\_config, [1912](#)
- MaxTasks, [783](#)
- mbFrameBuf
  - \_flexcan\_handle, [1915](#)
- mbIdx
  - \_flexcan\_mb\_transfer, [1916](#)
- mbState
  - \_flexcan\_handle, [1915](#)
- MCAN Driver Internals, [1155](#)
  - MCAN\_ACKNOWLEDGE\_ERROR, [1157](#)
  - MCAN\_BIT\_ERROR, [1157](#)
  - MCAN\_BUS\_OFF, [1157](#)
  - MCAN\_CRC\_ERROR, [1157](#)
  - MCAN\_ERROR\_LOGGING\_OVERFLOW, [1157](#)
  - MCAN\_ERROR\_PASSIVE, [1157](#)
  - MCAN\_FORMAT\_ERROR, [1157](#)
  - mcan\_interrupt\_source, [1157](#)
  - MCAN\_MESSAGE\_RAM\_ACCESS\_FAILURE, [1157](#)
  - mcan\_nonmatching\_frames\_action, [1157](#)
  - MCAN\_NONMATCHING\_FRAMES\_FIFO\_0, [1159](#)
  - MCAN\_NONMATCHING\_FRAMES\_FIFO\_1, [1159](#)
  - MCAN\_NONMATCHING\_FRAMES\_REJECT, [1159](#)
  - MCAN\_RX\_BUFFER\_NEW\_MESSAGE, [1157](#)
  - MCAN\_RX\_FIFO\_0\_FULL, [1157](#)
  - MCAN\_RX\_FIFO\_0\_LOST\_MESSAGE, [1157](#)
  - MCAN\_RX\_FIFO\_0\_NEW\_MESSAGE, [1157](#)
  - MCAN\_RX\_FIFO\_0\_WATERMARK, [1157](#)
  - MCAN\_RX\_FIFO\_1\_FULL, [1157](#)
  - MCAN\_RX\_FIFO\_1\_MESSAGE\_LOST, [1157](#)
  - MCAN\_RX\_FIFO\_1\_NEW\_MESSAGE, [1157](#)
  - MCAN\_RX\_FIFO\_1\_WATERMARK, [1157](#)
  - MCAN\_RX\_HIGH\_PRIORITY\_MESSAGE, [1157](#)
  - MCAN\_STUFF\_ERROR, [1157](#)
  - MCAN\_TIMEOUT\_CONTINUES, [1159](#)
  - mcan\_timeout\_mode, [1159](#)
  - MCAN\_TIMEOUT\_OCCURRED, [1157](#)
  - MCAN\_TIMEOUT\_RX\_FIFO\_0, [1159](#)



- MCAN\_TIMEOUT\_RX\_FIFO\_1, [1159](#)
- MCAN\_TIMEOUT\_TX\_EVEN\_FIFO, [1159](#)
- MCAN\_TIMESTAMP\_COMPLETE, [1157](#)
- MCAN\_TIMESTAMP\_WRAPAROUND, [1157](#)
- MCAN\_TX\_CANCELLATION\_FINISH, [1157](#)
- MCAN\_TX\_EVENT\_FIFO\_ELEMENT\_LOST, [1157](#)
- MCAN\_TX\_EVENT\_FIFO\_FULL, [1157](#)
- MCAN\_TX\_EVENT\_FIFO\_NEW\_ENTRY, [1157](#)
- MCAN\_TX\_EVENT\_FIFO\_WATERMARK, [1157](#)
- MCAN\_TX\_FIFO\_EMPTY, [1157](#)
- MCAN\_WARNING\_STATUS, [1157](#)
- MCAN\_WATCHDOG, [1157](#)
- MCAN\_ACKNOWLEDGE\_ERROR
  - MCAN Driver Internals, [1157](#)
- MCAN\_BIT\_ERROR
  - MCAN Driver Internals, [1157](#)
- MCAN\_BUS\_OFF
  - MCAN Driver Internals, [1157](#)
- MCAN\_CRC\_ERROR
  - MCAN Driver Internals, [1157](#)
- MCAN\_ERROR\_LOGGING\_OVERFLOW
  - MCAN Driver Internals, [1157](#)
- MCAN\_ERROR\_PASSIVE
  - MCAN Driver Internals, [1157](#)
- mcan\_extended\_message\_filter\_element, [2160](#)
- MCAN\_FORMAT\_ERROR
  - MCAN Driver Internals, [1157](#)
- mcan\_interrupt\_source
  - MCAN Driver Internals, [1157](#)
- MCAN\_MESSAGE\_RAM\_ACCESS\_FAILURE
  - MCAN Driver Internals, [1157](#)
- mcan\_module
  - mcanMODM7AE70::mcan\_module, [2161](#)
- mcan\_nonmatching\_frames\_action
  - MCAN Driver Internals, [1157](#)
- MCAN\_NONMATCHING\_FRAMES\_FIFO\_0
  - MCAN Driver Internals, [1159](#)
- MCAN\_NONMATCHING\_FRAMES\_FIFO\_1
  - MCAN Driver Internals, [1159](#)
- MCAN\_NONMATCHING\_FRAMES\_REJECT
  - MCAN Driver Internals, [1159](#)
- MCAN\_RX\_BUFFER\_NEW\_MESSAGE
  - MCAN Driver Internals, [1157](#)
- mcan\_rx\_element\_buffer, [2166](#)
- MCAN\_RX\_FIFO\_0\_FULL
  - MCAN Driver Internals, [1157](#)
- MCAN\_RX\_FIFO\_0\_LOST\_MESSAGE
  - MCAN Driver Internals, [1157](#)
- MCAN\_RX\_FIFO\_0\_NEW\_MESSAGE
  - MCAN Driver Internals, [1157](#)
- MCAN\_RX\_FIFO\_0\_WATERMARK
  - MCAN Driver Internals, [1157](#)
- MCAN\_RX\_FIFO\_1\_FULL
  - MCAN Driver Internals, [1157](#)
- MCAN\_RX\_FIFO\_1\_MESSAGE\_LOST
  - MCAN Driver Internals, [1157](#)
- MCAN\_RX\_FIFO\_1\_NEW\_MESSAGE
  - MCAN Driver Internals, [1157](#)
- MCAN Driver Internals, [1157](#)
- MCAN\_RX\_FIFO\_1\_WATERMARK
  - MCAN Driver Internals, [1157](#)
- MCAN\_RX\_HIGH\_PRIORITY\_MESSAGE
  - MCAN Driver Internals, [1157](#)
- MCAN\_STUFF\_ERROR
  - MCAN Driver Internals, [1157](#)
- MCAN\_TIMEOUT\_CONTINUES
  - MCAN Driver Internals, [1159](#)
- mcan\_timeout\_mode
  - MCAN Driver Internals, [1159](#)
- MCAN\_TIMEOUT\_OCCURRED
  - MCAN Driver Internals, [1157](#)
- MCAN\_TIMEOUT\_RX\_FIFO\_0
  - MCAN Driver Internals, [1159](#)
- MCAN\_TIMEOUT\_RX\_FIFO\_1
  - MCAN Driver Internals, [1159](#)
- MCAN\_TIMEOUT\_TX\_EVEN\_FIFO
  - MCAN Driver Internals, [1159](#)
- MCAN\_TIMESTAMP\_COMPLETE
  - MCAN Driver Internals, [1157](#)
- MCAN\_TIMESTAMP\_WRAPAROUND
  - MCAN Driver Internals, [1157](#)
- MCAN\_TX\_CANCELLATION\_FINISH
  - MCAN Driver Internals, [1157](#)
- mcan\_tx\_element, [2166](#)
- mcan\_tx\_event\_element, [2166](#)
- MCAN\_TX\_EVENT\_FIFO\_ELEMENT\_LOST
  - MCAN Driver Internals, [1157](#)
- MCAN\_TX\_EVENT\_FIFO\_FULL
  - MCAN Driver Internals, [1157](#)
- MCAN\_TX\_EVENT\_FIFO\_NEW\_ENTRY
  - MCAN Driver Internals, [1157](#)
- MCAN\_TX\_EVENT\_FIFO\_WATERMARK
  - MCAN Driver Internals, [1157](#)
- MCAN\_TX\_FIFO\_EMPTY
  - MCAN Driver Internals, [1157](#)
- MCAN\_WARNING\_STATUS
  - MCAN Driver Internals, [1157](#)
- MCAN\_WATCHDOG
  - MCAN Driver Internals, [1157](#)
- mcanMODM7AE70, [1894](#)
- mcanMODM7AE70::CanRxMessage, [1957](#)
  - CanRxMessage, [1958](#)
  - CopyData, [1958](#)
  - GetData, [1958](#)
  - GetId, [1958](#)
  - GetLength, [1958](#)
  - GetNewMessage, [1959](#)
  - GetTimeStamp, [1959](#)
  - IsValid, [1959](#)
- mcanMODM7AE70::mcan\_config, [2156](#)
  - automatic\_retransmission, [2158](#)
  - clock\_stop\_acknowledge, [2158](#)
  - clock\_stop\_request, [2158](#)
  - delay\_compensation\_filter\_window\_length, [2158](#)
  - delay\_compensation\_offset, [2158](#)
  - edge\_filtering, [2158](#)

- extended\_id\_mask, [2158](#)
- nonmatching\_frames\_action\_extended, [2159](#)
- nonmatching\_frames\_action\_standard, [2159](#)
- protocol\_exception\_handling, [2159](#)
- remote\_frames\_extended\_reject, [2159](#)
- remote\_frames\_standard\_reject, [2159](#)
- run\_in\_standby, [2159](#)
- rx\_fifo\_0\_overwrite, [2159](#)
- rx\_fifo\_0\_watermark, [2159](#)
- rx\_fifo\_1\_overwrite, [2159](#)
- rx\_fifo\_1\_watermark, [2159](#)
- set\_config\_defaults, [2157](#)
- tdc\_enable, [2159](#)
- timeout\_enable, [2159](#)
- timeout\_mode, [2160](#)
- timeout\_period, [2160](#)
- timestamp\_prescaler, [2160](#)
- transmit\_pause, [2160](#)
- tx\_event\_fifo\_watermark, [2160](#)
- tx\_queue\_mode, [2160](#)
- watchdog\_configuration, [2160](#)
- mcanMODM7AE70::mcan\_module, [2160](#)
- blocking\_send\_message, [2161](#)
- init, [2163](#)
- mcan\_module, [2161](#)
- MultiCanReplaceRTRMessage, [2163](#)
- MultiCanSetRTRMessage, [2163](#)
- MultiCanStopRTRMessage, [2164](#)
- RegisterRxFifo, [2164](#)
- RegisterRxFifoMask, [2164](#)
- RegisterRxFifoRange, [2165](#)
- send\_message, [2165](#)
- UnRegisterFifo, [2165](#)
- McastMac
  - IPADDR6, [2109](#)
- MCF5441x (DSPI), [1447](#)
  - CHIP\_SELECT\_0, [1453](#)
  - CHIP\_SELECT\_1, [1453](#)
  - CHIP\_SELECT\_2, [1453](#)
  - CHIP\_SELECT\_3, [1453](#)
  - CHIP\_SELECT\_DISABLED, [1453](#)
  - CR\_CLEAR\_NON\_GRP\_PRIO, [1450](#)
  - CR\_SET\_RR\_CH\_ARB, [1450](#)
  - CS\_ASSERT\_HIGH, [1454](#)
  - CS\_ASSERT\_LOW, [1454](#)
  - csReturnType, [1453](#)
  - CTAR\_CLOCK\_PHASE, [1450](#)
  - CTAR\_CLOCK\_POLARITY, [1450](#)
  - CTAR\_FRAME\_16BIT, [1451](#)
  - CTAR\_FRAME\_4BIT, [1451](#)
  - DEASSERT\_AFTER\_LAST, [1453](#)
  - DEASSERT\_EVERY\_TRANSFER, [1453](#)
  - DEASSERT\_NEVER, [1453](#)
  - DEFAULT\_DSPI\_MODULE, [1451](#)
  - DSPI\_BUSY, [1451](#)
  - DSPI\_ERROR, [1451](#)
  - DSPI\_MODULE\_COUNT, [1451](#)
  - DSPI\_OK, [1451](#)
  - DSPIdone, [1454](#)
  - DSPInit, [1454](#)
  - DSPISStart, [1455](#)
  - MCR\_CLR\_FIFOS, [1451](#)
  - MCR\_DIS\_RXF, [1451](#)
  - MCR\_DIS\_TXF, [1451](#)
  - MCR\_HALT\_BIT, [1451](#)
  - MCR\_MASTER\_INIT, [1451](#)
  - PUSHR\_CONT\_BIT, [1452](#)
  - PUSHR\_EOQ\_BIT, [1452](#)
  - QSPIdone, [1455](#)
  - QSPInit, [1456](#)
  - QSPISStart, [1457](#)
  - RSER\_DMA\_IRQ\_ONLY, [1452](#)
  - RSER\_EOQF\_IRQ\_ONLY, [1452](#)
  - spiChipSelect, [1453](#)
  - spiChipSelectPolarity, [1453](#)
  - SR\_CLR\_FLAGS, [1452](#)
  - SR\_EOQF\_MASK, [1452](#)
  - TCD\_ATTR\_16BIT\_TRANS, [1452](#)
  - TCD\_ATTR\_8BIT\_TRANS, [1452](#)
  - TCD\_CSR\_DISABLE\_REQ, [1452](#)
  - TCD\_CSR\_DONE\_BIT, [1452](#)
  - TCD\_CSR\_DREQ\_INT\_MAJOR, [1452](#)
  - TCD\_XITER\_CNT\_MASK, [1452](#)
  - TCD\_XOFF\_0uint8\_t, [1453](#)
  - TCD\_XOFF\_1uint8\_t, [1453](#)
  - TCD\_XOFF\_2uint8\_t, [1453](#)
- MCF5441x (Queued SPI 2.x compatibility layer), [1458](#)
  - QSPIdone, [1459](#)
  - QSPInit, [1459](#)
  - QSPISStart, [1461](#)
- MCR\_CLR\_FIFOS
  - MCF5441x (DSPI), [1451](#)
- MCR\_DIS\_RXF
  - MCF5441x (DSPI), [1451](#)
- MCR\_DIS\_TXF
  - MCF5441x (DSPI), [1451](#)
- MCR\_HALT\_BIT
  - MCF5441x (DSPI), [1451](#)
- MCR\_MASTER\_INIT
  - MCF5441x (DSPI), [1451](#)
- Message Passer, [1062](#)
- MessagePasserTask JSON, [1064](#)
- Migration Guides, [41](#)
- MIME Content Types, [1440](#)
- MIMX10xx (DSPI), [1462](#)
  - CHIP\_SELECT\_0, [1464](#)
  - CHIP\_SELECT\_1, [1464](#)
  - CHIP\_SELECT\_2, [1464](#)
  - CHIP\_SELECT\_3, [1464](#)
  - CHIP\_SELECT\_DISABLED, [1464](#)
  - CS\_ASSERT\_HIGH, [1464](#)
  - CS\_ASSERT\_LOW, [1464](#)
  - csReturnType, [1463](#)
  - DEASSERT\_AFTER\_LAST, [1463](#)
  - DEASSERT\_EVERY\_TRANSFER, [1463](#)
  - DEASSERT\_NEVER, [1463](#)



- DSPIdone, [1464](#)
- DSPInit, [1464](#), [1465](#)
- DSPISStart, [1466](#)
- spiChipSelect, [1463](#)
- spiChipSelectPolarity, [1464](#)
- minorLoopBytes
  - \_edma\_transfer\_config, [1910](#)
- minorOffset
  - \_edma\_minor\_offset\_config, [1908](#)
- MOD54415, MOD54417 Platform Reference, [107](#)
- MOD54415, MOD54417, NANO54415, SB800EX, [1371](#)
- MOD5441x Factory application, [677](#)
- MOD5441x Multiple EFFS Flash Card Test, [671](#)
- MOD5441x Only, [671](#)
- MOD5441x Rapid GPIO, [690](#)
- MOD5441x USB Mass Storage Device, [696](#)
- MOD5441x, NANO54415, SB800EX, [603](#), [1153](#), [1273](#)
  - I2CMultiChannelResetPeripheral, [1275](#)
  - MultiChannel\_I2CInit, [1275](#)
  - MultiChannel\_I2CRead, [1275](#)
  - MultiChannel\_I2CReadBuf, [1276](#)
  - MultiChannel\_I2CRestart, [1276](#)
  - MultiChannel\_I2CSend, [1277](#)
  - MultiChannel\_I2CSendBuf, [1277](#)
  - MultiChannel\_I2CStart, [1278](#)
  - MultiChannel\_I2CStop, [1278](#)
- Modify Boot, [405](#)
- Modify Ethernet Interface, [407](#)
- MODM7AE70, [720](#), [1153](#)
  - IsNBIdExt, [1155](#)
  - NbToNormId, [1155](#)
- MODM7AE70 Platform Reference, [105](#)
- MODM7AE70, SBE70LC, [704](#), [1279](#), [1385](#)
- MOST\_SIG\_FIRST
  - SAI Configuration Enumerations, [1529](#)
- MoveHandlerToLast
  - HtmlPageHandler, [2083](#)
- MoveToTop
  - IPV6\_ROOT\_EL\_CONTAINER, [2127](#)
- MQTT, [1332](#), [1895](#)
  - eObj\_Flag\_DynTopicName, [1898](#)
  - eObj\_Flag\_PubOnWrite, [1898](#)
  - eObj\_Flag\_PubQoS\_AtLeastOnce, [1898](#)
  - eObj\_Flag\_PubQoS\_ExactlyOnce, [1898](#)
  - eObj\_Flag\_PubRetain, [1898](#)
  - eObj\_Flag\_Sub\_HandlerOnly, [1898](#)
  - eObj\_Flag\_Sub\_IncludeLocal, [1898](#)
  - eObj\_Flag\_Sub\_NoLocal, [1898](#)
  - eObj\_Flag\_Sub\_QoS\_AtLeastOnce, [1898](#)
  - eObj\_Flag\_Sub\_QoS\_ExactlyOnce, [1898](#)
  - eObj\_Flag\_Sub\_Retained\_IfNew, [1898](#)
  - eObj\_Flag\_Sub\_Retained\_Never, [1898](#)
  - eObj\_Flag\_Subscribe, [1898](#)
  - eObj\_Flag\_WaitForPub, [1898](#)
  - eObj\_Serialize\_BigEndian, [1898](#)
  - eObj\_Serialize\_CURRENT\_FORM, [1899](#)
  - eObj\_Serialize\_Disabled, [1898](#)
  - eObj\_Serialize\_JSON, [1898](#)
  - eObj\_Serialize\_JSON\_ValOnly, [1898](#)
  - eObj\_Serialize\_LittleEndian, [1899](#)
  - eObj\_Serialize\_MqttString, [1898](#)
  - eObj\_Serialize\_NativeEndian, [1899](#)
  - eObj\_Serialize\_String, [1898](#)
  - eObj\_Serialize\_t, [1898](#)
  - eResult\_Cont\_Auth, [1900](#)
  - eResult\_Error\_BadArg, [1900](#)
  - eResult\_Error\_BadProp, [1900](#)
  - eResult\_Error\_InvalidURI, [1900](#)
  - eResult\_Error\_Malformed, [1900](#)
  - eResult\_Error\_NetErr, [1900](#)
  - eResult\_Error\_NoneAvail, [1900](#)
  - eResult\_Error\_NoSuchConn, [1900](#)
  - eResult\_Error\_Timeout, [1900](#)
  - eResult\_Mqtt\_Admin, [1899](#)
  - eResult\_Mqtt\_BadAuthMethod, [1899](#)
  - eResult\_Mqtt\_Banned, [1899](#)
  - eResult\_Mqtt\_ClientIdInvalid, [1899](#)
  - eResult\_Mqtt\_ConnRate, [1899](#)
  - eResult\_Mqtt\_FilterInvalid, [1899](#)
  - eResult\_Mqtt\_Implementation, [1899](#)
  - eResult\_Mqtt\_KATimeout, [1899](#)
  - eResult\_Mqtt\_MalformedPkt, [1899](#)
  - eResult\_Mqtt\_MaxConnTime, [1900](#)
  - eResult\_Mqtt\_MsgRate, [1899](#)
  - eResult\_Mqtt\_NameInvalid, [1899](#)
  - eResult\_Mqtt\_NotAuthorized, [1899](#)
  - eResult\_Mqtt\_NotSupported\_QoS, [1899](#)
  - eResult\_Mqtt\_NotSupported\_Retain, [1899](#)
  - eResult\_Mqtt\_NotSupported\_SharedSubs, [1899](#)
  - eResult\_Mqtt\_NotSupported\_SubIds, [1900](#)
  - eResult\_Mqtt\_NotSupported\_WildSubs, [1900](#)
  - eResult\_Mqtt\_PayFmt\_Invalid, [1899](#)
  - eResult\_Mqtt\_PkId\_InUse, [1899](#)
  - eResult\_Mqtt\_PkId\_NotFound, [1899](#)
  - eResult\_Mqtt\_Protocol, [1899](#)
  - eResult\_Mqtt\_Quota, [1899](#)
  - eResult\_Mqtt\_RcvMax\_Exceeded, [1899](#)
  - eResult\_Mqtt\_ServerBusy, [1899](#)
  - eResult\_Mqtt\_ServerMoved, [1899](#)
  - eResult\_Mqtt\_ServerShutdown, [1899](#)
  - eResult\_Mqtt\_ServerUnavail, [1899](#)
  - eResult\_Mqtt\_SessTakeover, [1899](#)
  - eResult\_Mqtt\_TooLarge, [1899](#)
  - eResult\_Mqtt\_TopicAlias\_Invalid, [1899](#)
  - eResult\_Mqtt\_Transport\_Error, [1900](#)
  - eResult\_Mqtt\_Unspecified, [1899](#)
  - eResult\_Mqtt\_UnsupportedProtoVer, [1899](#)
  - eResult\_Mqtt\_UseOther, [1899](#)
  - eResult\_Mqtt\_UserPassword, [1899](#)
  - eResult\_Socket\_NeedDNS, [1900](#)
  - eResult\_Socket\_TryAgain, [1900](#)
  - eResult\_Success, [1899](#)
  - eResult\_t, [1899](#)
  - GetConnectionStatusString, [1900](#)
  - GetPacketTypeString, [1900](#)
  - GetReturnCodeString, [1900](#)

- MsgCallback\_t, 1897
- MQTT Config Object Example, 566
- MQTT Custom Object Example, 569
- MQTT Examples, 565
- MQTT Object I/O Example, 571
- MQTT Simple Object Pub/Sub Example, 574
- MQTT SubHandler GPIO Example, 576
- MQTT::Client, 1960
  - Client, 1964
  - Connect, 1965
  - Disconnect, 1967
  - eConnectionStatus, 1963
  - eConnStat\_Error, 1963
  - eConnStat\_Established, 1963
  - eConnStat\_InProgress, 1963
  - eConnStat\_No\_Broker\_Configured, 1963
  - eConnStat\_Refused, 1963
  - eConnStat\_Resolving, 1963
  - eLimit\_InFlight\_Rx, 1964
  - eLimit\_InFlight\_Tx, 1963
  - eLimit\_PublishAliases, 1964
  - eLimit\_QoS, 1964
  - eLimit\_SessionExpiry, 1964
  - eLimit\_Size\_Rx, 1963
  - eLimit\_Size\_Tx, 1963
  - eLimit\_SubAliases, 1964
  - eLimit\_Subscriptions, 1964
  - eRateLimit\_Publish, 1963
  - eRateLimit\_Subscribe, 1963
  - eRateLimitAdj\_Add, 1963
  - eRateLimitAdj\_None, 1963
  - eRateLimitAdj\_Set, 1963
  - eRateLimitAdj\_Subtract, 1963
  - eRateLimitAdjustMethod, 1963
  - eRateLimitType, 1963
  - eSessionLimit, 1963
  - GetClientCert, 1967
  - GetConnectionStatus, 1967
  - GetFirstClient, 1967
  - Publish, 1967, 1968
  - QueueAllSubscriptions, 1969
  - QueuePublish, 1969
  - QueueSubscribe, 1970
  - QueueUnsubscribe, 1970, 1971
  - RegisterMqttObj, 1971
  - RegisterTopicHandler, 1971
  - RemoveObj, 1972
  - RemoveTopicHandler, 1972
  - SendNextPubReq, 1972
  - SendNextSubReq, 1973
  - SetClientCert, 1973
  - SetConnectNotify, 1973
  - SetRateLimit, 1973
  - SetRateLimit\_Publish, 1974
  - SetRateLimit\_Subscribe, 1974
  - StartTask, 1975
  - Subscribe, 1975
  - Unsubscribe, 1976
- MQTT::ConfigExposer, 2038
  - ConfigExposer, 2041
  - eRender\_Hidden, 2040
  - eRender\_ModifiedOnly, 2040
  - eRender\_Pretty, 2040
  - eRenderFlag\_t, 2040
  - GetPubTopicName, 2041
  - GetRenderLength, 2041
  - GetSubTopicFilter, 2042
  - ParseFd, 2042
  - RenderToBuffer, 2042
  - RenderToFd, 2043
  - SubscribeEvent, 2043
  - TestOrSetValue, 2043
- MQTT::ConfiguredClient, 2044
  - ConfiguredClient, 2046
- MQTT::ConnectRequest, 2047
- MQTT::mqtt\_bool, 2166
  - mqtt\_bool, 2169, 2170
  - operator bool, 2170
  - operator!, 2170
  - operator=, 2170
  - ParseFd, 2171
  - RenderToBuffer, 2171
  - RenderToFd, 2171
  - SubscribeEvent, 2172
  - TestOrSetValue, 2172
- MQTT::mqtt\_float, 2172
  - mqtt\_float, 2175, 2176
  - operator double, 2176
  - operator=, 2176
  - ParseFd, 2177
  - RenderToBuffer, 2177
  - RenderToFd, 2177
  - SetStringFormat, 2177
  - SubscribeEvent, 2178
  - TestOrSetValue, 2178
- MQTT::mqtt\_int, 2178
  - mqtt\_int, 2181, 2182
  - operator int, 2182
  - operator++, 2182, 2183
  - operator--, 2183
  - operator=, 2183
  - ParseFd, 2183
  - RenderToBuffer, 2184
  - RenderToFd, 2184
  - SubscribeEvent, 2184
  - TestOrSetValue, 2185
- MQTT::mqtt\_IO, 2185
  - ClearFlags, 2189
  - mqtt\_IO, 2188
  - operator bool, 2189
  - operator!, 2189
  - operator=, 2189
  - ParseFd, 2189
  - PinNotify, 2189
  - RenderToBuffer, 2190
  - RenderToFd, 2190

- SetFixedDirection, 2190
- SetFlags, 2191
- SubscribeEvent, 2191
- TestOrSetValue, 2191
- MQTT::mqtt\_IO\_Input, 2192
  - mqtt\_IO\_Input, 2194, 2195
  - operator=, 2195
- MQTT::mqtt\_IO\_Output, 2195
  - mqtt\_IO\_Output, 2198
  - operator=, 2199
- MQTT::mqtt\_Leaf, 2199
  - AddChild, 2202
  - Assign, 2202
  - ClearFlags, 2203
  - ForEachLeaf, 2203
  - GetChildren, 2203
  - GetFlags, 2203
  - getPriority, 2203
  - GetPubTopicName, 2203
  - GetRenderLength, 2204
  - GetSerializationForm, 2204
  - GetSubTopicFilter, 2204
  - HandlePubAck, 2204
  - HandleSubAck, 2205
  - HandleUnsubAck, 2205
  - MatchId, 2205
  - mqtt\_Leaf, 2202
  - MqttRxPublish, 2205
  - NameMatch, 2206
  - ParseFd, 2206
  - Publish, 2206
  - PubState, 2207
  - QueuePublish, 2207
  - QueueSubscribe, 2207
  - QueueUnsubscribe, 2207
  - RemoveFirstChild, 2207
  - RenderToBuffer, 2208
  - RenderToFd, 2208
  - RootParse, 2208
  - SetFlags, 2208
  - SetPublishInterval, 2209
  - SetPubState, 2209
  - SetSerializationForm, 2209
  - SetSubState, 2209
  - SubscribeEvent, 2210
  - SubState, 2210
  - TestOrSetValue, 2210
- MQTT::mqtt\_obj, 2211
  - AddChild, 2214
  - FindChild, 2214
  - GetChildren, 2214
  - GetRenderLength, 2214
  - GetSubTopicFilter, 2215
  - mqtt\_obj, 2213
  - ParseFd, 2215
  - RemoveFirstChild, 2215
  - RenderToBuffer, 2215
  - RenderToFd, 2216
- SetPubState, 2216
- SetSubState, 2216
- SubscribeEvent, 2216
- TestOrSetValue, 2217
- MQTT::mqtt\_string, 2217
  - mqtt\_string, 2220, 2221
  - operator NBString, 2222
  - operator=, 2222
  - operator[], 2222, 2223
  - ParseFd, 2223
  - RenderToBuffer, 2223
  - RenderToFd, 2223
  - SubscribeEvent, 2224
  - TestOrSetValue, 2224
- MQTT::mqtt\_uint, 2224
  - mqtt\_uint, 2227, 2228
  - operator unsigned, 2228
  - operator++, 2228, 2229
  - operator--, 2229
  - operator=, 2229
  - ParseFd, 2229
  - RenderToBuffer, 2230
  - RenderToFd, 2230
  - SubscribeEvent, 2230
  - TestOrSetValue, 2231
- MQTT::PacketInfo, 1901
- MQTT::PacketInfo::Msg, 2231
  - GetFlags, 2231
  - GetType, 2231
  - Msg, 2231
- MQTT::PublishRequest, 2298
- MQTT::SubHandler, 2355
  - ParseFd, 2358
  - RenderToBuffer, 2358
  - RenderToFd, 2358
  - SubHandler, 2357
  - SubscribeEvent, 2358
  - TestOrSetValue, 2360
- MQTT::SubscribeFilter, 2360
- MQTT::SubscribeFilter\_NbStr, 2360
- MQTT::SubscribeRequest, 2361
- MQTT::TopicHandler, 2364
- mqtt\_bool
  - MQTT::mqtt\_bool, 2169, 2170
- mqtt\_float
  - MQTT::mqtt\_float, 2175, 2176
- mqtt\_int
  - MQTT::mqtt\_int, 2181, 2182
- mqtt\_IO
  - MQTT::mqtt\_IO, 2188
- mqtt\_IO\_Input
  - MQTT::mqtt\_IO\_Input, 2194, 2195
- mqtt\_IO\_Output
  - MQTT::mqtt\_IO\_Output, 2198
- mqtt\_Leaf
  - MQTT::mqtt\_Leaf, 2202
- mqtt\_obj
  - MQTT::mqtt\_obj, 2213

- mqtt\_string
  - MQTT::mqtt\_string, [2220](#), [2221](#)
- mqtt\_uint
  - MQTT::mqtt\_uint, [2227](#), [2228](#)
- MqttRxBroadcast
  - MQTT::mqtt\_leaf, [2205](#)
- Msg
  - MQTT::PacketInfo::Msg, [2231](#)
- MsgCallback\_t
  - MQTT, [1897](#)
- MultiCanReplaceRTRMessage
  - mcanMODM7AE70::mcan\_module, [2163](#)
- MultiCanSetRTRMessage
  - mcanMODM7AE70::mcan\_module, [2163](#)
- MultiCanStopRTRMessage
  - mcanMODM7AE70::mcan\_module, [2164](#)
- Multicast, [584](#), [1333](#)
  - RegisterMulticastFifo4, [1333](#)
  - RegisterMulticastFifo6, [1334](#)
  - UnregisterMulticastFifo4, [1334](#)
  - UnregisterMulticastFifo6, [1334](#)
- MultiChannel\_I2CInit
  - MOD5441x, NANO54415, SB800EX, [1275](#)
- MultiChannel\_I2CRead
  - MOD5441x, NANO54415, SB800EX, [1275](#)
- MultiChannel\_I2CReadBuf
  - MOD5441x, NANO54415, SB800EX, [1276](#)
- MultiChannel\_I2CRestart
  - MOD5441x, NANO54415, SB800EX, [1276](#)
- MultiChannel\_I2CSend
  - MOD5441x, NANO54415, SB800EX, [1277](#)
- MultiChannel\_I2CSendBuf
  - MOD5441x, NANO54415, SB800EX, [1277](#)
- MultiChannel\_I2CStart
  - MOD5441x, NANO54415, SB800EX, [1278](#)
- MultiChannel\_I2CStop
  - MOD5441x, NANO54415, SB800EX, [1278](#)
- multidrv
  - PinIO, [2292](#)
- Multihome, [586](#)
- Multihome and VLAN, [1335](#)
  - AddInterface, [1335](#), [1336](#)
  - AddVlanInterface, [1336](#), [1337](#)
- Multiple Tasks, [466](#)
- Multiple TCP Interface Test, [929](#)
- Mute
  - WM8904, [2530](#)
- MuteMic
  - WM8904, [2530](#)
- Mux
  - SEMC, [1424](#)
- muxMode
  - SEMC\_cfg\_t, [2314](#)
- MyDNSAddress
  - IPv6Interface, [2138](#)
- MySourceAddress
  - IPv6Interface, [2138](#)
- NAME
  - JSON Lexer, [1331](#)
- Name
  - TimeZoneRecord, [2363](#)
- name
  - JsonRef, [2150](#)
  - ParsedJsonDataSet, [2277](#)
- NameMatch
  - MQTT::mqtt\_leaf, [2206](#)
- NANO54415 Factory App, [703](#)
- NANO54415 Only, [703](#)
- NANO54415 Platform Reference, [108](#)
- NB Approve Shutdown, [589](#)
- NB::Error, [1901](#)
  - BusTimeout, [1902](#)
  - GeneralErrors, [1901](#)
  - InvalidArgument, [1902](#)
  - InvalidRequest, [1902](#)
  - NoError, [1902](#)
  - Timeout, [1902](#)
  - TooManyPendingCommands, [1902](#)
- NB::Error::Connect, [1902](#)
  - AlreadyConnected, [1902](#)
  - BSSID\_NotFound, [1902](#)
  - Cipher\_NotFound, [1902](#)
  - ConnectErrors, [1902](#)
  - ConnectFailed, [1902](#)
  - CouldNotConfig, [1902](#)
  - NotInitialized, [1902](#)
  - Option, [1902](#)
  - Sec\_NotFound, [1902](#)
  - SSID\_NotFound, [1902](#)
  - Success, [1902](#)
- NB::Error::Init, [1902](#)
  - AlreadyInit, [1903](#)
  - DevFirmVer, [1903](#)
  - DevHwVer, [1903](#)
  - InitializationErrors, [1903](#)
  - InvalidInfo, [1903](#)
  - NoDevice, [1903](#)
  - OptionTables, [1903](#)
  - Success, [1903](#)
- NB::Error::Scan, [1903](#)
  - InProgress, [1903](#)
  - NotInitialized, [1903](#)
  - Option, [1903](#)
  - ScanErrors, [1903](#)
  - Success, [1903](#)
- NB::WebSocket, [2426](#)
  - cloption, [2428](#)
  - Connect, [2430](#), [2435](#), [2438](#)
  - DumpSock, [2439](#)
  - DumpSockets, [2442](#)
  - Flush, [2445](#)
  - GetFlushTime, [2448](#)
  - GetFreeWebSocketCount, [2450](#)
  - getoption, [2453](#)
  - GetPingReplyTick, [2457](#)
  - GetState, [2460](#)

- GetWebSocketRecord, [2463](#)
- GetWriteSpace, [2463](#)
- m\_fd\_tcp, [2513](#)
- m\_fd\_ws, [2515](#)
- Ping, [2467](#)
- Pong, [2470](#)
- promote\_tcp\_ws, [2473](#)
- ReadData, [2475](#)
- RunSkippedCallBack, [2479](#)
- s\_openSocketCount, [2517](#)
- setoption, [2480](#)
- socket\_crit, [2519](#)
- WaitForPingReply, [2483](#)
- WriteData, [2486](#)
- ws\_cloption, [2491](#)
- ws\_externalclose, [2492](#)
- ws\_flush, [2494](#)
- ws\_getoption, [2496](#)
- ws\_read, [2498](#)
- ws\_read\_notify, [2501](#)
- ws\_readwto, [2503](#)
- ws\_setoption, [2506](#)
- ws\_write, [2507](#)
- ws\_write\_notify, [2511](#)
- WSSockStructs, [2522](#)
- NB::Wifi::driverStatusStruct, [2061](#)
- NBApproveShutdown
  - Shutdown Notifications, [1520](#)
- NBEclipse, [55](#)
- NBFaultNotify
  - Shutdown Notifications, [1522](#)
- NBInterpolatedString, [2232](#)
  - Append, [2235](#)
  - FdAppend, [2236](#)
  - Interpolate, [2236](#)
  - NBInterpolatedString, [2234](#), [2235](#)
- NBRTOS Real Time Operating System, [1338](#)
- NbSshInit
  - SSH, [1478](#)
- NBString, [2236](#)
  - Append, [2239](#)
  - c\_str, [2239](#)
  - clear, [2240](#)
  - compare, [2240](#)
  - copy, [2240](#)
  - empty, [2241](#)
  - FdAppend, [2241](#)
  - find, [2241](#), [2242](#)
  - Interpolate, [2242](#), [2243](#)
  - length, [2243](#)
  - NBString, [2238](#), [2239](#)
  - replace, [2243](#), [2244](#)
  - Reserve, [2244](#)
  - siprintf, [2244](#)
  - size, [2245](#)
  - sprintf, [2245](#)
  - stod, [2245](#)
  - stoi, [2245](#)
  - stol, [2245](#)
  - stoui, [2246](#)
  - stoul, [2246](#)
  - strcpy, [2246](#)
  - substr, [2246](#)
  - swap, [2247](#)
  - to\_ipaddr, [2246](#)
  - vsiprintf, [2247](#)
- NBString - NetBurner String Class, [1338](#)
- NBString Class, [591](#)
- NbToNormId
  - MODM7AE70, [1155](#)
- NBUpdate Function Return Values, [1524](#)
- NBYTES
  - \_edma\_tcd, [1909](#)
- nbytes
  - \_lpspi\_master\_edma\_handle, [1926](#)
  - \_lpspi\_slave\_edma\_handle, [1932](#)
- ncs\_rd\_pulse
  - EBI - External Bus Interface, [1401](#)
- ncs\_rd\_setup
  - EBI - External Bus Interface, [1401](#)
- ncs\_wr\_pulse
  - EBI - External Bus Interface, [1401](#)
- ncs\_wr\_setup
  - EBI - External Bus Interface, [1401](#)
- Nesting Config Objects, [396](#)
- NetBurner 3.5.6, [1](#)
- NetBurner Demo Server for JSON Client Examples, [538](#)
- NetBurner RTOS, [240](#)
- NetDoRX
  - Low Level Processing (NetDoRx), [1348](#)
- netDoRXFunc
  - Low Level Processing (NetDoRx), [1347](#)
- Network Interfaces, [1338](#)
  - DisableMulticast, [1340](#)
  - EnableMulticast, [1340](#)
  - GetFirstInterface, [1340](#)
  - GetInterfaceBlock, [1340](#)
  - GetInterfaceForMyAddress4, [1341](#)
  - GetInterfaceLink, [1341](#)
  - GetInterfaceNumber, [1341](#)
  - GetNextInterface, [1341](#)
  - InterfaceLinkActive, [1342](#)
  - InterfaceLinkDuplex, [1342](#)
  - InterfaceLinkSpeed, [1342](#)
  - InterfaceMAC, [1343](#)
  - InterfaceName, [1343](#)
  - Removeinterface, [1343](#)
- Network Protocols TCP and UDP, [256](#)
- Network Time Server (NTP) Client, [592](#)
- NewUri
  - ParsedURI, [2283](#)
- NO\_AUTOMATIC\_2ND\_ETHERNET
  - Ethernet, [1208](#)
- NoBlockConnect
  - TCP, [1660](#)
- NoBlockConnectVia

- TCP, [1665](#)
- NoBlockConnectwlocal
  - TCP, [1670](#)
- NoContentResponse
  - HTTP and HTML Functions, [1264](#)
- NoDevice
  - NB::Error::Init, [1903](#)
- NoError
  - NB::Error, [1902](#)
- nonmatching\_frames\_action\_extended
  - mcanMODM7AE70::mcan\_config, [2159](#)
- nonmatching\_frames\_action\_standard
  - mcanMODM7AE70::mcan\_config, [2159](#)
- NotAvailableResponse
  - HTTP and HTML Functions, [1264](#)
- NOTFOUND
  - JSON Lexer, [1332](#)
- NotFoundResponse
  - HTTP and HTML Functions, [1265](#)
- NotifyReachable
  - IPv6Interface, [2138](#)
- NotifyUnreachable
  - IPv6Interface, [2138](#)
- NotInitialized
  - NB::Error::Connect, [1902](#)
  - NB::Error::Scan, [1903](#)
- NotNull
  - config\_IPADDR, [1998](#)
  - config\_IPADDR4, [2001](#)
  - CUR\_IPADDR4, [2048](#)
  - IPADDR4, [2105](#)
  - IPADDR6, [2109](#)
- Now
  - NTPServer::NtpTimeStamp, [2249](#)
- nr
  - aes\_context, [1942](#)
- nrd\_cycles
  - EBI - External Bus Interface, [1402](#)
- nrd\_pulse
  - EBI - External Bus Interface, [1402](#)
- nrd\_setup
  - EBI - External Bus Interface, [1402](#)
- NTP & Real-Time Clock, [595](#), [724](#)
- NTPServer, [2247](#)
  - getPort, [2248](#)
  - isRunning, [2248](#)
  - NTPServer, [2248](#)
  - serverTask, [2248](#)
  - start, [2248](#)
  - stop, [2248](#)
- NTPServer::NtpTimeStamp, [2249](#)
  - Now, [2249](#)
  - NtpTimeStamp, [2249](#)
  - operator=, [2249](#), [2250](#)
  - SetFrom, [2250](#)
- NtpTimeStamp
  - NTPServer::NtpTimeStamp, [2249](#)
- NULL\_EL
  - JSON Lexer, [1332](#)
- NullIP
  - IPADDR6, [2109](#)
- NUMBER
  - JSON Lexer, [1332](#)
- nWait
  - EBI - External Bus Interface, [1402](#)
- nwe\_cycles
  - EBI - External Bus Interface, [1402](#)
- nwe\_pulse
  - EBI - External Bus Interface, [1403](#)
- nwe\_setup
  - EBI - External Bus Interface, [1403](#)
- object
  - JsonRef, [2151](#)
  - ParsedJsonDataSet, [2278](#)
- OkToListen
  - SerialRecord, [2316](#)
- On-board Cert Generation - Advanced, [845](#)
- On-board Cert Generation - Compiled Certificate Authority, [849](#)
- On-board Cert Generation - Simple, [842](#)
- On-Chip Temperature Sensor (TEMPMON), [776](#)
- Onboard Self-Signed Certificate Generation, [1510](#)
  - ALT\_NAME\_TYPE\_DNS, [1511](#)
  - ALT\_NAME\_TYPE\_IP, [1511](#)
  - ALT\_NAME\_TYPE\_NONE, [1511](#)
  - AltNameType\_t, [1511](#)
  - CERT\_GEN\_RETURN\_CERT\_CREATE\_ERROR, [1511](#)
  - CERT\_GEN\_RETURN\_CERT\_INIT\_ERROR, [1511](#)
  - CERT\_GEN\_RETURN\_CERT\_SAVING\_ERROR, [1511](#)
  - CERT\_GEN\_RETURN\_CERT\_SIGNING\_ERROR, [1511](#)
  - CERT\_GEN\_RETURN\_ECC\_CREATE\_ERROR, [1511](#)
  - CERT\_GEN\_RETURN\_ECC\_INIT\_ERROR, [1511](#)
  - CERT\_GEN\_RETURN\_INVALID\_RANDOM, [1511](#)
  - CERT\_GEN\_RETURN\_KEY\_CONVERT\_ERROR, [1512](#)
  - CERT\_GEN\_RETURN\_KEY\_SAVING\_ERROR, [1512](#)
  - CERT\_GEN\_RETURN\_MALLOC\_ERROR, [1511](#)
  - CERT\_GEN\_RETURN\_NOT\_REQUIRED, [1511](#)
  - CERT\_GEN\_RETURN\_RSA\_CREATE\_ERROR, [1511](#)
  - CERT\_GEN\_RETURN\_RSA\_INIT\_ERROR, [1511](#)
  - CERT\_GEN\_RETURN\_SET\_ISSUER\_ERROR, [1512](#)
  - CERT\_GEN\_RETURN\_SUCCESS, [1511](#)
  - CertGenReturnCode, [1511](#)
  - EnableOnboardCertificateCreation, [1512](#)
  - FillInAltNamesASN1, [1512](#)
  - GetDataForCertGen, [1513](#)
  - SSL\_CreateNewSelfSignedCert, [1513](#)
  - SSL\_KEY\_ECC\_SECP192R1, [1512](#)



- SSL\_KEY\_ECC\_SECP224R1, [1512](#)
- SSL\_KEY\_ECC\_SECP256R1, [1512](#)
- SSL\_KEY\_ECC\_SECP384R1, [1512](#)
- SSL\_KEY\_ECC\_SECP521R1, [1512](#)
- SSL\_KEY\_RSA\_1024, [1512](#)
- SSL\_KEY\_RSA\_2048, [1512](#)
- SSL\_KEY\_RSA\_4096, [1512](#)
- SslKeyType\_t, [1512](#)
- OnLink
  - IPV6\_PREFIX, [2124](#)
  - IPv6Interface, [2139](#)
- onReceive
  - TwoWire, [2368](#)
- onRequest
  - TwoWire, [2368](#)
- op\_code
  - ARP - Address Resolution Protocol, [1117](#)
- Open All Serial Ports, [725](#)
- Open All UARTs, [723](#), [735](#)
- OpenDefaultSerial
  - Serial Interfaces, [1516](#)
- OpenListenPort
  - SerialRecord, [2316](#)
- OpenSerial
  - Serial Interfaces, [1516](#)
- OpenSerialPort
  - SerialRecord, [2317](#)
- operator bool
  - config\_bool, [1978](#)
  - JsonRef, [2151](#)
  - MQTT::mqtt\_bool, [2170](#)
  - MQTT::mqtt\_IO, [2189](#)
  - PinIO, [2292](#)
- operator const char \*
  - JsonRef, [2151](#)
- operator double
  - config\_double, [1986](#)
  - JsonRef, [2151](#)
  - MQTT::mqtt\_float, [2176](#)
- operator float
  - config\_double, [1986](#)
  - JsonRef, [2152](#)
- operator int
  - config\_double, [1986](#)
  - config\_int, [1991](#)
  - JsonRef, [2152](#)
  - MQTT::mqtt\_int, [2182](#)
- operator int16\_t
  - JsonRef, [2152](#)
- operator int32\_t
  - JsonRef, [2152](#)
- operator int8\_t
  - JsonRef, [2152](#)
- operator IPADDR
  - config\_IPADDR, [1998](#)
- operator IPADDR4
  - config\_IPADDR4, [2003](#)
  - CUR\_IPADDR4, [2048](#)
- operator MACADR
  - config\_MACADR, [2011](#)
- operator NBString
  - config\_chooser, [1983](#)
  - config\_localname, [2006](#)
  - config\_pass, [2016](#)
  - config\_string, [2024](#)
  - JsonRef, [2152](#)
  - MQTT::mqtt\_string, [2222](#)
- operator time\_t
  - config\_time\_t, [2031](#)
  - JsonRef, [2152](#)
- operator uint16\_t
  - JsonRef, [2153](#)
- operator uint32\_t
  - \_PinVector, [1938](#)
  - config\_uint, [2034](#)
  - JsonRef, [2153](#)
- operator uint8\_t
  - JsonRef, [2153](#)
- operator unsigned
  - MQTT::mqtt\_uint, [2228](#)
- operator!
  - MQTT::mqtt\_bool, [2170](#)
  - MQTT::mqtt\_IO, [2189](#)
  - PinIO, [2292](#)
- operator()
  - JsonRef, [2153](#)
  - ParsedJsonDataSet, [2278](#)
- operator+
  - MACADR, [2156](#)
- operator++
  - MQTT::mqtt\_int, [2182](#), [2183](#)
  - MQTT::mqtt\_uint, [2228](#), [2229](#)
- operator--
  - MQTT::mqtt\_int, [2183](#)
  - MQTT::mqtt\_uint, [2229](#)
- operator=
  - \_PinVector, [1938](#)
  - config\_bool, [1978](#), [1979](#)
  - config\_chooser, [1983](#)
  - config\_double, [1986](#)
  - config\_double\_limit, [1989](#)
  - config\_int, [1991](#)
  - config\_int\_limit, [1994](#)
  - config\_IPADDR, [1998](#)
  - config\_IPADDR4, [2003](#)
  - config\_localname, [2006](#), [2007](#)
  - config\_MACADR, [2011](#)
  - config\_pass, [2017](#)
  - config\_pass\_limit, [2020](#)
  - config\_string, [2024](#)
  - config\_string\_limit, [2028](#)
  - config\_time\_t, [2031](#), [2032](#)
  - config\_uint, [2034](#)
  - config\_uint\_limit, [2036](#)
  - MQTT::mqtt\_bool, [2170](#)
  - MQTT::mqtt\_float, [2176](#)

- MQTT::mqtt\_int, [2183](#)
- MQTT::mqtt\_IO, [2189](#)
- MQTT::mqtt\_IO\_Input, [2195](#)
- MQTT::mqtt\_IO\_Output, [2199](#)
- MQTT::mqtt\_string, [2222](#)
- MQTT::mqtt\_uint, [2229](#)
- NTPServer::NtpTimeStamp, [2249](#), [2250](#)
- PinIO, [2292](#), [2293](#)
- PinVector< n >, [2297](#)
- operator[]
  - \_PinVector, [1938](#)
  - config\_string, [2024](#)
  - JsonRef, [2153](#)
  - MQTT::mqtt\_string, [2222](#), [2223](#)
  - ParsedJsonDataSet, [2278](#)
- Option
  - NB::Error::Connect, [1902](#)
  - NB::Error::Scan, [1903](#)
- OptionTables
  - NB::Error::Init, [1903](#)
- OSCrit - Critical Section, [786](#)
- OSFifo - FIFO, [788](#)
- OSFlags, [790](#)
- OSMailbox, [793](#)
- OSQueue, [794](#)
- OSSemaphore, [795](#)
- OSTaskCreate, [795](#)
- outbyte
  - Utilities, [1793](#)
- Overload Directory & System Files, [598](#)
- OverrideConfig
  - Configuration Variables, [1172](#)
- PAC193X, [506](#)
- panicAlarmTemp
  - \_tempmon\_config, [1940](#)
- parity\_mode
  - Serial Interfaces, [1515](#)
- PARP
  - ARP - Address Resolution Protocol, [1107](#)
- Parse Test, [535](#)
- ParsedJsonDataSet, [2250](#)
  - Add, [2256–2258](#)
  - AddArrayElement, [2258–2260](#)
  - AddArrayStart, [2260](#)
  - AddMyMac, [2260](#)
  - AddNull, [2260](#)
  - AddObjectStart, [2261](#)
  - CopyCurrentString, [2261](#)
  - CopyObject, [2261](#)
  - CurrentBool, [2261](#)
  - CurrentName, [2262](#)
  - CurrentNumber, [2262](#)
  - CurrentString, [2262](#)
  - FindAndCopyFullNameString, [2262](#)
  - FindAndCopyGlobalString, [2264](#)
  - FindAndCopyString, [2264](#), [2265](#)
  - FindBoolean, [2265](#)
  - FindBooleanInCurentObject, [2265](#)
  - FindElementAfterName, [2266](#)
  - FindElementAfterNameInCurrentArray, [2266](#)
  - FindElementAfterNameInCurrentObject, [2266](#)
  - FindFullAtName, [2267](#)
  - FindFullName, [2267](#)
  - FindFullNameBoolean, [2267](#)
  - FindFullNameNumber, [2268](#)
  - FindFullNamePermissiveBoolean, [2268](#)
  - FindFullNameString, [2268](#)
  - FindGlobalBoolean, [2268](#)
  - FindGlobalElementAfterName, [2269](#)
  - FindGlobalNumber, [2269](#)
  - FindGlobalObject, [2269](#)
  - FindGlobalPermissiveBoolean, [2270](#)
  - FindGlobalString, [2270](#)
  - FindNumber, [2270](#)
  - FindNumberInCurentObject, [2271](#)
  - FindObject, [2271](#)
  - FindObjectInCurentObject, [2271](#)
  - FindPermissiveBoolean, [2271](#)
  - FindPermissiveBooleanInCurentObject, [2272](#)
  - FindString, [2272](#)
  - FindStringInCurentObject, [2272](#)
  - GetCurrent, [2273](#)
  - GetFirst, [2273](#)
  - GetNext, [2273](#)
  - GetNextArray, [2274](#)
  - GetNextBoolInCurrentArray, [2274](#)
  - GetNextName, [2274](#)
  - GetNextNameInCurrentArray, [2275](#)
  - GetNextNameInCurrentObject, [2275](#)
  - GetNextNumberInCurrentArray, [2275](#)
  - GetNextObject, [2276](#)
  - GetNextObjectInCurrentArray, [2276](#)
  - GetNextStringInCurrentArray, [2276](#)
  - GetParsePosition, [2277](#)
  - GetPrintSize, [2277](#)
  - GetRawCurrent, [2277](#)
  - name, [2277](#)
  - object, [2278](#)
  - operator(), [2278](#)
  - operator[], [2278](#)
  - PermissiveCurrentBool, [2279](#)
  - PrintObject, [2279](#)
  - PrintObjectToBuffer, [2279](#)
  - PrintObjectToFd, [2279](#)
  - PrintObjectToString, [2280](#)
  - ReadFrom, [2280](#)
  - ResetPosition, [2280](#)
  - SetParsePosition, [2280](#)
  - SkipCurrentValue, [2280](#)
  - WriteData, [2281](#)
- ParsedURI, [2281](#)
  - GetAddr, [2282](#)
  - GetHost, [2282](#)
  - GetPath, [2283](#)
  - GetPort, [2283](#)
  - IsSecure, [2283](#)



- NewUri, [2283](#)
- ParsedURI, [2282](#)
- valid, [2283](#)
- ParseFd
  - MQTT::ConfigExposer, [2042](#)
  - MQTT::mqtt\_bool, [2171](#)
  - MQTT::mqtt\_float, [2177](#)
  - MQTT::mqtt\_int, [2183](#)
  - MQTT::mqtt\_IO, [2189](#)
  - MQTT::mqtt\_leaf, [2206](#)
  - MQTT::mqtt\_obj, [2215](#)
  - MQTT::mqtt\_string, [2223](#)
  - MQTT::mqtt\_uint, [2229](#)
  - MQTT::SubHandler, [2358](#)
- PartitionEntry, [2284](#)
- PartitionTable, [2284](#)
- PC Tools and Utilities, [95](#)
- pcsActiveHighOrLow
  - \_lpspi\_master\_config, [1924](#)
  - \_lpspi\_slave\_config, [1931](#)
- pcsToSckDelayInNanoSec
  - \_lpspi\_master\_config, [1924](#)
- peek
  - IOSYS - I/O System, [1289](#)
- Peek\_Begin
  - fifo\_buffer\_storage, [2080](#)
- Peek\_End
  - fifo\_buffer\_storage, [2080](#)
- PeekData
  - fifo\_buffer\_storage, [2080](#)
- PEM Conversion Utilities, [1350](#)
  - CONVERT\_BUFFER\_LENGTH\_MAX, [1351](#)
  - convertLittleEndianDword, [1352](#)
  - convertLittleEndianWord, [1353](#)
  - ConvertPEMFormattedData, [1354](#)
- PEM Data Type Identifiers, [1358](#)
  - CONVERT\_CERTIFICATE, [1359](#)
  - CONVERT\_DSA\_PRIVATE\_KEY, [1359](#)
  - CONVERT\_ECDSA\_PRIVATE\_KEY, [1359](#)
  - CONVERT\_NONE, [1359](#)
  - CONVERT\_RSA\_PRIVATE\_KEY, [1359](#)
- PEM Format Footers, [1360](#)
- PEM Format Headers, [1360](#)
- Performance Tests, [466](#)
- period
  - SSC\_rxtx\_cfg\_t, [2345](#)
- Periodic ADC, [611](#)
- Periodic Interrupt Timer, [669](#)
- Periodic Interrupt Timer (PIT), [734](#)
- PermissiveCurrentBool
  - ParsedJsonDataSet, [2279](#)
- pFragFunc
  - IP, [1321](#)
- phaseSeg1
  - \_flexcan\_timing\_config, [1918](#)
- phaseSeg2
  - \_flexcan\_timing\_config, [1918](#)
- PicKit Serial I2C Demo Board, [509](#)
- Pin Interrupt Request, [723](#)
- PIN\_FN\_0
  - PinIO, [2288–2290](#)
- PIN\_FN\_1
  - PinIO, [2288–2290](#)
- PIN\_FN\_10
  - PinIO, [2288–2290](#)
- PIN\_FN\_11
  - PinIO, [2288, 2289, 2291](#)
- PIN\_FN\_12
  - PinIO, [2288, 2289, 2291](#)
- PIN\_FN\_13
  - PinIO, [2288, 2289, 2291](#)
- PIN\_FN\_14
  - PinIO, [2288, 2289, 2291](#)
- PIN\_FN\_15
  - PinIO, [2288, 2289, 2291](#)
- PIN\_FN\_16
  - PinIO, [2288, 2289, 2291](#)
- PIN\_FN\_17
  - PinIO, [2288, 2289, 2291](#)
- PIN\_FN\_18
  - PinIO, [2288, 2290, 2291](#)
- PIN\_FN\_19
  - PinIO, [2288, 2290, 2291](#)
- PIN\_FN\_2
  - PinIO, [2288–2290](#)
- PIN\_FN\_20
  - PinIO, [2288, 2290, 2291](#)
- PIN\_FN\_21
  - PinIO, [2288, 2290, 2291](#)
- PIN\_FN\_22
  - PinIO, [2288, 2290, 2291](#)
- PIN\_FN\_23
  - PinIO, [2288, 2290, 2291](#)
- PIN\_FN\_24
  - PinIO, [2288, 2290, 2291](#)
- PIN\_FN\_25
  - PinIO, [2289–2291](#)
- PIN\_FN\_26
  - PinIO, [2289–2291](#)
- PIN\_FN\_27
  - PinIO, [2289–2291](#)
- PIN\_FN\_28
  - PinIO, [2289–2291](#)
- PIN\_FN\_29
  - PinIO, [2289–2291](#)
- PIN\_FN\_3
  - PinIO, [2288–2290](#)
- PIN\_FN\_30
  - PinIO, [2289–2291](#)
- PIN\_FN\_31
  - PinIO, [2289–2291](#)
- PIN\_FN\_32
  - PinIO, [2289–2291](#)
- PIN\_FN\_4
  - PinIO, [2288–2290](#)
- PIN\_FN\_5

- PinIO, [2288–2290](#)
- PIN\_FN\_6
  - PinIO, [2288–2290](#)
- PIN\_FN\_7
  - PinIO, [2288–2290](#)
- PIN\_FN\_8
  - PinIO, [2288–2290](#)
- PIN\_FN\_9
  - PinIO, [2288–2290](#)
- PIN\_FN\_A
  - PinIO, [2288–2290](#)
- PIN\_FN\_B
  - PinIO, [2288–2290](#)
- PIN\_FN\_C
  - PinIO, [2288–2290](#)
- PIN\_FN\_D
  - PinIO, [2288–2290](#)
- PIN\_FN\_IN
  - PinIO, [2288–2290](#)
- PIN\_FN\_OUT
  - PinIO, [2288–2290](#)
- pin\_fn\_t
  - PinIO, [2288–2290](#)
- pinCfg
  - \_lpspi\_master\_config, [1924](#)
  - \_lpspi\_slave\_config, [1931](#)
- Ping, [600](#)
  - IP, [1316](#)
  - NB::WebSocket, [2467](#)
- ping
  - IPv6Interface, [2139](#)
  - TwoWire, [2368](#)
- Ping From Web, [1038](#)
- Ping4
  - IP, [1316](#)
- Ping4VialInterface
  - IP, [1317](#)
- Ping6
  - IP, [1318](#)
  - IPv6Interface, [2141](#)
- Ping6VialInterface
  - IP, [1318](#)
- PingVialInterface
  - IP, [1319](#)
- PinIO, [1246](#), [2285](#)
  - function, [2292](#)
  - getFn, [2292](#)
  - mask, [1246](#)
  - multidrv, [2292](#)
  - operator bool, [2292](#)
  - operator!, [2292](#)
  - operator=, [2292](#), [2293](#)
  - PIN\_FN\_0, [2288–2290](#)
  - PIN\_FN\_1, [2288–2290](#)
  - PIN\_FN\_10, [2288–2290](#)
  - PIN\_FN\_11, [2288](#), [2289](#), [2291](#)
  - PIN\_FN\_12, [2288](#), [2289](#), [2291](#)
  - PIN\_FN\_13, [2288](#), [2289](#), [2291](#)
  - PIN\_FN\_14, [2288](#), [2289](#), [2291](#)
  - PIN\_FN\_15, [2288](#), [2289](#), [2291](#)
  - PIN\_FN\_16, [2288](#), [2289](#), [2291](#)
  - PIN\_FN\_17, [2288](#), [2289](#), [2291](#)
  - PIN\_FN\_18, [2288](#), [2290](#), [2291](#)
  - PIN\_FN\_19, [2288](#), [2290](#), [2291](#)
  - PIN\_FN\_2, [2288–2290](#)
  - PIN\_FN\_20, [2288](#), [2290](#), [2291](#)
  - PIN\_FN\_21, [2288](#), [2290](#), [2291](#)
  - PIN\_FN\_22, [2288](#), [2290](#), [2291](#)
  - PIN\_FN\_23, [2288](#), [2290](#), [2291](#)
  - PIN\_FN\_24, [2288](#), [2290](#), [2291](#)
  - PIN\_FN\_25, [2289–2291](#)
  - PIN\_FN\_26, [2289–2291](#)
  - PIN\_FN\_27, [2289–2291](#)
  - PIN\_FN\_28, [2289–2291](#)
  - PIN\_FN\_29, [2289–2291](#)
  - PIN\_FN\_3, [2288–2290](#)
  - PIN\_FN\_30, [2289–2291](#)
  - PIN\_FN\_31, [2289–2291](#)
  - PIN\_FN\_32, [2289–2291](#)
  - PIN\_FN\_4, [2288–2290](#)
  - PIN\_FN\_5, [2288–2290](#)
  - PIN\_FN\_6, [2288–2290](#)
  - PIN\_FN\_7, [2288–2290](#)
  - PIN\_FN\_8, [2288–2290](#)
  - PIN\_FN\_9, [2288–2290](#)
  - PIN\_FN\_A, [2288–2290](#)
  - PIN\_FN\_B, [2288–2290](#)
  - PIN\_FN\_C, [2288–2290](#)
  - PIN\_FN\_D, [2288–2290](#)
  - PIN\_FN\_IN, [2288–2290](#)
  - PIN\_FN\_OUT, [2288–2290](#)
  - pin\_fn\_t, [2288–2290](#)
  - PinIO, [2291](#)
  - pio, [1246](#)
  - pull\_t, [2291](#)
  - PullDown, [2293](#)
  - PullUp, [2293](#)
  - read, [2293](#)
  - readBack, [2293](#)
  - setFn, [2294](#)
  - setHighStrength, [2294](#)
  - setPull, [2294](#)
  - tgl, [2294](#)
  - toggle, [2294](#)
  - PinIOArray2, [2295](#)
  - PinIOArrayJ1, [2295](#)
  - PinIOJ1Array, [2295](#)
  - PinNotify
    - MQTT::mqtt\_IO, [2189](#)
  - PinVector
    - PinVector< n >, [2296](#)
  - PinVector< n >, [2296](#)
    - operator=, [2297](#)
    - PinVector, [2296](#)
  - pio
    - PinIO, [1246](#)

- Pit, [1368](#)
  - \_pit\_chnl, [1369](#)
  - \_pit\_interrupt\_enable, [1369](#)
  - \_pit\_status\_flags, [1369](#)
  - kPIT\_Chnl\_0, [1369](#)
  - kPIT\_Chnl\_1, [1369](#)
  - kPIT\_Chnl\_2, [1369](#)
  - kPIT\_Chnl\_3, [1369](#)
  - kPIT\_TimerFlag, [1369](#)
  - kPIT\_TimerInterruptEnable, [1369](#)
  - pit\_chnl\_t, [1368](#)
  - pit\_config\_t, [1369](#)
  - PIT\_Deinit, [1370](#)
  - PIT\_Init, [1370](#)
- pit\_chnl\_t
  - Pit, [1368](#)
- pit\_config\_t
  - Pit, [1369](#)
- PIT\_Deinit
  - Pit, [1370](#)
- PIT\_Init
  - Pit, [1370](#)
- Platform References, [105](#)
- PLATFORM SPECIFIC, [603](#)
- Platform Specific API, [1370](#)
- Pong
  - NB::WebSocket, [2470](#)
- pool\_buffer, [2297](#)
- POP3 - Post Office Protocol, [1360](#)
  - GetPOPErrorString, [1361](#)
  - POP3\_CloseSession, [1361](#)
  - POP3\_DeleteCmd, [1361](#)
  - POP3\_InitializeSession, [1363](#)
  - POP3\_ListCmd, [1363](#)
  - POP3\_RetrieveMessage, [1363](#)
  - POP3\_StatCmd, [1364](#)
  - POPGetResultCode, [1364](#)
  - SSL\_POP3\_InitializeSession, [1365](#)
- POP3 Return Codes, [1365](#)
- POP3\_CloseSession
  - POP3 - Post Office Protocol, [1361](#)
- POP3\_DeleteCmd
  - POP3 - Post Office Protocol, [1361](#)
- POP3\_InitializeSession
  - POP3 - Post Office Protocol, [1363](#)
- POP3\_ListCmd
  - POP3 - Post Office Protocol, [1363](#)
- POP3\_RetrieveMessage
  - POP3 - Post Office Protocol, [1363](#)
- POP3\_StatCmd
  - POP3 - Post Office Protocol, [1364](#)
- POPGetResultCode
  - POP3 - Post Office Protocol, [1364](#)
- PopulateAuthHeader
  - Web Client, [1862](#)
- Posix
  - TimeZoneRecord, [2364](#)
- Post JSON to Server, [552](#)
- Post JSON to Server and Display Result, [550](#)
- PostConfigHardwareInit
  - HAL - Hardware Abstraction Layer, [1255](#)
- PPP, [776](#)
- PPP - Point to Point Protocol, [1366](#)
- PPP Connection State, [1366](#)
  - eAnswering, [1367](#)
  - eCHAPAuthenticate, [1367](#)
  - eClosed, [1367](#)
  - eClosing, [1367](#)
  - eDialing, [1367](#)
  - eInitializingModem, [1367](#)
  - eLCPNegotiate, [1367](#)
  - eNCPNegotiate, [1367](#)
  - enum\_PPPState, [1367](#)
  - eOpen, [1367](#)
  - ePAPAuthenticate, [1367](#)
  - eWait4Ring, [1367](#)
  - eWaitForTrain, [1367](#)
- PPP Return Codes, [1367](#)
- pPromiscuousPacketFunc
  - IP, [1321](#)
- PPS\_1588\_TimeCapIsr
  - IEEE1588Timer, [2097](#)
- preDivider
  - \_flexcan\_timing\_config, [1918](#)
- Prefered
  - IPV6\_PREFIX, [2124](#)
- primarySource
  - \_qtmr\_config, [1940](#)
- print
  - IPADDR6, [2110](#)
  - Utilities, [1793](#)
- PrintNError
  - SMTP - Send Email, [1435](#)
- PrintObject
  - ParsedJsonDataSet, [2279](#)
- PrintObjectToBuffer
  - ParsedJsonDataSet, [2279](#)
- PrintObjectToFd
  - ParsedJsonDataSet, [2279](#)
- PrintObjectToString
  - ParsedJsonDataSet, [2280](#)
- PrintServerLog
  - SMTP - Send Email, [1436](#)
- priority
  - \_flexcan\_rx\_fifo\_config, [1917](#)
- ProcessAccept
  - SerialRecord, [2317](#)
- ProcessAdvert
  - IPV6\_NEIGHBOR, [2120](#)
- ProcessATcommandFunc
  - AT Commands, [1119](#)
- ProcessListenError
  - SerialRecord, [2317](#)
- ProcessNetworkError
  - SerialRecord, [2317](#)
- ProcessRaw

- CallbackFunctionPageHandler, [1953](#)
  - CallbackFunctionPostHandler, [1954](#)
  - HtmlPageHandler, [2084](#)
- ProcessReadNetworkData
  - SerialRecord, [2317](#)
- ProcessSerialError
  - SerialRecord, [2317](#)
- ProcessSpecialFrameTCPReadSerialData
  - SerialRecord, [2317](#)
- ProcessSpecialFrameWriteNetworkData
  - SerialRecord, [2317](#)
- ProcessSpecialFrameWriteTimeout
  - SerialRecord, [2317](#)
- ProcessTCPReadSerialData
  - SerialRecord, [2317](#)
- ProcessTimeouts
  - SerialRecord, [2318](#)
- ProcessWriteNetworkData
  - SerialRecord, [2318](#)
- ProcessWriteSerialData
  - SerialRecord, [2318](#)
- Production & Deployment, [127](#)
- Profiler, [781](#)
- ProgramApplication
  - Signed Application Update, [1523](#)
- Programmable Clock Output, [723](#)
- Programmers Guide, [133](#)
- PromisciousPacketFunc
  - IP, [1303](#)
- promote\_tcp\_ws
  - NB::WebSocket, [2473](#)
- propSeg
  - \_flexcan\_timing\_config, [1918](#)
- prot\_size
  - ARP - Address Resolution Protocol, [1117](#)
- prot\_Type
  - ARP - Address Resolution Protocol, [1117](#)
- protocol\_exception\_handling
  - mcanMODM7AE70::mcan\_config, [2159](#)
- PsudeoHeader, [2298](#)
- pTaskPacketFunc
  - IP, [1321](#)
- Publish
  - MQTT::Client, [1967](#), [1968](#)
  - MQTT::mqtt\_leaf, [2206](#)
- PubState
  - MQTT::mqtt\_leaf, [2207](#)
- pull\_t
  - PinIO, [2291](#)
- PullDown
  - PinIO, [2293](#)
- PullUp
  - PinIO, [2293](#)
- Pulse Generator and Counter, [669](#)
- PUSHR\_CONT\_BIT
  - MCF5441x (DSPI), [1452](#)
- PUSHR\_EOQ\_BIT
  - MCF5441x (DSPI), [1452](#)
- putbytenum
  - Utilities, [1794](#)
- putnum
  - Utilities, [1794](#)
- PWM, [670](#), [724](#), [726](#), [738](#)
- QSPI state, [1461](#)
- QSPI\_Record, [2298](#)
- QSPIdone
  - MCF5441x (DSPI), [1455](#)
  - MCF5441x (Queued SPI 2.x compatibility layer), [1459](#)
  - SAME70 (DSPI), [1471](#)
- QSPInit
  - MCF5441x (DSPI), [1456](#)
  - MCF5441x (Queued SPI 2.x compatibility layer), [1459](#)
  - SAME70 (DSPI), [1471](#)
- QSPISart
  - MCF5441x (DSPI), [1457](#)
  - MCF5441x (Queued SPI 2.x compatibility layer), [1461](#)
  - SAME70 (DSPI), [1473](#)
- QTMR (Quad Timer), [1404](#)
  - \_qtmr\_channel\_selection, [1407](#)
  - \_qtmr\_counting\_mode, [1407](#)
  - \_qtmr\_debug\_action, [1408](#)
  - \_qtmr\_dma\_enable, [1408](#)
  - \_qtmr\_input\_capture\_edge, [1408](#)
  - \_qtmr\_input\_source, [1408](#)
  - \_qtmr\_interrupt\_enable, [1409](#)
  - \_qtmr\_output\_mode, [1409](#)
  - \_qtmr\_preload\_control, [1409](#)
  - \_qtmr\_primary\_count\_source, [1410](#)
  - \_qtmr\_pwm\_out\_state, [1410](#)
  - \_qtmr\_status\_flags, [1410](#)
  - FSL\_QTMR\_DRIVER\_VERSION, [1407](#)
  - kQTMR\_AssertWhenCountActive, [1409](#)
  - kQTMR\_CascadeCount, [1408](#)
  - kQTMR\_Channel\_0, [1407](#)
  - kQTMR\_Channel\_1, [1407](#)
  - kQTMR\_Channel\_2, [1407](#)
  - kQTMR\_Channel\_3, [1407](#)
  - kQTMR\_ClearOnCompare, [1409](#)
  - kQTMR\_ClockCounter0InputPin, [1410](#)
  - kQTMR\_ClockCounter0Output, [1410](#)
  - kQTMR\_ClockCounter1InputPin, [1410](#)
  - kQTMR\_ClockCounter1Output, [1410](#)
  - kQTMR\_ClockCounter2InputPin, [1410](#)
  - kQTMR\_ClockCounter2Output, [1410](#)
  - kQTMR\_ClockCounter3InputPin, [1410](#)
  - kQTMR\_ClockCounter3Output, [1410](#)
  - kQTMR\_ClockDivide\_1, [1410](#)
  - kQTMR\_ClockDivide\_128, [1410](#)
  - kQTMR\_ClockDivide\_16, [1410](#)
  - kQTMR\_ClockDivide\_2, [1410](#)
  - kQTMR\_ClockDivide\_32, [1410](#)
  - kQTMR\_ClockDivide\_4, [1410](#)
  - kQTMR\_ClockDivide\_64, [1410](#)

- kQTMR\_ClockDivide\_8, [1410](#)
- kQTMR\_ComparatorPreload1DmaEnable, [1408](#)
- kQTMR\_ComparatorPreload2DmaEnable, [1408](#)
- kQTMR\_Compare1Flag, [1411](#)
- kQTMR\_Compare1InterruptEnable, [1409](#)
- kQTMR\_Compare2Flag, [1411](#)
- kQTMR\_Compare2InterruptEnable, [1409](#)
- kQTMR\_CompareFlag, [1410](#)
- kQTMR\_CompareInterruptEnable, [1409](#)
- kQTMR\_Counter0InputPin, [1409](#)
- kQTMR\_Counter1InputPin, [1409](#)
- kQTMR\_Counter2InputPin, [1409](#)
- kQTMR\_Counter3InputPin, [1409](#)
- kQTMR\_EdgeFlag, [1411](#)
- kQTMR\_EdgeInterruptEnable, [1409](#)
- kQTMR\_EnableGateClock, [1409](#)
- kQTMR\_FallingEdge, [1408](#)
- kQTMR\_ForceOutToZero, [1408](#)
- kQTMR\_HaltCounter, [1408](#)
- kQTMR\_HaltCountForceOutZero, [1408](#)
- kQTMR\_InputEdgeFlagDmaEnable, [1408](#)
- kQTMR\_LoadOnComp1, [1410](#)
- kQTMR\_LoadOnComp2, [1410](#)
- kQTMR\_NoCapture, [1408](#)
- kQTMR\_NoOperation, [1408](#)
- kQTMR\_NoPreload, [1409](#)
- kQTMR\_OverflowFlag, [1411](#)
- kQTMR\_OverflowInterruptEnable, [1409](#)
- kQTMR\_PriSrcRiseAndFallEdge, [1408](#)
- kQTMR\_PriSrcRiseEdge, [1408](#)
- kQTMR\_PriSrcRiseEdgeSecDir, [1408](#)
- kQTMR\_PriSrcRiseEdgeSecInpHigh, [1408](#)
- kQTMR\_PwmHigh, [1410](#)
- kQTMR\_PwmLow, [1410](#)
- kQTMR\_QuadCountMode, [1408](#)
- kQTMR\_RisingAndFallingEdge, [1408](#)
- kQTMR\_RisingEdge, [1408](#)
- kQTMR\_RunNormalInDebug, [1408](#)
- kQTMR\_SecSrcTrigPriCnt, [1408](#)
- kQTMR\_SetOnCompare, [1409](#)
- kQTMR\_SetOnCompareClearOnCountRoll, [1409](#)
- kQTMR\_SetOnCompareClearOnSecSrcInp, [1409](#)
- kQTMR\_ToggleOnAltCompareReg, [1409](#)
- kQTMR\_ToggleOnCompare, [1409](#)
- QTMR\_ClearStatusFlags, [1411](#)
- qtmr\_config\_t, [1407](#)
- QTMR\_Deinit, [1411](#)
- QTMR\_DisableDma, [1411](#)
- QTMR\_DisableInterrupts, [1412](#)
- QTMR\_EnableDma, [1412](#)
- QTMR\_EnableInterrupts, [1412](#)
- QTMR\_GetDefaultConfig, [1413](#)
- QTMR\_GetEnabledInterrupts, [1413](#)
- QTMR\_GetPwmChannelStatus, [1413](#)
- QTMR\_GetStatus, [1414](#)
- QTMR\_Init, [1414](#)
- QTMR\_SetCompareValue, [1414](#)
- QTMR\_SetPwmClockMode, [1415](#)
- QTMR\_SetPwmOutputToldle, [1415](#)
- QTMR\_SetTimerPeriod, [1416](#)
- QTMR\_SetupInputCapture, [1416](#)
- QTMR\_SetupPwm, [1417](#)
- QTMR PWM, [739](#)
- QTMR\_ClearStatusFlags
  - QTMR (Quad Timer), [1411](#)
- qtmr\_config\_t
  - QTMR (Quad Timer), [1407](#)
- QTMR\_Deinit
  - QTMR (Quad Timer), [1411](#)
- QTMR\_DisableDma
  - QTMR (Quad Timer), [1411](#)
- QTMR\_DisableInterrupts
  - QTMR (Quad Timer), [1412](#)
- QTMR\_EnableDma
  - QTMR (Quad Timer), [1412](#)
- QTMR\_EnableInterrupts
  - QTMR (Quad Timer), [1412](#)
- QTMR\_GetDefaultConfig
  - QTMR (Quad Timer), [1413](#)
- QTMR\_GetEnabledInterrupts
  - QTMR (Quad Timer), [1413](#)
- QTMR\_GetPwmChannelStatus
  - QTMR (Quad Timer), [1413](#)
- QTMR\_GetStatus
  - QTMR (Quad Timer), [1414](#)
- QTMR\_Init
  - QTMR (Quad Timer), [1414](#)
- QTMR\_SetCompareValue
  - QTMR (Quad Timer), [1414](#)
- QTMR\_SetPwmClockMode
  - QTMR (Quad Timer), [1415](#)
- QTMR\_SetPwmOutputToldle
  - QTMR (Quad Timer), [1415](#)
- QTMR\_SetTimerPeriod
  - QTMR (Quad Timer), [1416](#)
- QTMR\_SetupInputCapture
  - QTMR (Quad Timer), [1416](#)
- QTMR\_SetupPwm
  - QTMR (Quad Timer), [1417](#)
- QueueAllSubscriptions
  - MQTT::Client, [1969](#)
- QueuePublish
  - MQTT::Client, [1969](#)
  - MQTT::mqtt\_leaf, [2207](#)
- QueueSubscribe
  - MQTT::Client, [1970](#)
  - MQTT::mqtt\_leaf, [2207](#)
- QueueUnsubscribe
  - MQTT::Client, [1970](#), [1971](#)
  - MQTT::mqtt\_leaf, [2207](#)
- Quick Reference, [758](#)
- RAM Drive, [467](#)
- Raw Form Post 2pX, [1041](#)
- rd\_cycle
  - SEMC\_cfg\_t, [2314](#)
- rd\_en\_hi

- SEMC\_cfg\_t, [2314](#)
- rd\_en\_lo
  - SEMC\_cfg\_t, [2314](#)
- rd\_hold
  - SEMC\_cfg\_t, [2314](#)
- rdMode
  - EBI - External Bus Interface, [1403](#)
- read
  - IOSYS - I/O System, [1289](#)
  - PinIO, [2293](#)
  - TwoWire, [2368](#)
  - WireIntf, [2526](#)
- readall
  - IOSYS - I/O System, [1290](#)
- ReadAllWithTickTimeout
  - IOSYS - I/O System, [1290](#)
- ReadAllWithTimeout
  - IOSYS - I/O System, [1291](#)
- readBack
  - PinIO, [2293](#)
- ReadBinaryApplicationCodeFromStream
  - Stream Update, [1525](#)
- ReadData
  - fifo\_buffer\_storage, [2080](#)
  - NB::WebSocket, [2475](#)
- ReadFrom
  - ParsedJsonDataSet, [2280](#)
- ReadReg
  - WM8904, [2530](#)
- readReg8
  - BBI2C, [1945](#)
  - I2C, [2089](#)
  - I2CDevice, [2092](#)
  - TwoWire, [2369](#)
- readRegN
  - BBI2C, [1945](#)
  - I2C, [2089](#)
  - I2CDevice, [2092](#)
  - TwoWire, [2369](#)
- readRegRemainingTimes
  - \_lpspi\_master\_edma\_handle, [1926](#)
  - \_lpspi\_master\_handle, [1929](#)
  - \_lpspi\_slave\_edma\_handle, [1932](#)
  - \_lpspi\_slave\_handle, [1934](#)
- ReadS19ApplicationCodeFromStream
  - Stream Update, [1525](#)
- ReadSimpleBody
  - HTTP\_Request, [2086](#)
- ReadWithTickTimeout
  - IOSYS - I/O System, [1291](#)
- ReadWithTimeout
  - IOSYS - I/O System, [1292](#)
- ReadyReceiveBuffer
  - SAICtx\_t, [2307](#)
  - SSCCTX\_t, [2347](#)
  - WM8904, [2530](#)
- RebindDHCP
  - DhcpObject, [2056](#)
- Receive
  - UDPPacket, [2400](#)
- ReceiveUDP, [1903](#)
  - send\_test\_packet, [1904](#)
  - signal\_handler, [1904](#)
- Recovery: MOD5441x, NANO54415, SB800EX, [118](#)
- Recovery: MODM7AE70, SBE70LC, [115](#)
- Recovery: SOMRT1061, [122](#)
- recvfrom
  - UDP Socket API, [1773](#)
- RedirectResponse
  - HTTP and HTML Functions, [1265](#)
- RegisterAppSigningPublicKey
  - Signed Application Update, [1523](#)
- RegisterEphemeralFifo
  - UDP, [1748](#)
- RegisterFDCallBack
  - IOSYS - I/O System, [1293](#)
- RegisterInterface
  - InterfaceBlock, [2102](#)
- RegisterMqttObj
  - MQTT::Client, [1971](#)
- RegisterMulticastFifo4
  - Multicast, [1333](#)
- RegisterMulticastFifo6
  - Multicast, [1334](#)
- RegisterRxBufferCB
  - SAICtx\_t, [2308](#)
- RegisterRxFifo
  - mcanMODM7AE70::mcan\_module, [2164](#)
- RegisterRxFifoMask
  - mcanMODM7AE70::mcan\_module, [2164](#)
- RegisterRxFifoRange
  - mcanMODM7AE70::mcan\_module, [2165](#)
- RegisterSem
  - DSPIModule, [2069](#)
  - SPIModule, [2338](#)
- RegisterTCPReadNotify
  - TCP Notification System, [1709](#)
- RegisterTCPWriteNotify
  - TCP Notification System, [1711](#)
- RegisterTopicHandler
  - MQTT::Client, [1971](#)
- RegisterTxBufferCB
  - SAICtx\_t, [2309](#)
- RegisterUDPFifo
  - UDP, [1750](#)
- RegisterUDPFifoVia
  - UDP, [1751](#)
- RegisterUDPFifoWithNotify
  - UDP, [1754](#)
- RegisterUDPFifoWithNotifyVia
  - UDP, [1755](#)
- Release 2.x to 3.x Overview, [41](#)
- Release 2.x to 3.x Porting Guide, [44](#)
- RemainingValidTime
  - IPV6\_PREFIX, [2124](#)
- Remote Web Console for Serial/stdout, [1043](#)

- remote\_frames\_extended\_reject
  - mcanMODM7AE70::mcan\_config, 2159
- remote\_frames\_standard\_reject
  - mcanMODM7AE70::mcan\_config, 2159
- Remove
  - buffer\_list, 1950
  - IPV6\_ROOT\_EL\_CONTAINER, 2127
- RemoveDefaultGateway
  - IPv6Interface, 2139
- RemoveFirstChild
  - MQTT::mqtt\_leaf, 2207
  - MQTT::mqtt\_obj, 2215
- RemoveHead
  - buffer\_list, 1950
- Removeinterface
  - Network Interfaces, 1343
- RemoveObj
  - MQTT::Client, 1972
- RemoveStaticAddress
  - IPv6Interface, 2140
- RemoveStaticDNS
  - IPv6Interface, 2140
- RemoveStaticIPv6Address
  - DHCPv6 Client, 1179
- RemoveTail
  - buffer\_list, 1950
- RemoveTopicHandler
  - MQTT::Client, 1972
- RemoveUserAuth
  - UserAuthManager, 2424
- RenderToBuffer
  - MQTT::ConfigExposer, 2042
  - MQTT::mqtt\_bool, 2171
  - MQTT::mqtt\_float, 2177
  - MQTT::mqtt\_int, 2184
  - MQTT::mqtt\_IO, 2190
  - MQTT::mqtt\_leaf, 2208
  - MQTT::mqtt\_obj, 2215
  - MQTT::mqtt\_string, 2223
  - MQTT::mqtt\_uint, 2230
  - MQTT::SubHandler, 2358
- RenderToFd
  - MQTT::ConfigExposer, 2043
  - MQTT::mqtt\_bool, 2171
  - MQTT::mqtt\_float, 2177
  - MQTT::mqtt\_int, 2184
  - MQTT::mqtt\_IO, 2190
  - MQTT::mqtt\_leaf, 2208
  - MQTT::mqtt\_obj, 2216
  - MQTT::mqtt\_string, 2223
  - MQTT::mqtt\_uint, 2230
  - MQTT::SubHandler, 2358
- RenewDHCP
  - DhcpObject, 2056
- replace
  - NBString, 2243, 2244
- ReplaceStdio
  - IOSYS - I/O System, 1293
- ReportGetTextValue
  - I6Record, 2095
- requestFrom
  - TwoWire, 2369
  - WireIntf, 2526
- Reserve
  - NBString, 2244
- RESERVED0
  - SEMC\_cfg\_t, 2314
- RESERVED1
  - SEMC\_cfg\_t, 2314
- reserved3
  - \_FlexSPIConfig, 1921
- Reset
  - fifo\_buffer\_storage, 2080
- resetConsecutiveSeconds
  - IEEE1588Timer, 2097
- ResetData
  - UDPPacket, 2402
- ResetPosition
  - ParsedJsonDataSet, 2280
- RestartDHCP
  - DhcpObject, 2056
- Result\_t
  - BBI2C, 1944
  - I2C, 2088
- ring\_diff\_forward
  - IEEE1588Timer, 2097
- rJumpwidth
  - \_flexcan\_timing\_config, 1918
- RootParse
  - MQTT::mqtt\_leaf, 2208
- Round Robin Task Sheduler, 795
- RouterHeaderParseHasError
  - IPv6FrameProcessingStruct, 2131
- RSER\_DMA\_IRQ\_ONLY
  - MCF5441x (DSPI), 1452
- RSER\_EOQF\_IRQ\_ONLY
  - MCF5441x (DSPI), 1452
- RTC - External, 655
- RTC - On Chip, 669
- RTOS, 783
- run\_in\_standby
  - mcanMODM7AE70::mcan\_config, 2159
- RunSkippedCallBack
  - NB::WebSocket, 2479
- RV3032 I2C Real-Time Clock, 512
- Rx
  - DSPIModule, 2069
  - SPI\_QSPI, 2323
  - SPI\_USART, 2328
  - SPIModule, 2339
- rx
  - SAI\_cfg\_t, 2300
  - SSC\_cfg\_t, 2343
- rx\_fifo\_0\_overwrite
  - mcanMODM7AE70::mcan\_config, 2159
- rx\_fifo\_0\_watermark



- mcanMODM7AE70::mcan\_config, 2159
- rx\_fifo\_1\_overwrite
  - mcanMODM7AE70::mcan\_config, 2159
- rx\_fifo\_1\_watermark
  - mcanMODM7AE70::mcan\_config, 2159
- rxBuffIfNull
  - \_lpspi\_master\_edma\_handle, 1926
  - \_lpspi\_slave\_edma\_handle, 1933
- rxClkFreq
  - SAI\_cfg\_t, 2300
- rxData
  - \_lpspi\_master\_edma\_handle, 1926
  - \_lpspi\_master\_handle, 1929
  - \_lpspi\_slave\_edma\_handle, 1933
  - \_lpspi\_slave\_handle, 1935
  - \_lpspi\_transfer, 1936
- rxFifoFrameBuf
  - \_flexcan\_handle, 1915
- rxFifoFrameNum
  - \_flexcan\_handle, 1915
- rxFifoState
  - \_flexcan\_handle, 1916
- rxFifoTransferTotalNum
  - \_flexcan\_handle, 1916
- rxRemainingByteCount
  - \_lpspi\_master\_edma\_handle, 1926
  - \_lpspi\_master\_handle, 1929
  - \_lpspi\_slave\_edma\_handle, 1933
  - \_lpspi\_slave\_handle, 1935
- RXTX\_ASYNC
  - SAI Configuration Enumerations, 1532
- RXTX\_SYNC\_RX
  - SAI Configuration Enumerations, 1532
- RXTX\_SYNC\_TX
  - SAI Configuration Enumerations, 1532
- rxTxSync
  - SAI\_cfg\_t, 2300
- rxTxSync\_t
  - SAI Configuration Enumerations, 1531
- rxWatermark
  - \_lpspi\_master\_edma\_handle, 1926
  - \_lpspi\_master\_handle, 1929
  - \_lpspi\_slave\_edma\_handle, 1933
  - \_lpspi\_slave\_handle, 1935
- s\_openSocketCount
  - NB::WebSocket, 2517
- SADDR
  - \_edma\_tcd, 1909
- sai
  - SAI\_cfg\_t, 2300
  - Synchronous Audio Interface (SOMRT1061), 1527
- SAI Callback Types, 1527
  - SAI\_BufferDoneFn\_t, 1527
- SAI Configuration Enumerations, 1528
  - bitOrder\_t, 1529
  - BUFFER\_WORD16\_LEFT, 1531
  - BUFFER\_WORD16\_RIGHT, 1531
  - BUFFER\_WORD32\_LEFT, 1531
  - BUFFER\_WORD32\_RIGHT, 1531
  - BUFFER\_WORD8, 1530
  - bufferDepletionBehavior\_t, 1529
  - CLK\_ACTIVE\_HIGH, 1530
  - CLK\_ACTIVE\_LOW, 1530
  - CLK\_CH\_CH, 1529
  - CLK\_CH\_RK, 1529
  - CLK\_CH\_TK, 1529
  - CLK\_GATE\_CONTINUOUS, 1529
  - CLK\_GATE\_FRAME\_HIGH, 1529
  - CLK\_GATE\_FRAME\_LOW, 1529
  - CLK\_OUT\_CONTINUOUS, 1530
  - CLK\_OUT\_INPUT, 1530
  - CLK\_OUT\_TRANSFER, 1530
  - CLK\_SRC\_EXT, 1530
  - CLK\_SRC\_MCK, 1530
  - clkCh\_t, 1529
  - clkGate\_t, 1529
  - clkOut\_t, 1529
  - clkPol\_t, 1530
  - clkSrc\_t, 1530
  - dataPacking\_t, 1530
  - DEPLETED\_PAUSE, 1529
  - DEPLETED\_REPEAT\_LAST, 1529
  - FRAME\_SYNC\_FALLING, 1531
  - FRAME\_SYNC\_INPUT, 1531
  - FRAME\_SYNC\_NEGATIVE, 1531
  - FRAME\_SYNC\_POSITIVE, 1531
  - FRAME\_SYNC\_RISING, 1531
  - frameEdge\_t, 1531
  - frameSyncCh\_t, 1531
  - frameSyncOut\_t, 1531
  - LEAST\_SIG\_FIRST, 1529
  - MOST\_SIG\_FIRST, 1529
  - RXTX\_ASYNC, 1532
  - RXTX\_SYNC\_RX, 1532
  - RXTX\_SYNC\_TX, 1532
  - rxTxSync\_t, 1531
  - START\_CMP\_0, 1532
  - START\_CONTINUOUS, 1532
  - START\_FRAME\_EDGE, 1532
  - START\_FRAME\_FALLING, 1532
  - START\_FRAME\_HIGH, 1532
  - START\_FRAME\_LEVEL, 1532
  - START\_FRAME\_LOW, 1532
  - START\_FRAME\_RISING, 1532
  - START\_SYNC\_RX\_TX, 1532
  - startCond\_t, 1532
  - SYNC\_CH\_CH, 1531
  - SYNC\_CH\_RX, 1531
  - SYNC\_CH\_TX, 1531
  - SAI Configuration Structures, 1532
  - SAI Driver Class, 1532
  - SAI Driver Constants, 1532
    - SAI\_HW\_COUNT, 1533
    - SAI\_RXBD\_COUNT, 1533
    - SAI\_TXBD\_COUNT, 1533
  - SAI Predefined Configurations, 1533



- SAI\_I2S\_SLAVE\_16\_TXRX\_RK, [1534](#)
- SAI\_I2S\_SLAVE\_16\_TXRX\_TK, [1534](#)
- SAI\_I2S\_SLAVE\_16\_TXRX\_TKRK, [1534](#)
- SAI\_I2S\_SLAVE\_24\_TXRX\_RK, [1534](#)
- SAI\_I2S\_SLAVE\_24\_TXRX\_TK, [1534](#)
- SAI\_I2S\_SLAVE\_24\_TXRX\_TKRK, [1534](#)
- SAI\_I2S\_SLAVE\_8\_TXRX\_RK, [1534](#)
- SAI\_I2S\_SLAVE\_8\_TXRX\_TK, [1534](#)
- SAI\_I2S\_SLAVE\_8\_TXRX\_TKRK, [1534](#)
- SAI\_LJUST\_SLAVE\_16\_TXRX\_RK, [1534](#)
- SAI\_LJUST\_SLAVE\_16\_TXRX\_TK, [1535](#)
- SAI\_LJUST\_SLAVE\_16\_TXRX\_TKRK, [1535](#)
- SAI\_LJUST\_SLAVE\_24\_TXRX\_RK, [1535](#)
- SAI\_LJUST\_SLAVE\_24\_TXRX\_TK, [1535](#)
- SAI\_LJUST\_SLAVE\_24\_TXRX\_TKRK, [1535](#)
- SAI\_LJUST\_SLAVE\_8\_TXRX\_RK, [1535](#)
- SAI\_LJUST\_SLAVE\_8\_TXRX\_TK, [1535](#)
- SAI\_LJUST\_SLAVE\_8\_TXRX\_TKRK, [1535](#)
- SAI\_BufferDoneFn\_t
  - SAI Callback Types, [1527](#)
- SAI\_cfg\_t, [2299](#)
  - rx, [2300](#)
  - rxClkFreq, [2300](#)
  - rxTxSync, [2300](#)
  - sai, [2300](#)
  - tx, [2300](#)
  - txClkFreq, [2300](#)
- SAI\_HW\_COUNT
  - SAI Driver Constants, [1533](#)
- SAI\_I2S\_SLAVE\_16\_TXRX\_RK
  - SAI Predefined Configurations, [1534](#)
- SAI\_I2S\_SLAVE\_16\_TXRX\_TK
  - SAI Predefined Configurations, [1534](#)
- SAI\_I2S\_SLAVE\_16\_TXRX\_TKRK
  - SAI Predefined Configurations, [1534](#)
- SAI\_I2S\_SLAVE\_24\_TXRX\_RK
  - SAI Predefined Configurations, [1534](#)
- SAI\_I2S\_SLAVE\_24\_TXRX\_TK
  - SAI Predefined Configurations, [1534](#)
- SAI\_I2S\_SLAVE\_24\_TXRX\_TKRK
  - SAI Predefined Configurations, [1534](#)
- SAI\_I2S\_SLAVE\_8\_TXRX\_RK
  - SAI Predefined Configurations, [1534](#)
- SAI\_I2S\_SLAVE\_8\_TXRX\_TK
  - SAI Predefined Configurations, [1534](#)
- SAI\_I2S\_SLAVE\_8\_TXRX\_TKRK
  - SAI Predefined Configurations, [1534](#)
- SAI\_LJUST\_SLAVE\_16\_TXRX\_RK
  - SAI Predefined Configurations, [1534](#)
- SAI\_LJUST\_SLAVE\_16\_TXRX\_TK
  - SAI Predefined Configurations, [1535](#)
- SAI\_LJUST\_SLAVE\_16\_TXRX\_TKRK
  - SAI Predefined Configurations, [1535](#)
- SAI\_LJUST\_SLAVE\_24\_TXRX\_RK
  - SAI Predefined Configurations, [1535](#)
- SAI\_LJUST\_SLAVE\_24\_TXRX\_TK
  - SAI Predefined Configurations, [1535](#)
- SAI\_LJUST\_SLAVE\_24\_TXRX\_TKRK
  - SAI Predefined Configurations, [1535](#)
- SAI\_SAI\_Predefined\_Configurations, [1535](#)
- SAI\_RXBD\_COUNT
  - SAI Driver Constants, [1533](#)
- SAI\_rxtx\_cfg\_t, [2300](#)
  - bitOrder, [2301](#)
  - bitsPerWord, [2301](#)
  - clkCh, [2301](#)
  - clkGate, [2302](#)
  - clkOut, [2302](#)
  - clkPol, [2302](#)
  - clkSrc, [2302](#)
  - dataPacking, [2302](#)
  - depletionBehavior, [2302](#)
  - enable, [2302](#)
  - linIdleState, [2302](#)
  - startCond, [2302](#)
  - startDly, [2302](#)
  - syncCh, [2303](#)
  - syncEdge, [2303](#)
  - syncLen, [2303](#)
  - syncOut, [2303](#)
  - unifyDataStream, [2303](#)
  - wordMask, [2303](#)
  - wordsPerFrame, [2303](#)
- SAI\_TXBD\_COUNT
  - SAI Driver Constants, [1533](#)
- SAICtx\_t, [2303](#)
  - clrStickyRxError, [2305](#)
  - clrStickyTxError, [2305](#)
  - CTX\_STATE\_INIT, [2305](#)
  - CTX\_STATE\_RX, [2305](#)
  - CTX\_STATE\_STOPPED, [2305](#)
  - CTX\_STATE\_TX, [2305](#)
  - CTX\_STATE\_TXRX, [2305](#)
  - ctxState\_t, [2305](#)
  - dump, [2306](#)
  - getCurrentConfig, [2306](#)
  - GetMaxRxBuffers, [2306](#)
  - GetMaxTxBuffers, [2306](#)
  - getState, [2306](#)
  - getStickyRxError, [2307](#)
  - getStickyTxError, [2307](#)
  - Init, [2307](#)
  - ReadyReceiveBuffer, [2307](#)
  - RegisterRxBufferCB, [2308](#)
  - RegisterTxBufferCB, [2309](#)
  - Shutdown, [2309](#)
  - TransmitBuffer, [2309](#)
- SAME70 (DSPI), [1467](#)
  - CHIP\_SELECT\_0, [1473](#)
  - CHIP\_SELECT\_DISABLED, [1473](#)
  - CS\_ASSERT\_HIGH, [1469](#)

- CS\_ASSERT\_LOW, 1469
- csReturnType, 1469
- DEASSERT\_AFTER\_LAST, 1469
- DEASSERT\_EVERY\_TRANSFER, 1469
- DEASSERT\_NEVER, 1469
- DSPIDone, 1470
- DSPIInit, 1470
- DSPIStart, 1471
- QSPIDone, 1471
- QSPIInit, 1471
- QSPIStart, 1473
- spiChipSelectPolarity, 1469
- SAME70 (QuadSPI), 1474
- SAME70 (USART), 1474
- Save Config Record Errors, 1890
- SaveAuthRecordsFn
  - User Authorization Manager, 1777
- SaveConfigToStorage
  - Configuration Server, 1171
- SaveUserParameters
  - System Functions, 1541
- SB800EX Diagnostic Monitor, 704
- SB800EX Only, 703
- SB800EX Platform Reference, 110
- SBE70LC, 725
- SBE70LC Platform Reference, 112
- Scan Request Errors, 1890
- ScanAndShowNetworks
  - Wifi, 1886
- ScanErrors
  - NB::Error::Scan, 1903
- ScanForNetworks
  - Wifi, 1886
- SDHC Flash Card, 640
- Sec\_NotFound
  - NB::Error::Connect, 1902
- secondarySource
  - \_qtmr\_config, 1940
- SecureSerToEthFactoryApp, 834
- Security Options, 1891
- select
  - IOSYS - I/O System, 1294
- SEMC, 1417
  - AdvMux, 1424
  - Async, 1426
  - Burst\_1, 1425
  - Burst\_16, 1425
  - Burst\_2, 1425
  - Burst\_32, 1425
  - Burst\_4, 1425
  - Burst\_64, 1425
  - Burst\_8, 1425
  - Bus, 1424
  - ConfigureSEMC, 1427, 1429
  - Mux, 1424
  - SEMC\_AddrMux\_t, 1424
  - SEMC\_Burst\_t, 1424
  - SEMC\_BusWidth\_t, 1425
  - SEMC\_ClkMode\_t, 1426
  - SEMC\_ColAddrWidth\_t, 1426
  - Sync, 1426
  - Width\_0, 1427
  - Width\_10, 1427
  - Width\_11, 1427
  - Width\_12, 1427
  - Width\_16, 1426
  - Width\_2, 1427
  - Width\_3, 1427
  - Width\_4, 1427
  - Width\_5, 1427
  - Width\_6, 1427
  - Width\_7, 1427
  - Width\_8, 1426, 1427
  - Width\_9, 1427
- SEMC Simple, 739
- SEMC to SRAM, FPGA, LCD, 742
- SEMC\_AddrMux\_t
  - SEMC, 1424
- SEMC\_Burst\_t
  - SEMC, 1424
- SEMC\_BusWidth\_t
  - SEMC, 1425
- SEMC\_cfg\_t, 2310
  - addr\_hold, 2312
  - addr\_setup, 2313
  - addr\_wr\_hold, 2313
  - advHoldLvl, 2313
  - advPol, 2313
  - burstLen, 2313
  - busWidth, 2313
  - clkMode, 2313
  - clkPin, 2313
  - colAddrWidth, 2313
  - CR0, 2313
  - CR1, 2313
  - CR2, 2313
  - cs\_hold\_min, 2314
  - cs\_min\_intv, 2314
  - cs\_setup, 2314
  - csPin, 2314
  - latency, 2314
  - muxMode, 2314
  - rd\_cycle, 2314
  - rd\_en\_hi, 2314
  - rd\_en\_lo, 2314
  - rd\_hold, 2314
  - RESERVED0, 2314
  - RESERVED1, 2314
  - turnaround, 2315
  - wr\_dat\_hold, 2315
  - wr\_dat\_setup, 2315
  - wr\_en\_hi, 2315
  - wr\_en\_lo, 2315
- SEMC\_ClkMode\_t
  - SEMC, 1426
- SEMC\_ColAddrWidth\_t

- SEMC, [1426](#)
- Send
  - IPV6\_NEIGHBOR, [2120](#)
  - UDPPacket, [2403](#)
- send\_message
  - mcanMODM7AE70::mcan\_module, [2165](#)
- Send\_ND\_Solicit
  - IPV6\_NEIGHBOR, [2120](#)
- send\_test\_packet
  - ReceiveUDP, [1904](#)
- SendAndKeep
  - UDPPacket, [2406](#)
- SendAndKeepVialfAddr
  - UDPPacket, [2408](#)
- SendAndKeepVialInterfaceNum
  - UDPPacket, [2409](#)
- SendBreak
  - Serial Interfaces, [1517](#)
- SendCmd
  - WM8904, [2531](#)
- SendCmdList
  - WM8904, [2531](#)
- SendDiagReport
  - Diagnostics (DIAG), [1190](#)
- SendEmailResponse
  - HTTP and HTML Functions, [1265](#)
- sender\_ip
  - ARP - Address Resolution Protocol, [1117](#)
- sender\_phy
  - ARP - Address Resolution Protocol, [1117](#)
- SendFileFragment
  - HTTP and HTML Functions, [1266](#)
- SendFragmentedUdpPacket
  - UDP Socket API, [1774](#)
- SendFullResponse
  - HTTP and HTML Functions, [1266](#), [1267](#)
- SendGifHeader
  - HTTP and HTML Functions, [1267](#)
- sendGratuitousArp
  - ARP - Address Resolution Protocol, [1113](#)
- SendHeaderResponse
  - HTTP and HTML Functions, [1267](#), [1269](#)
- SendHTMLHeader
  - HTTP and HTML Functions, [1269](#)
- SendHTMLHeaderWCookie
  - HTTP and HTML Functions, [1269](#)
- SendICMPError
  - IP, [1319](#)
- SendMail
  - SMTP - Send Email, [1436](#)
- SendMailAuth
  - SMTP - Send Email, [1436](#)
- SendMailAuthAddMIME
  - SMTP - Send Email, [1437](#)
- SendMailAuthEndMIME
  - SMTP - Send Email, [1438](#)
- SendMailAuthStartMIME
  - SMTP - Send Email, [1438](#)
- SendMailEx
  - SMTP - Send Email, [1439](#)
- SendMLDRegistration
  - IPv6Interface, [2140](#)
- SendNextPubReq
  - MQTT::Client, [1972](#)
- SendNextSubReq
  - MQTT::Client, [1973](#)
- SendPending
  - IPV6\_NEIGHBOR, [2121](#)
- SendTextHeader
  - HTTP and HTML Functions, [1270](#)
- SendTFTP
  - TFTP Client, [1716](#)
- sendto
  - UDP Socket API, [1775](#)
- SendToAll
  - Command Processor, [1164](#)
- SendVialfAddr
  - UDPPacket, [2411](#)
- SendVialInterfaceNum
  - UDPPacket, [2412](#)
- Serial, [795](#)
- Serial Application Update, [805](#)
- Serial Callback Function, [711](#)
- Serial HTTP Get Request, [801](#)
- Serial Interfaces, [1514](#)
  - eParityEven, [1515](#)
  - eParityMulti, [1515](#)
  - eParityMultiEven, [1515](#)
  - eParityMultiOdd, [1515](#)
  - eParityNone, [1515](#)
  - eParityOdd, [1515](#)
  - GetUartErrorReg, [1515](#)
  - OpenDefaultSerial, [1516](#)
  - OpenSerial, [1516](#)
  - parity\_mode, [1515](#)
  - SendBreak, [1517](#)
  - Serial485HalfDupMode, [1517](#)
  - SerialClose, [1517](#)
  - SerialEnableHwRxFlow, [1517](#)
  - SerialEnableHwTxFlow, [1518](#)
  - SerialEnableRxFlow, [1518](#)
  - SerialEnableTxFlow, [1518](#)
  - SerialExpandRxBuffer, [1518](#)
  - SerialSendComplete, [1519](#)
  - serwriteaddress, [1519](#)
  - SetRTS, [1519](#)
  - SimpleOpenSerial, [1515](#)
- Serial Port Error Codes, [1519](#)
- Serial Receive Callback, [801](#)
- Serial To Ethernet, [798](#)
- Serial to Serial, [803](#)
- Serial to SPI, [820](#)
- Serial Transceiver Test, [704](#)
- Serial Webserver, [808](#), [1046](#)
- Serial-to-SPI QuadSPI, [725](#)
- Serial-to-SPI USART, [725](#)

- Serial485HalfDupMode
  - Serial Interfaces, [1517](#)
- serialClkFreq
  - \_FlexSPIConfig, [1921](#)
- SerialClose
  - Serial Interfaces, [1517](#)
- SerialConfigServlet, [2315](#)
- SerialEnableHwRxFlow
  - Serial Interfaces, [1517](#)
- SerialEnableHwTxFlow
  - Serial Interfaces, [1518](#)
- SerialEnableRxFlow
  - Serial Interfaces, [1518](#)
- SerialEnableTxFlow
  - Serial Interfaces, [1518](#)
- SerialExpandRxBuffer
  - Serial Interfaces, [1518](#)
- SerialRecord, [2315](#)
  - AssignUartNumber, [2316](#)
  - CloseListenPort, [2316](#)
  - GetCurrentChannelStatus, [2316](#)
  - MakeTcpConnection, [2316](#)
  - MakeUdpConnection, [2316](#)
  - OkToListen, [2316](#)
  - OpenListenPort, [2316](#)
  - OpenSerialPort, [2317](#)
  - ProcessAccept, [2317](#)
  - ProcessListenError, [2317](#)
  - ProcessNetworkError, [2317](#)
  - ProcessReadNetworkData, [2317](#)
  - ProcessSerialError, [2317](#)
  - ProcessSpecialFrameTCPReadSerialData, [2317](#)
  - ProcessSpecialFrameWriteNetworkData, [2317](#)
  - ProcessSpecialFrameWriteTimeout, [2317](#)
  - ProcessTCPReadSerialData, [2317](#)
  - ProcessTimeouts, [2318](#)
  - ProcessWriteNetworkData, [2318](#)
  - ProcessWriteSerialData, [2318](#)
- SerialSendComplete
  - Serial Interfaces, [1519](#)
- serverTask
  - NTPServer, [2248](#)
- serwriteaddress
  - Serial Interfaces, [1519](#)
- set\_config\_defaults
  - mcanMODM7AE70::mcan\_config, [2157](#)
- set\_time
  - Time, [1719](#)
- SetBusSpeed
  - SPI\_QSPI, [2323](#)
  - SPI\_USART, [2328](#)
  - SPIModule, [2339](#), [2340](#)
- SetChoices
  - config\_chooser, [1983](#)
- SetClientCert
  - MQTT::Client, [1973](#)
- setClock
  - TwoWire, [2370](#)
- SetConnectNotify
  - MQTT::Client, [1973](#)
- SetCS
  - DSPIModule, [2069](#)
  - SPI\_QSPI, [2323](#)
  - SPI\_USART, [2328](#)
  - SPIModule, [2340](#)
- SetCustomNetDoRX
  - Low Level Processing (NetDoRx), [1349](#)
- SetDataSize
  - UDPPacket, [2415](#)
- SetDestinationPort
  - UDPPacket, [2415](#)
- SetDSCP
  - UDPPacket, [2416](#)
- SetEnumList
  - config\_string, [2024](#)
- SetFixedDirection
  - MQTT::mqtt\_IO, [2190](#)
- SetFlags
  - MQTT::mqtt\_IO, [2191](#)
  - MQTT::mqtt\_leaf, [2208](#)
- setFn
  - PinIO, [2294](#)
- SetFrom
  - NTPServer::NtpTimeStamp, [2250](#)
- SetFromAscii
  - IPADDR4, [2105](#)
  - IPADDR6, [2110](#)
- SetFromIP4
  - IPADDR6, [2110](#)
- SetFromUint32Shortcut
  - IPADDR6, [2110](#)
- setHighStrength
  - PinIO, [2294](#)
- SetHttpDiag
  - Web Client, [1863](#)
- SetMaxBusDelay
  - BBI2C, [1945](#)
- SetMaxSessionReuseTime
  - SSL/TLS, [1491](#)
- SetMicGain
  - WM8904, [2531](#)
- SetNull
  - config\_IPADDR4, [2003](#)
  - CUR\_IPADDR4, [2048](#)
  - IPADDR4, [2105](#)
  - IPADDR6, [2111](#)
- setNumAddressBytes
  - BBI2C, [1947](#)
  - I2C, [2090](#)
- setoption
  - NB::WebSocket, [2480](#)
- SetOutOfOrderBuffers
  - TCP, [1676](#)
- SetParsePosition
  - ParsedJsonDataSet, [2280](#)
- SetPublishInterval

- MQTT::mqtt\_leaf, [2209](#)
- SetPubState
  - MQTT::mqtt\_leaf, [2209](#)
  - MQTT::mqtt\_obj, [2216](#)
- setPull
  - PinIO, [2294](#)
- SetRateLimit
  - MQTT::Client, [1973](#)
- SetRateLimit\_Publish
  - MQTT::Client, [1974](#)
- SetRateLimit\_Subscribe
  - MQTT::Client, [1974](#)
- SetRTS
  - Serial Interfaces, [1519](#)
- SetSerializationForm
  - MQTT::mqtt\_leaf, [2209](#)
- SetSocketRxBuffers
  - TCP, [1681](#)
- SetSocketTxBuffers
  - TCP, [1682](#)
- SetSocketUnackBuffers
  - TCP, [1683](#)
- setsockopt
  - TCP, [1684](#)
- SetSocksProxySettings
  - SOCKS, [1445](#)
- SetSourcePort
  - UDPPacket, [2416](#)
- SetSR\_IntLevel
  - Interrupt Macro: MOD5441x,NANO54415, SB800EX, [1372](#)
- SetState
  - IPV6\_NEIGHBOR, [2121](#)
- SetStaticDNS
  - IPv6Interface, [2140](#)
- SetStringFormat
  - MQTT::mqtt\_float, [2177](#)
- SetSubState
  - MQTT::mqtt\_leaf, [2209](#)
  - MQTT::mqtt\_obj, [2216](#)
- setup
  - BBI2C, [1947](#)
  - I2C, [2090](#)
  - I2CDevice, [2093](#)
- SetUserAuthLevel
  - UserAuthManager, [2424](#)
- SetVolume
  - WM8904, [2531](#)
- SetWifiSPISpeed
  - Wifi, [1886](#)
- setWireTimeout
  - TwoWire, [2370](#)
- SHA1 Digest, [809](#)
- Short\_name
  - TimeZoneRecord, [2364](#)
- Show Configuration, [414](#)
- Show Network Interfaces, [810](#)
- ShowAllInfo
  - IPv6Interface, [2141](#)
- ShowAllV6Info
  - DHCPv6 Client, [1179](#)
- ShowArp
  - ARP - Address Resolution Protocol, [1115](#)
- ShowBuffer
  - Buffers - System Buffer Pool, [1125](#)
- ShowData
  - Utilities, [1794](#)
- ShowInfo
  - IPv6Interface, [2141](#)
- ShowIP4
  - Utilities, [1794](#)
- ShowIP6
  - Utilities, [1795](#)
- ShowIP6Counters
  - IPv6 Interface Management, [1323](#)
- ShowIPBuffer
  - IP, [1320](#)
- ShowMac
  - Utilities, [1795](#)
- ShowRingData
  - Utilities, [1795](#)
- Shutdown
  - SAICtx\_t, [2309](#)
- Shutdown Notifications, [1520](#)
  - NBApproveShutdown, [1520](#)
  - NBFaultNotify, [1522](#)
- ShutdownReasons, [1522](#)
- Sign Board, [380](#)
- signal\_handler
  - ReceiveUDP, [1904](#)
- Signed Application, [382](#)
- Signed Application Update, [1523](#)
  - ProgramApplication, [1523](#)
  - RegisterAppSigningPublicKey, [1523](#)
  - UpdateFromStream, [1524](#)
- Simple ADC, [607](#)
- Simple FTP Server (FTPD), [496](#)
- Simple HTML, [1048](#)
- Simple JSON HTML Example, [557](#)
- Simple JSON Post Receiver Example, [560](#)
- SimpleBufferObject, [2318](#)
- SimpleOpenSerial
  - Serial Interfaces, [1515](#)
- siprintf
  - NBString, [2244](#)
- size
  - NBString, [2245](#)
- SkipCurrentValue
  - ParsedJsonDataSet, [2280](#)
- SLAST
  - \_edma\_tcd, [1909](#)
- SMPoolPtr, [2319](#)
  - SMPoolIPtr, [2319](#)
- SMTP - Send Email, [1434](#)
  - CONTENT\_TYPE\_BINARY\_ATTACH, [1435](#)
  - CONTENT\_TYPE\_END, [1435](#)

- CONTENT\_TYPE\_ENUM, [1435](#)
- CONTENT\_TYPE\_HTML\_DECOMP, [1435](#)
- CONTENT\_TYPE\_PLAIN\_TEXT, [1435](#)
- CONTENT\_TYPE\_PLAIN\_TEXT\_ATTACH, [1435](#)
- IsMailError, [1435](#)
- PrintNError, [1435](#)
- PrintServerLog, [1436](#)
- SendMail, [1436](#)
- SendMailAuth, [1436](#)
- SendMailAuthAddMIME, [1437](#)
- SendMailAuthEndMIME, [1438](#)
- SendMailAuthStartMIME, [1438](#)
- SendMailEx, [1439](#)
- SMTP Error Codes, [1440](#)
- snShowIP4
  - Utilities, [1796](#)
- snShowIP6
  - Utilities, [1796](#)
- snShowMac
  - Utilities, [1796](#)
- socket\_crit
  - NB::WebSocket, [2519](#)
- SocketPeek
  - TCP, [1688](#)
- SocketPeekWithTimeout
  - IOSYS - I/O System, [1294](#)
- SocketReadWithTimeout
  - TCP, [1689](#)
- SOCKS, [1444](#)
  - AuthWithGssApi, [1445](#)
  - eSocksAdrTypeDomain, [1445](#)
  - eSocksAdrTypeIpv4, [1445](#)
  - eSocksAdrTypeIpv6, [1445](#)
  - eSocksAdrTypeNone, [1445](#)
  - eSocksAuthTypeGssApi, [1445](#)
  - eSocksAuthTypeNoAuth, [1445](#)
  - eSocksAuthTypeUnPw, [1445](#)
  - eSocksClientCmdBind, [1445](#)
  - eSocksClientCmdConnect, [1445](#)
  - eSocksClientCmdUdpAssoc, [1445](#)
  - GetSocksProxySettings, [1445](#)
  - SetSocksProxySettings, [1445](#)
  - SocksAdrType, [1444](#)
  - SocksAuthType, [1445](#)
  - SocksClientCmd, [1445](#)
- SOCKS Error Codes, [1446](#)
- SOCKS5 Client, [820](#)
- SocksAdrType
  - SOCKS, [1444](#)
- SocksAuthType
  - SOCKS, [1445](#)
- SocksClientCmd
  - SOCKS, [1445](#)
- SOFF
  - \_edma\_tcd, [1909](#)
- SOMRT1061, [726](#), [1404](#)
- SOMRT1061 Pin Function Constants, [1247](#)
- SOMRT1061 Pin IRQ Example, [736](#)
- SOMRT1061 Platform Reference, [113](#)
- SOMRT1061 Temperature Monitor, [1432](#)
  - \_tempmon\_alarm\_mode, [1432](#)
  - kTEMPMON\_HighAlarmMode, [1433](#)
  - kTEMPMON\_LowAlarmMode, [1433](#)
  - kTEMPMON\_PanicAlarmMode, [1433](#)
  - TEMPMON\_Deinit, [1433](#)
  - TEMPMON\_GetCurrentTemperature, [1433](#)
  - TEMPMON\_GetDefaultConfig, [1433](#)
  - TEMPMON\_Init, [1433](#)
  - TEMPMON\_SetTempAlarm, [1434](#)
- Source
  - IPV6\_PREFIX, [2124](#)
- SpaceAvail
  - fifo\_buffer\_storage, [2081](#)
- spaceleft
  - HAL - Hardware Abstraction Layer, [1256](#)
- SpaceUsed
  - fifo\_buffer\_storage, [2081](#)
- SparkFun Qwiic Alphanumeric Display, [515](#)
- SparkFun Qwiic Joystick, [517](#)
- SparkFun Qwiic KeyPad, [521](#)
- SparkFun Qwiic OLED Display, [523](#)
- SPI, [725](#), [820](#), [1447](#)
- SPI PicKit Demo Board, [773](#), [820](#)
- SPI\_QSPI, [2319](#)
  - Init, [2322](#)
  - Rx, [2323](#)
  - SetBusSpeed, [2323](#)
  - SetCS, [2323](#)
  - SPI\_QSPI, [2321](#)
  - Start, [2324](#)
  - Tx, [2324](#)
- SPI\_USART, [2325](#)
  - Init, [2327](#)
  - Rx, [2328](#)
  - SetBusSpeed, [2328](#)
  - SetCS, [2328](#)
  - SPI\_USART, [2326](#)
  - Start, [2329](#)
  - Tx, [2329](#)
- spiChipSelect
  - MCF5441x (DSPI), [1453](#)
  - MIMX10xx (DSPI), [1463](#)
- spiChipSelectPolarity
  - MCF5441x (DSPI), [1453](#)
  - MIMX10xx (DSPI), [1464](#)
  - SAME70 (DSPI), [1469](#)
- spiDriverStruct, [2330](#)
- SPIModule, [2330](#)
  - ClrSem, [2335](#)
  - Done, [2335](#)
  - GetActualBaudrate, [2336](#)
  - GetSem, [2336](#)
  - Init, [2336](#), [2337](#)
  - RegisterSem, [2338](#)
  - Rx, [2339](#)
  - SetBusSpeed, [2339](#), [2340](#)

- SetCS, [2340](#)
- SPIModule, [2332–2334](#)
- Start, [2340](#), [2341](#)
- Tx, [2342](#)
- sprintf
  - IPADDR4, [2105](#)
  - IPADDR6, [2111](#)
  - NBString, [2245](#)
- SR\_CLR\_FLAGS
  - MCF5441x (DSPI), [1452](#)
- SR\_EOQF\_MASK
  - MCF5441x (DSPI), [1452](#)
- SRAM\_Timing\_Spec\_t, [2342](#)
- srcAddr
  - \_edma\_transfer\_config, [1910](#)
- srcOffset
  - \_edma\_transfer\_config, [1910](#)
- srcTransferSize
  - \_edma\_transfer\_config, [1910](#)
- SSC I2S Interface, [725](#)
- SSC\_cfg\_t, [2343](#)
  - clkDiv, [2343](#)
  - rx, [2343](#)
  - tx, [2344](#)
- SSC\_rxtx\_cfg\_t, [2344](#)
  - bitOrder, [2344](#)
  - bitsPerWord, [2344](#)
  - clkGate, [2344](#)
  - clkOut, [2345](#)
  - clkSrc, [2345](#)
  - dataValid, [2345](#)
  - depletionBehavior, [2345](#)
  - enable, [2345](#)
  - linIdleState, [2345](#)
  - period, [2345](#)
  - startCond, [2345](#)
  - startDly, [2345](#)
  - syncDataEnabled, [2345](#)
  - syncEdge, [2345](#)
  - syncLen, [2345](#)
  - syncOut, [2346](#)
  - wordsPerFrame, [2346](#)
- SSCCtx\_t, [2346](#)
  - getCurrentConfig, [2346](#)
  - getState, [2347](#)
  - Init, [2347](#)
  - ReadyReceiveBuffer, [2347](#)
  - TransmitBuffer, [2347](#)
- SSH, [821](#), [1475](#)
  - NbSshInit, [1478](#)
  - SshAccept, [1479](#)
  - SshClientGetUserKeyFn, [1479](#)
  - SshClientGetUserPasswordFn, [1479](#)
  - SshClientSetGetUserKeyFn, [1480](#)
  - SshClientSetGetUserPasswordFn, [1480](#)
  - SshClrSockOption, [1480](#)
  - SshConnect, [1481](#)
  - SshGetKeySize, [1481](#)
  - SshGetSockOption, [1481](#)
  - SshGetUserAuthenticate, [1482](#)
  - SshGetUserAuthenticateWithType, [1482](#)
  - SshGetUserGetKey, [1482](#)
  - sshGetUserKeyFn, [1477](#)
  - sshGetUserPwFn, [1477](#)
  - SshNegotiateSession, [1483](#)
  - SshNegotiateSessionClient, [1483](#)
  - sShowCounters
- SshGetSockOption, [1481](#)
- SshGetUserAuthenticate, [1482](#)
- SshGetUserAuthenticateWithType, [1482](#)
- SshGetUserGetKey, [1482](#)
- sshGetUserKeyFn, [1477](#)
- sshGetUserPwFn, [1477](#)
- SshNegotiateSession, [1483](#)
- SshNegotiateSessionClient, [1483](#)
- SshPrintStatistics, [1483](#)
- SshSetBannerText, [1483](#)
- SshSetSockOption, [1484](#)
- SshSetUserAuthenticate, [1484](#)
- SshSetUserAuthenticateWithType, [1484](#)
- SshSetUserGetKey, [1485](#)
- sshUserAuthenticateFn, [1477](#)
- sshUserAuthenticateWithTypeFn, [1478](#)
- sshUserGetKeyFn, [1478](#)
- SshValidateKey, [1485](#)
- SshWritePublicKey, [1485](#)
- SSH Client, [827](#)
- SSH Error Codes, [1486](#)
- SSH Programming Guide, [311](#)
- SSH Server, [821](#)
- SSH Server with User Authorization, [830](#)
- SSH Server with User Key, [824](#)
- SshAccept
  - SSH, [1479](#)
- SshClientGetUserKeyFn
  - SSH, [1479](#)
- SshClientGetUserPasswordFn
  - SSH, [1479](#)
- SshClientSetGetUserKeyFn
  - SSH, [1480](#)
- SshClientSetGetUserPasswordFn
  - SSH, [1480](#)
- SshClrSockOption
  - SSH, [1480](#)
- SshConnect
  - SSH, [1481](#)
- SshGetKeySize
  - SSH, [1481](#)
- SshGetSockOption
  - SSH, [1481](#)
- SshGetUserAuthenticate
  - SSH, [1482](#)
- SshGetUserAuthenticateWithType
  - SSH, [1482](#)
- SshGetUserGetKey
  - SSH, [1482](#)
- sshGetUserKeyFn
  - SSH, [1477](#)
- sshGetUserPwFn
  - SSH, [1477](#)
- SshNegotiateSession
  - SSH, [1483](#)
- SshNegotiateSessionClient
  - SSH, [1483](#)



- Utilities, [1797](#)
- SshPrintStatistics
  - SSH, [1483](#)
- SshSetBannerText
  - SSH, [1483](#)
- SshSetSockOption
  - SSH, [1484](#)
- SshSetUserAuthenticate
  - SSH, [1484](#)
- SshSetUserAuthenticateWithType
  - SSH, [1484](#)
- SshSetUserGetKey
  - SSH, [1485](#)
- sshUserAuthenticateFn
  - SSH, [1477](#)
- sshUserAuthenticateWithTypeFn
  - SSH, [1478](#)
- sshUserGetKeyFn
  - SSH, [1478](#)
- SshValidateKey
  - SSH, [1485](#)
- SshWritePublicKey
  - SSH, [1485](#)
- SSID\_NotFound
  - NB::Error::Connect, [1902](#)
- SSL Error Codes, [1513](#)
- SSL SMTP Functions, [1440](#)
  - SSL\_SendMail, [1441](#)
  - SSL\_SendMailHTML, [1442](#)
  - SSL\_SendMailStartMIME, [1443](#)
- SSL/TLS, [838](#), [1486](#)
  - E\_BAD\_EXPIRATION\_VALUE, [1489](#)
  - E\_CERT\_EXPIRED, [1489](#)
  - E\_CERT\_VALID, [1489](#)
  - E\_NO\_SYSTEM\_TIME, [1489](#)
  - ExpireRetCodes, [1489](#)
  - FreeSSLSockets, [1490](#)
  - GetCertificateLen, [1490](#)
  - GetCertificatePEM, [1490](#)
  - GetPrivateKeyLen, [1490](#)
  - GetPrivateKeyPEM, [1490](#)
  - IsSSL\_CertNKeyValid, [1491](#)
  - IsSSLfd, [1507](#)
  - SetMaxSessionReuseTime, [1491](#)
  - SSL\_accept, [1492](#)
  - SSL\_AddCertToClientCaList, [1492](#)
  - SSL\_AddCertToServerCaList, [1493](#)
  - SSL\_ClearClientCaList, [1493](#)
  - SSL\_ClearServerCaList, [1493](#)
  - SSL\_ClientReadyCert, [1494](#)
  - SSL\_clrsockoption, [1508](#)
  - SSL\_connect, [1494](#), [1495](#)
  - SSL\_GetAltNameFromCert, [1496](#)
  - SSL\_GetCnFromCert, [1496](#)
  - SSL\_GetExpirationDate, [1497](#)
  - SSL\_GetIssuerFromCert, [1497](#)
  - SSL\_GetRawSocket, [1508](#)
  - SSL\_getsocketerror, [1498](#)
  - SSL\_GetSocketInterface, [1498](#)
  - SSL\_GetSocketLocalAddr, [1498](#)
  - SSL\_GetSocketLocalPort, [1499](#)
  - SSL\_GetSocketRemoteAddr, [1499](#)
  - SSL\_GetSocketRemotePort, [1500](#)
  - SSL\_getsockoption, [1508](#)
  - SSL\_GetSubjectNameFromCert, [1500](#)
  - SSL\_GetTcpRtxCount, [1500](#)
  - SSL\_GetTimeToExpire, [1501](#)
  - SSL\_IsCertAddedToClientCaList, [1501](#)
  - SSL\_IsCertAddedToServerCaList, [1501](#)
  - SSL\_IsCertExpired, [1502](#)
  - SSL\_makeSecure, [1502](#)
  - SSL\_negotiate, [1503](#)
  - SSL\_ServerReadyCert, [1504](#)
  - SSL\_setsockoption, [1509](#)
  - SSL\_TcpGetLastRxInterval, [1504](#)
  - SSL\_TcpGetLastRxTime, [1504](#)
  - SSL\_TcpSendKeepAlive, [1505](#)
  - SSLGetRxBufferSpaceUsed, [1505](#)
  - SSLGetTxBufferAvailSpace, [1505](#)
  - SslInit, [1506](#)
  - SSLReadWithTimeout, [1506](#)
  - SslSetVerifyPeerClient, [1506](#)
  - SslSetVerifyPeerServer, [1507](#)
  - SSLSocketsAvail, [1507](#)
- SSL/TLS and Certificates, [289](#)
- SSL/TLS Client, [880](#)
- SSL/TLS Client Certificate, [882](#)
- SSL/TLS Client Verify Peer - Basic, [885](#)
- SSL/TLS Client Verify Peer - EFFT-STD, [888](#)
- SSL/TLS Receive Mail (POP3), [892](#)
- SSL/TLS Send Mail, [894](#)
- SSL/TLS Send Mail w/ Attachment, [898](#)
- SSL/TLS Send Mail w/ EFFT Atchment, [901](#)
- SSL/TLS Server, [904](#)
- SSL\_accept
  - SSL/TLS, [1492](#)
- SSL\_AddCertToClientCaList
  - SSL/TLS, [1492](#)
- SSL\_AddCertToServerCaList
  - SSL/TLS, [1493](#)
- SSL\_ClearClientCaList
  - SSL/TLS, [1493](#)
- SSL\_ClearServerCaList
  - SSL/TLS, [1493](#)
- SSL\_ClientReadyCert
  - SSL/TLS, [1494](#)
- SSL\_clrsockoption
  - SSL/TLS, [1508](#)
- SSL\_connect
  - SSL/TLS, [1494](#), [1495](#)
- SSL\_CreateNewSelfSignedCert
  - Onboard Self-Signed Certificate Generation, [1513](#)
- SSL\_GetAltNameFromCert
  - SSL/TLS, [1496](#)
- SSL\_GetCnFromCert
  - SSL/TLS, [1496](#)



- SSL\_GetExpirationDate
  - SSL/TLS, [1497](#)
- SSL\_GetIssuerFromCert
  - SSL/TLS, [1497](#)
- SSL\_GetRawSocket
  - SSL/TLS, [1508](#)
- SSL\_getsocketerror
  - SSL/TLS, [1498](#)
- SSL\_GetSocketInterface
  - SSL/TLS, [1498](#)
- SSL\_GetSocketLocalAddr
  - SSL/TLS, [1498](#)
- SSL\_GetSocketLocalPort
  - SSL/TLS, [1499](#)
- SSL\_GetSocketRemoteAddr
  - SSL/TLS, [1499](#)
- SSL\_GetSocketRemotePort
  - SSL/TLS, [1500](#)
- SSL\_getsockopt
  - SSL/TLS, [1508](#)
- SSL\_GetSubjectNameFromCert
  - SSL/TLS, [1500](#)
- SSL\_GetTcpRtxCount
  - SSL/TLS, [1500](#)
- SSL\_GetTimeToExpire
  - SSL/TLS, [1501](#)
- SSL\_IsCertAddedToClientCaList
  - SSL/TLS, [1501](#)
- SSL\_IsCertAddedToServerCaList
  - SSL/TLS, [1501](#)
- SSL\_IsCertExpired
  - SSL/TLS, [1502](#)
- SSL\_KEY\_ECC\_SECP192R1
  - Onboard Self-Signed Certificate Generation, [1512](#)
- SSL\_KEY\_ECC\_SECP224R1
  - Onboard Self-Signed Certificate Generation, [1512](#)
- SSL\_KEY\_ECC\_SECP256R1
  - Onboard Self-Signed Certificate Generation, [1512](#)
- SSL\_KEY\_ECC\_SECP384R1
  - Onboard Self-Signed Certificate Generation, [1512](#)
- SSL\_KEY\_ECC\_SECP521R1
  - Onboard Self-Signed Certificate Generation, [1512](#)
- SSL\_KEY\_RSA\_1024
  - Onboard Self-Signed Certificate Generation, [1512](#)
- SSL\_KEY\_RSA\_2048
  - Onboard Self-Signed Certificate Generation, [1512](#)
- SSL\_KEY\_RSA\_4096
  - Onboard Self-Signed Certificate Generation, [1512](#)
- SSL\_makeSecure
  - SSL/TLS, [1502](#)
- SSL\_negotiate
  - SSL/TLS, [1503](#)
- SSL\_POP3\_InitializeSession
  - POP3 - Post Office Protocol, [1365](#)
- SSL\_SendMail
  - SSL SMTP Functions, [1441](#)
- SSL\_SendMailHTML
  - SSL SMTP Functions, [1442](#)
- SSL\_SendMailStartMIME
  - SSL SMTP Functions, [1443](#)
- SSL\_ServerReadyCert
  - SSL/TLS, [1504](#)
- SSL\_setsockopt
  - SSL/TLS, [1509](#)
- SSL\_TcpGetLastRxInterval
  - SSL/TLS, [1504](#)
- SSL\_TcpGetLastRxTime
  - SSL/TLS, [1504](#)
- SSL\_TcpSendKeepAlive
  - SSL/TLS, [1505](#)
- SSLGetRxBufferSpaceUsed
  - SSL/TLS, [1505](#)
- SSLGetTxBufferAvailSpace
  - SSL/TLS, [1505](#)
- SslInit
  - SSL/TLS, [1506](#)
- SslKeyType\_t
  - Onboard Self-Signed Certificate Generation, [1512](#)
- SSLReadWithTimeout
  - SSL/TLS, [1506](#)
- SslSetVerifyPeerClient
  - SSL/TLS, [1506](#)
- SslSetVerifyPeerServer
  - SSL/TLS, [1507](#)
- SSLSocketsAvail
  - SSL/TLS, [1507](#)
- Stack Protection, [910](#)
- Standard File System (EFS-STD), [467](#)
- Start
  - DSPIModule, [2070](#)
  - SPI\_QSPI, [2324](#)
  - SPI\_USART, [2329](#)
  - SPIModule, [2340](#), [2341](#)
  - StopWatch, [2353](#)
- start
  - NTPServer, [2248](#)
- START\_CMP\_0
  - SAI Configuration Enumerations, [1532](#)
- START\_CONTINUOUS
  - SAI Configuration Enumerations, [1532](#)
- START\_FRAME\_EDGE
  - SAI Configuration Enumerations, [1532](#)
- START\_FRAME\_FALLING
  - SAI Configuration Enumerations, [1532](#)
- START\_FRAME\_HIGH
  - SAI Configuration Enumerations, [1532](#)
- START\_FRAME\_LEVEL
  - SAI Configuration Enumerations, [1532](#)
- START\_FRAME\_LOW
  - SAI Configuration Enumerations, [1532](#)
- START\_FRAME\_RISING
  - SAI Configuration Enumerations, [1532](#)
- START\_SYNC\_RX\_TX
  - SAI Configuration Enumerations, [1532](#)
- startCond
  - SAI\_rtx\_cfg\_t, [2302](#)

- SSC\_rtx\_cfg\_t, [2345](#)
- startCond\_t
  - SAI Configuration Enumerations, [1532](#)
- StartConnection
  - Web Client, [1863](#)
- StartDHCP
  - DhcpObject, [2057](#)
- StartDHCPv6
  - DHCPv6 Client, [1180](#)
- StartDHCPv6\_InfoReq
  - DHCPv6 Client, [1180](#)
- StartDHCPv6\_Solicit
  - DHCPv6 Client, [1180](#)
- startDly
  - SAI\_rtx\_cfg\_t, [2302](#)
  - SSC\_rtx\_cfg\_t, [2345](#)
- StartHttp
  - HTTP and HTML Functions, [1270](#)
  - Initialization - System Initialization Functions, [1329](#)
- StartHttpCustom
  - Initialization - System Initialization Functions, [1329](#)
- StartHttps
  - HTTP and HTML Functions, [1270](#)
  - Initialization - System Initialization Functions, [1329](#)
- StartTask
  - MQTT::Client, [1975](#)
- StartTransaction
  - Web Client, [1864](#)
- StartWebClient
  - Web Client, [1865](#), [1866](#)
- state
  - \_lpspi\_master\_edma\_handle, [1926](#)
  - \_lpspi\_master\_handle, [1929](#)
  - \_lpspi\_slave\_edma\_handle, [1933](#)
  - \_lpspi\_slave\_handle, [1935](#)
- STD File System Seek Codes, [1198](#)
- StdioCheckIntc
  - HAL - Hardware Abstraction Layer, [1256](#)
- Stepper Interval Timer, [714](#)
- Stepper Step Count, [716](#)
- StillValidToSend
  - IPV6\_NEIGHBOR, [2121](#)
- stod
  - NBString, [2245](#)
- stoi
  - NBString, [2245](#)
- stol
  - NBString, [2245](#)
- Stop
  - StopWatch, [2354](#)
- stop
  - NTPServer, [2248](#)
- StopDHCP
  - DhcpObject, [2057](#)
- StopHttp
  - HTTP and HTML Functions, [1271](#)
- StopWatch, [2348](#)
  - Clear, [2349](#)
- Convert, [2350](#)
- CountResolution, [2351](#)
- GetTime, [2352](#)
- Start, [2353](#)
- Stop, [2354](#)
- StopWatch, [2349](#)
- Stopwatch Timer, [963](#), [1739](#)
- stoui
  - NBString, [2246](#)
- stoul
  - NBString, [2246](#)
- strcpy
  - NBString, [2246](#)
- Stream Update, [1525](#)
  - ReadBinaryApplicationCodeFromStream, [1525](#)
  - ReadS19ApplicationCodeFromStream, [1525](#)
- Stream Update Return Values, [1526](#)
- STRING
  - JSON Lexer, [1331](#)
- STRING\_CHUNK
  - JSON Lexer, [1332](#)
- STRING\_TOO\_BIG
  - JSON Lexer, [1332](#)
- StringBufferObject, [2355](#)
- SubHandler
  - MQTT::SubHandler, [2357](#)
- Subscribe
  - MQTT::Client, [1975](#)
- SubscribeEvent
  - MQTT::ConfigExposer, [2043](#)
  - MQTT::mqtt\_bool, [2172](#)
  - MQTT::mqtt\_float, [2178](#)
  - MQTT::mqtt\_int, [2184](#)
  - MQTT::mqtt\_IO, [2191](#)
  - MQTT::mqtt\_leaf, [2210](#)
  - MQTT::mqtt\_obj, [2216](#)
  - MQTT::mqtt\_string, [2224](#)
  - MQTT::mqtt\_uint, [2230](#)
  - MQTT::SubHandler, [2358](#)
- SubState
  - MQTT::mqtt\_leaf, [2210](#)
- substr
  - NBString, [2246](#)
- Success
  - NB::Error::Connect, [1902](#)
  - NB::Error::Init, [1903](#)
  - NB::Error::Scan, [1903](#)
- swap
  - NBString, [2247](#)
- Sync
  - SEMC, [1426](#)
- SYNC\_CH\_CH
  - SAI Configuration Enumerations, [1531](#)
- SYNC\_CH\_RX
  - SAI Configuration Enumerations, [1531](#)
- SYNC\_CH\_TX
  - SAI Configuration Enumerations, [1531](#)
- syncCh

- SAI\_rtx\_cfg\_t, [2303](#)
- syncDataEnabled
  - SSC\_rtx\_cfg\_t, [2345](#)
- syncEdge
  - SAI\_rtx\_cfg\_t, [2303](#)
  - SSC\_rtx\_cfg\_t, [2345](#)
- Synchronous Audio Interface (SOMRT1061), [1526](#)
  - sai, [1527](#)
- syncLen
  - SAI\_rtx\_cfg\_t, [2303](#)
  - SSC\_rtx\_cfg\_t, [2345](#)
- syncOut
  - SAI\_rtx\_cfg\_t, [2303](#)
  - SSC\_rtx\_cfg\_t, [2346](#)
- SysLog
  - Syslog, [1536](#)
- Syslog, [913](#), [1535](#)
  - SysLog, [1536](#)
  - SysLogAddress, [1540](#)
  - SysLogVia, [1538](#)
- SysLogAddress
  - Syslog, [1540](#)
- SysLogCheckIntc
  - HAL - Hardware Abstraction Layer, [1256](#)
- SysLogVia
  - Syslog, [1538](#)
- System Diagnostics, [365](#), [915](#)
- System Functions, [1540](#)
  - GetReleaseTag, [1541](#)
  - GetUserParameters, [1541](#)
  - SaveUserParameters, [1541](#)
- tail
  - \_edma\_handle, [1907](#)
- target\_ip
  - ARP - Address Resolution Protocol, [1117](#)
- target\_phy
  - ARP - Address Resolution Protocol, [1118](#)
- TCD\_ATTR\_16BIT\_TRANS
  - MCF5441x (DSPI), [1452](#)
- TCD\_ATTR\_8BIT\_TRANS
  - MCF5441x (DSPI), [1452](#)
- TCD\_CSR\_DISABLE\_REQ
  - MCF5441x (DSPI), [1452](#)
- TCD\_CSR\_DONE\_BIT
  - MCF5441x (DSPI), [1452](#)
- TCD\_CSR\_DREQ\_INT\_MAJOR
  - MCF5441x (DSPI), [1452](#)
- TCD\_XITER\_CNT\_MASK
  - MCF5441x (DSPI), [1452](#)
- TCD\_XOFF\_0uint8\_t
  - MCF5441x (DSPI), [1453](#)
- TCD\_XOFF\_1uint8\_t
  - MCF5441x (DSPI), [1453](#)
- TCD\_XOFF\_2uint8\_t
  - MCF5441x (DSPI), [1453](#)
- tcdPool
  - \_edma\_handle, [1907](#)
- tcdSize
  - \_edma\_handle, [1907](#)
- tcdUsed
  - \_edma\_handle, [1907](#)
- TCP, [917](#), [1542](#)
  - abortsocket, [1550](#)
  - accept, [1551](#), [1557](#)
  - clrsockoption, [1560](#)
  - connect, [1563](#), [1570](#)
  - connectvia, [1577](#), [1585](#), [1594](#), [1600](#)
  - connectwlocal, [1608](#), [1616](#)
  - DumpTcpDebug, [1623](#)
  - EnableTcpDebug, [1624](#)
  - getsocketerror, [1626](#)
  - GetSocketLocalAddr, [1631](#)
  - GetSocketLocalPort, [1632](#)
  - GetSocketRemoteAddr, [1632](#)
  - GetSocketRemotePort, [1633](#)
  - getsockoption, [1634](#)
  - GetTcpRtxCount, [1638](#)
  - listen, [1643](#)
  - listenvia, [1650](#), [1652](#)
  - NoBlockConnect, [1660](#)
  - NoBlockConnectVia, [1665](#)
  - NoBlockConnectwlocal, [1670](#)
  - SetOutOfOrderBuffers, [1676](#)
  - SetSocketRxBuffers, [1681](#)
  - SetSocketTxBuffers, [1682](#)
  - SetSocketUnackBuffers, [1683](#)
  - setsockoption, [1684](#)
  - SocketPeek, [1688](#)
  - SocketReadWithTimeout, [1689](#)
  - TcpAllDataAcked, [1691](#)
  - TcpGetLastRxInterval, [1691](#)
  - TcpGetLastRxTime, [1695](#)
  - TcpGetRxBufferSpaceUsed, [1700](#)
  - TcpGetSocketInterface, [1700](#)
  - TcpGetSocketState, [1702](#)
  - TcpGetTxBufferAvailSpace, [1703](#)
  - TcpGetTxDataWaiting, [1703](#)
  - TcpSendKeepAlive, [1703](#)
  - WaitForSocketFlush, [1706](#)
- TCP Client, [918](#)
- TCP Keepalive Multiple, [925](#)
- TCP Keepalive Simple, [923](#)
- TCP Multi-Socket Server, [931](#)
- TCP No Block Connect Test, [920](#)
- TCP Notification System, [1707](#)
  - RegisterTCPReadNotify, [1709](#)
  - RegisterTCPWriteNotify, [1711](#)
  - tcp\_notify\_handler, [1707](#)
- TCP Prococol, [260](#)
- TCP Resource Information, [933](#)
- TCP Server, [935](#)
- TCP Server Using Via, [937](#)
- TCP Socket Options, [1713](#)
- TCP Socket State, [1714](#)
- TCP Socket Status, [1715](#)
- TCP Speed Test, [940](#)

- TCP Stress Test, [943](#)
- TCP to Serial, [805](#)
- tcp\_notify\_handler
  - TCP Notification System, [1707](#)
- TcpAllDataAcked
  - TCP, [1691](#)
- TcpGetLastRxInterval
  - TCP, [1691](#)
- TcpGetLastRxTime
  - TCP, [1695](#)
- TcpGetRxBufSpaceUsed
  - TCP, [1700](#)
- TcpGetSocketInterface
  - TCP, [1700](#)
- TcpGetSocketState
  - TCP, [1702](#)
- TcpGetTxBufferAvailSpace
  - TCP, [1703](#)
- TcpGetTxDataWaiting
  - TCP, [1703](#)
- TcpSendKeepAlive
  - TCP, [1703](#)
- tdc\_enable
  - mcanMODM7AE70::mcan\_config, [2159](#)
- tDelete
  - HTTP and HTML Functions, [1261](#)
- tdf\_cycles
  - EBI - External Bus Interface, [1403](#)
- Telnet Command, [945](#)
- TEMPMON\_Deinit
  - SOMRT1061 Temperature Monitor, [1433](#)
- TEMPMON\_GetCurrentTemperature
  - SOMRT1061 Temperature Monitor, [1433](#)
- TEMPMON\_GetDefaultConfig
  - SOMRT1061 Temperature Monitor, [1433](#)
- TEMPMON\_Init
  - SOMRT1061 Temperature Monitor, [1433](#)
- TEMPMON\_SetTempAlarm
  - SOMRT1061 Temperature Monitor, [1434](#)
- TestNewValue
  - CUR\_IPADDR4, [2048](#)
  - Dynamic\_IPADDR, [2073](#)
- TestOrSetValue
  - MQTT::ConfigExposer, [2043](#)
  - MQTT::mqtt\_bool, [2172](#)
  - MQTT::mqtt\_float, [2178](#)
  - MQTT::mqtt\_int, [2185](#)
  - MQTT::mqtt\_IO, [2191](#)
  - MQTT::mqtt\_leaf, [2210](#)
  - MQTT::mqtt\_obj, [2217](#)
  - MQTT::mqtt\_string, [2224](#)
  - MQTT::mqtt\_uint, [2231](#)
  - MQTT::SubHandler, [2360](#)
- TFTP, [1715](#)
- TFTP Client, [948](#), [1715](#)
  - GetTFTP, [1716](#)
  - SendTFTP, [1716](#)
- TFTP Return Codes, [1717](#)
- TFTP Server, [952](#), [1717](#)
- TFTP\_Server, [2361](#)
  - Add, [2361](#)
  - TFTP\_Server, [2361](#)
- TFTP\_Server\_source\_File, [2362](#)
- tGet
  - HTTP and HTML Functions, [1261](#)
- tgl
  - PinIO, [2294](#)
- tHead
  - HTTP and HTML Functions, [1261](#)
- TICKS\_PER\_SECOND
  - Constants (constants.h), [1175](#)
- TicTacToe, [1050](#)
- Time, [1718](#)
  - set\_time, [1719](#)
  - time, [1719](#)
  - timegm, [1720](#)
  - TZRecords, [1720](#)
  - tzsetchar, [1720](#)
- time
  - Time, [1719](#)
- Time Functions, [954](#)
- timegm
  - Time, [1720](#)
- Timeout
  - NB::Error, [1902](#)
- timeout\_enable
  - mcanMODM7AE70::mcan\_config, [2159](#)
- timeout\_mode
  - mcanMODM7AE70::mcan\_config, [2160](#)
- timeout\_period
  - mcanMODM7AE70::mcan\_config, [2160](#)
- Timer, [958](#)
- Timer Capture, [712](#)
- Timer Waveform Output, [713](#)
- Timers, [711](#), [958](#), [1722](#)
- timestamp
  - \_flexcan\_frame, [1915](#)
  - \_flexcan\_handle, [1916](#)
- timestamp\_prescaler
  - mcanMODM7AE70::mcan\_config, [2160](#)
- TimeZoneRecord, [2362](#)
  - bUsCanada, [2363](#)
  - Description, [2363](#)
  - Name, [2363](#)
  - Posix, [2364](#)
  - Short\_name, [2364](#)
- to\_ipaddr
  - NBString, [2246](#)
- toggle
  - PinIO, [2294](#)
- TooManyPendingCommands
  - NB::Error, [1902](#)
- totalByteCount
  - \_lpspi\_master\_edma\_handle, [1926](#)
  - \_lpspi\_master\_handle, [1929](#)
  - \_lpspi\_slave\_edma\_handle, [1933](#)

- [\\_lpspi\\_slave\\_handle](#), 1935
- tPost
  - HTTP and HTML Functions, 1261
- tPut
  - HTTP and HTML Functions, 1261
- transmit\_pause
  - mcanMODM7AE70::mcan\_config, 2160
- TransmitBuffer
  - SAICtx\_t, 2309
  - SSCCtx\_t, 2347
  - WM8904, 2532
- transmitCommand
  - [\\_lpspi\\_master\\_edma\\_handle](#), 1926
- TRUE\_EL
  - JSON Lexer, 1332
- tUnknown
  - HTTP and HTML Functions, 1261
- turnaround
  - SEMC\_cfg\_t, 2315
- TwoWire, 2365
  - available, 2367
  - begin, 2367
  - beginTransmission, 2367
  - endTransmission, 2367
  - onReceive, 2368
  - onRequest, 2368
  - ping, 2368
  - read, 2368
  - readReg8, 2369
  - readRegN, 2369
  - requestFrom, 2369
  - setClock, 2370
  - setWireTimeout, 2370
  - TwoWire, 2366
  - write, 2370, 2371
  - writeReg8, 2371
  - writeRegN, 2372
- Tx
  - DSPIModule, 2070
  - SPI\_QSPI, 2324
  - SPI\_USART, 2329
  - SPIModule, 2342
- tx
  - SAI\_cfg\_t, 2300
  - SSC\_cfg\_t, 2344
- tx\_event\_fifo\_watermark
  - mcanMODM7AE70::mcan\_config, 2160
- tx\_queue\_mode
  - mcanMODM7AE70::mcan\_config, 2160
- txBuffIfNull
  - [\\_lpspi\\_master\\_edma\\_handle](#), 1926
  - [\\_lpspi\\_master\\_handle](#), 1929
  - [\\_lpspi\\_slave\\_edma\\_handle](#), 1933
- txClkFreq
  - SAI\_cfg\_t, 2300
- txData
  - [\\_lpspi\\_master\\_edma\\_handle](#), 1927
  - [\\_lpspi\\_master\\_handle](#), 1929
  - [\\_lpspi\\_slave\\_edma\\_handle](#), 1933
  - [\\_lpspi\\_slave\\_handle](#), 1935
  - [\\_lpspi\\_transfer](#), 1936
- txRemainingByteCount
  - [\\_lpspi\\_master\\_edma\\_handle](#), 1927
  - [\\_lpspi\\_master\\_handle](#), 1929
  - [\\_lpspi\\_slave\\_edma\\_handle](#), 1933
  - [\\_lpspi\\_slave\\_handle](#), 1935
- type
  - [\\_flexcan\\_frame](#), 1915
  - [\\_flexcan\\_rx\\_mb\\_config](#), 1918
- TZRecords
  - Time, 1720
- tzsetchar
  - Time, 1720
- UDP, 965, 1740
  - RegisterEphemeralFifo, 1748
  - RegisterUDPFifo, 1750
  - RegisterUDPFifoVia, 1751
  - RegisterUDPFifoWithNotify, 1754
  - RegisterUDPFifoWithNotifyVia, 1755
  - udp\_data\_notify, 1747
  - UnregisterUDPFifo, 1757
- UDP C++ Packet Class, 971
- UDP Echo, 971
- UDP Error Codes, 1758
- UDP Notify Callback Function, 969
- UDP Packet API, 1759
- UDP Protocol, 277
- UDP Send/Receive, 973
- UDP Socket API, 1759
  - CreateRxTxUdpSocket, 1766
  - CreateRxUdpSocket, 1768
  - CreateTxUdpSocket, 1771
  - recvfrom, 1773
  - SendFragmentedUdpPacket, 1774
  - sendto, 1775
- UDP Sockets, 976
- UDP Sockets Via, 978
- UDP to Serial, 965
- udp\_data\_notify
  - UDP, 1747
- UdpCSum
  - UDPPKT, 2419
- UDPPacket, 2372
  - ~UDPPacket, 2386
  - AddData, 2389, 2390
  - AddDataByte, 2390
  - AddDataWord, 2391
  - blsIPv6, 2391
  - GetDataBuffer, 2392
  - GetDataSize, 2392
  - GetDestinationAddress, 2393
  - GetDestinationPort, 2393
  - GetDSCP, 2394
  - GetMacSource, 2395
  - GetPacketId, 2397
  - GetSourceAddress, 2397

- GetSourcePort, [2398](#)
- Receive, [2400](#)
- ResetData, [2402](#)
- Send, [2403](#)
- SendAndKeep, [2406](#)
- SendAndKeepVialfAddr, [2408](#)
- SendAndKeepVialInterfaceNum, [2409](#)
- SendVialfAddr, [2411](#)
- SendVialInterfaceNum, [2412](#)
- SetDataSize, [2415](#)
- SetDestinationPort, [2415](#)
- SetDSCP, [2416](#)
- SetSourcePort, [2416](#)
- UDPPacket, [2374](#), [2376](#), [2379](#), [2382](#), [2384](#)
- Validate, [2418](#)
- UDPPKT, [2419](#)
- UdpCsum, [2419](#)
- UNDEFINED
- JSON Lexer, [1331](#)
- unifyDataStream
  - SAI\_rtx\_cfg\_t, [2303](#)
- UniquelIdentifier, [2420](#)
  - GetBuffer, [2420](#)
  - GetUniquelIdentifier, [2420](#), [2421](#)
- UnRegisterFifo
  - mcanMODM7AE70::mcan\_module, [2165](#)
- UnregisterMulticastFifo4
  - Multicast, [1334](#)
- UnregisterMulticastFifo6
  - Multicast, [1334](#)
- UnregisterUDPFifo
  - UDP, [1757](#)
- Unsubscribe
  - MQTT::Client, [1976](#)
- UpdateCmd
  - WM8904, [2532](#)
- updateConsecutiveSeconds
  - IEEE1588Timer, [2097](#)
- UpdateFromStream
  - Signed Application Update, [1524](#)
- UpdateUserAuth
  - UserAuthManager, [2425](#)
- Upgrade 5441x Platforms to 3.x, [50](#)
- UsartModuleNumber, [1475](#)
- User Authorization Manager, [1777](#)
  - LoadAuthRecordsFn, [1777](#)
  - SaveAuthRecordsFn, [1777](#)
- UserAuthManager, [2421](#)
  - AddUserAuth, [2422](#)
  - CheckUserAuth, [2422](#), [2423](#)
  - CheckUserAuthLevel, [2423](#)
  - ClrUserAuthLevel, [2423](#)
  - Init, [2424](#)
  - RemoveUserAuth, [2424](#)
  - SetUserAuthLevel, [2424](#)
  - UpdateUserAuth, [2425](#)
  - UserExists, [2425](#)
- UserAuthRecord, [2426](#)
  - m\_authLevel, [2426](#)
- userData
  - \_edma\_handle, [1907](#)
  - \_flexcan\_handle, [1916](#)
  - \_lpspi\_master\_edma\_handle, [1927](#)
  - \_lpspi\_master\_handle, [1929](#)
  - \_lpspi\_slave\_edma\_handle, [1933](#)
  - \_lpspi\_slave\_handle, [1935](#)
- UserExists
  - UserAuthManager, [2425](#)
- Utilities, [1779](#)
  - \_exit, [1782](#)
  - \_fini, [1782](#)
  - AsciiToIp4, [1782](#)
  - AsciiToIp6, [1782](#)
  - AsciiToMac, [1782](#)
  - bufnstr, [1783](#)
  - convertBinaryToHexAscii, [1783](#)
  - convertHexAsciiToBinary, [1787](#)
  - DB\_TCPIP, [1781](#)
  - FdShowData, [1791](#)
  - fdShowMac, [1792](#)
  - FdShowRingData, [1792](#)
  - Get\_msec, [1792](#)
  - GetPreciseTime, [1793](#)
  - kill, [1793](#)
  - MacToId, [1793](#)
  - outbyte, [1793](#)
  - print, [1793](#)
  - putbytenum, [1794](#)
  - putnum, [1794](#)
  - ShowData, [1794](#)
  - ShowIP4, [1794](#)
  - ShowIP6, [1795](#)
  - ShowMac, [1795](#)
  - ShowRingData, [1795](#)
  - snShowIP4, [1796](#)
  - snShowIP6, [1796](#)
  - snShowMac, [1796](#)
  - sShowCounters, [1797](#)
  - ValidIPv4, [1797](#)
- Valid
  - JsonRef, [2154](#)
- valid
  - DelayObject, [2052](#)
  - ParsedURI, [2283](#)
- Validate
  - UDPPacket, [2418](#)
- ValidDhcpLease
  - DhcpObject, [2057](#)
- ValidIPv4
  - Utilities, [1797](#)
- VALUE\_SEPERATOR
  - JSON Lexer, [1332](#)
- vfdiprintf
  - File Descriptor Printf - fdprintf, [1241](#)
- vfdprintf
  - File Descriptor Printf - fdprintf, [1243](#)

- VLAN, [981](#)
- vsiprintf
  - NBString, [2247](#)
- WaitForActiveNetwork
  - Initialization - System Initialization Functions, [1330](#)
- WaitForDHCPInterface
  - DHCPv4 Client, [1177](#)
- WaitForPingReply
  - NB::Websocket, [2483](#)
- WaitForSocketFlush
  - TCP, [1706](#)
- waitTimeCfgCommands
  - \_FlexSPICfg, [1921](#)
- wakeupSrc
  - \_flexcan\_config, [1912](#)
- WalkNextHeader
  - IPv6FrameProcessingStruct, [2132](#)
- Watchdog Timer, [720](#)
- watchdog\_configuration
  - mcanMODM7AE70::mcan\_config, [2160](#)
- watchdog\_service\_function
  - HAL - Hardware Abstraction Layer, [1257](#)
- WAV File Audio Player, [670](#)
- Web Client, [1053](#), [1797](#)
  - DoActualClientRequest, [1806](#)
  - DoGet, [1806–1810](#)
  - DoGetEx, [1810](#), [1811](#), [1813–1815](#)
  - DoGetUpdate, [1816](#), [1817](#)
  - DoHttp, [1817–1821](#)
  - DoHttpEx, [1822–1826](#)
  - DoJsonHttp, [1827–1834](#)
  - DoJsonHttpFile, [1835](#), [1836](#), [1838](#)
  - DoJsonPost, [1839–1845](#)
  - DoJsonPostHttpFile, [1846–1848](#)
  - DoMultipartBoundary, [1849](#)
  - DoMultipartFinished, [1850](#), [1851](#)
  - DoMultipartItem, [1851](#)
  - DoMultipartStartHttp, [1852](#), [1853](#)
  - DoMultipartStartPost, [1854](#), [1855](#)
  - DoUrlEncodedFormHttp, [1856–1858](#)
  - DoUrlEncodedFormPost, [1859–1861](#)
  - PopulateAuthHeader, [1862](#)
  - SetHttpDiag, [1863](#)
  - StartConnection, [1863](#)
  - StartTransaction, [1864](#)
  - StartWebClient, [1865](#), [1866](#)
  - WriteHttpRequestHeaders, [1866](#)
- Web Client Error Codes, [1867](#)
- Web Server, [328](#), [984](#)
- Web Socket Connect, [1067](#)
- Web Socket Console, [1069](#)
- Web Socket Definitions, [1882](#)
- Web Socket DIP Switch, [1072](#)
- Web Socket Echo, [1075](#)
- Web Sockets, [344](#), [1067](#), [1868](#)
  - WSGetPingReplyTick, [1869](#)
  - WSPing, [1872](#)
  - WSUpgrade, [1874](#)
  - WSWaitForPingReply, [1879](#)
- WebClient Http Request Methods, [1058](#)
- WebConfigServlet, [2426](#)
- whichPcs
  - \_lpspi\_master\_config, [1924](#)
  - \_lpspi\_slave\_config, [1931](#)
- Width\_0
  - SEMC, [1427](#)
- Width\_10
  - SEMC, [1427](#)
- Width\_11
  - SEMC, [1427](#)
- Width\_12
  - SEMC, [1427](#)
- Width\_16
  - SEMC, [1426](#)
- Width\_2
  - SEMC, [1427](#)
- Width\_3
  - SEMC, [1427](#)
- Width\_4
  - SEMC, [1427](#)
- Width\_5
  - SEMC, [1427](#)
- Width\_6
  - SEMC, [1427](#)
- Width\_7
  - SEMC, [1427](#)
- Width\_8
  - SEMC, [1426](#), [1427](#)
- Width\_9
  - SEMC, [1427](#)
- Wifi, [1883](#)
  - InitAP\_SPI, [1884](#)
  - InitWifi\_Serial, [1885](#)
  - InitWifi\_SPI, [1885](#)
  - ScanAndShowNetworks, [1886](#)
  - ScanForNetworks, [1886](#)
  - SetWifiSPISpeed, [1886](#)
  - WifiInitScan\_Serial, [1887](#)
  - WifiInitScan\_SPI, [1887](#)
  - WifiInitScanAndShow\_Serial, [1887](#)
  - WifiInitScanAndShow\_SPI, [1888](#)
- Wifi Access Point, [1088](#)
- Wifi Client, [1091](#)
- Wifi Config AP, [1081](#)
- WiFi Examples, [1081](#)
- Wifi Firmware Update, [1087](#)
- Wifi Scan, [1093](#)
- wifi\_init, [2524](#)
- WifiInitScan\_Serial
  - Wifi, [1887](#)
- WifiInitScan\_SPI
  - Wifi, [1887](#)
- WifiInitScanAndShow\_Serial
  - Wifi, [1887](#)
- WifiInitScanAndShow\_SPI
  - Wifi, [1888](#)



- WifiInterface, [2524](#)
- Wire Address Scan, [513](#)
- Wire Interface, [1279](#)
- WireIntf, [2525](#)
  - available, [2525](#)
  - beginTransmission, [2526](#)
  - endTransmission, [2526](#)
  - flush, [2526](#)
  - read, [2526](#)
  - requestFrom, [2526](#)
  - write, [2527](#)
- WM8904, [2528](#)
  - GetMicGain, [2529](#)
  - GetVolume, [2529](#)
  - Init, [2530](#)
  - Mute, [2530](#)
  - MuteMic, [2530](#)
  - ReadReg, [2530](#)
  - ReadyReceiveBuffer, [2530](#)
  - SendCmd, [2531](#)
  - SendCmdList, [2531](#)
  - SetMicGain, [2531](#)
  - SetVolume, [2531](#)
  - TransmitBuffer, [2532](#)
  - UpdateCmd, [2532](#)
  - WM8904, [2529](#)
  - WriteReg, [2532](#)
- wordMask
  - SAI\_rtx\_cfg\_t, [2303](#)
- wordsPerFrame
  - SAI\_rtx\_cfg\_t, [2303](#)
  - SSC\_rtx\_cfg\_t, [2346](#)
- wr\_dat\_hold
  - SEMC\_cfg\_t, [2315](#)
- wr\_dat\_setup
  - SEMC\_cfg\_t, [2315](#)
- wr\_en\_hi
  - SEMC\_cfg\_t, [2315](#)
- wr\_en\_lo
  - SEMC\_cfg\_t, [2315](#)
- write
  - IOSYS - I/O System, [1295](#)
  - TwoWire, [2370](#), [2371](#)
  - WireIntf, [2527](#)
- writeall
  - IOSYS - I/O System, [1295](#)
- writeallsafestring
  - HTTP and HTML Functions, [1271](#)
- writeavail
  - IOSYS - I/O System, [1296](#)
- WriteData
  - fifo\_buffer\_storage, [2081](#)
  - NB::WebSocket, [2486](#)
  - ParsedJsonDataSet, [2281](#)
- WriteHtmlVariable
  - HTTP and HTML Functions, [1271](#)
- WriteHttpRequestHeaders
  - Web Client, [1866](#)
- WriteReg
  - WM8904, [2532](#)
- writeReg8
  - BBI2C, [1947](#)
  - I2C, [2090](#)
  - I2CDevice, [2093](#)
  - TwoWire, [2371](#)
- writeRegN
  - BBI2C, [1947](#)
  - I2C, [2090](#)
  - I2CDevice, [2093](#)
  - TwoWire, [2372](#)
- writeRegRemainingTimes
  - \_lpspi\_master\_edma\_handle, [1927](#)
  - \_lpspi\_master\_handle, [1930](#)
  - \_lpspi\_slave\_edma\_handle, [1933](#)
  - \_lpspi\_slave\_handle, [1935](#)
- writesafestring
  - HTTP and HTML Functions, [1272](#)
- writestring
  - IOSYS - I/O System, [1296](#)
- writeTcrInIsr
  - \_lpspi\_master\_handle, [1930](#)
- wrMode
  - EBI - External Bus Interface, [1403](#)
- ws\_cloption
  - NB::WebSocket, [2491](#)
- ws\_externalclose
  - NB::WebSocket, [2492](#)
- ws\_flush
  - NB::WebSocket, [2494](#)
- ws\_getoption
  - NB::WebSocket, [2496](#)
- ws\_read
  - NB::WebSocket, [2498](#)
- ws\_read\_notify
  - NB::WebSocket, [2501](#)
- ws\_readwto
  - NB::WebSocket, [2503](#)
- ws\_setoption
  - NB::WebSocket, [2506](#)
- ws\_write
  - NB::WebSocket, [2507](#)
- ws\_write\_notify
  - NB::WebSocket, [2511](#)
- WSGetPingReplyTick
  - Web Sockets, [1869](#)
- WSPing
  - Web Sockets, [1872](#)
- WSockStructs
  - NB::WebSocket, [2522](#)
- WSUpgrade
  - Web Sockets, [1874](#)
- WSWaitForPingReply
  - Web Sockets, [1879](#)
- ZeroWaitSelect
  - IOSYS - I/O System, [1297](#)