

# Microchip TCP/IP Stack Help

# Table of Contents

<b>Introduction</b>	<b>1</b>
Getting Help	1
Directory Structure	1
<b>SW License Agreement</b>	<b>3</b>
<b>Release Notes</b>	<b>6</b>
Stack Performance	54
Memory Usage	54
Peripheral Usage	54
<b>Silicon Solutions</b>	<b>56</b>
<b>Software</b>	<b>57</b>
TCP/IP Configuration Wizard	57
MPFS2 Utility	57
Building MPFS2 Images	58
Uploading Pre-built MPFS2 Images	58
Advanced MPFS2 Settings	59
MPFS2 Command Line Options	59
Hash Table Filter Entry Calculator	60
Microchip TCP/IP Discoverer	60
<b>Getting Started</b>	<b>62</b>
Hardware Setup	62
Daughter Boards	62
PICDEM.net 2	63
PIC18 Explorer	65
Explorer 16 and PIC32 Starter Kit	66
PIC24FJ256DA210 Dev Board	69
Programming and First Run	69
Configure your WiFi Access Point	70
Connecting to the Network	72

---

<b>Uploading Web Pages</b>	<b>72</b>
<b>Accessing the Demo Application</b>	<b>73</b>
<b>Configuring WiFi Security</b>	<b>74</b>
<b>Demo Information</b>	<b>77</b>
<b>Demo Compatibility Table</b>	<b>77</b>
<b>Available Demos</b>	<b>81</b>
Demo App	81
TCPIP Demo App Features by Hardware Platform	81
Demo Modules	82
Web Page Demos	82
E-mail (SMTP) Demo	91
Generic TCP Client	92
Generic TCP Server	95
Ping (ICMP) Demo	96
Network Management (SNMP) Server	98
UART-to-TCP Bridge	112
Zero Configuration (ZeroConf)	113
Internet Bootloader	114
Bootloader Design	114
Using the Bootloader	117
WebVend	120
Internet Radio	120
WiFi Console	121
Standalone Commands	121
iwconfig Commands	122
ifconfig Commands	123
iwpriv Commands	124
iperf Example	125
WiFi EZConfig	127
Demo App MDD	129
Google PowerMeter	129
Google PowerMeter EZConfig	130
Energy Monitoring	130
<b>Using the Stack</b>	<b>131</b>
<b>Stack Architecture</b>	<b>131</b>
<b>How the Stack Works</b>	<b>131</b>
Required Files	131

---

---

APP_CONFIG Structure	132
Main File	132
Initialization	132
Main Loop	132
Cooperative Multitasking	133
RTOS	135
<b>Configuring the Stack</b>	<b>136</b>
<b>Hardware Configuration</b>	<b>136</b>
Clock Frequency	136
External Storage	136
ENC28J60 Config	137
ENCX24J600 Config	138
PIC18F97J60 Config	140
PIC32MX7XX Config	140
<b>Address</b>	<b>141</b>
MAC Address	141
IP Address	142
<b>Protocol Configuration</b>	<b>143</b>
Protocol Macros and Files	144
Additional Features	145
Sockets	146
Memory Allocation	146
Socket Types	146
Initialization Structure	147
UDP Sockets	148
BSD Sockets	148
<b>Stack API</b>	<b>149</b>
<b>Announce</b>	<b>149</b>
Stack Members	150
AnnounceIP Function	150
DiscoveryTask Function	150
<b>ARP</b>	<b>151</b>
Public Members	151
ARPResolve Function	152
ARPIsResolved Function	153
ARPDeRegisterCallbacks Function	153
ARPRegisterCallbacks Function	154
ARPSendPkt Function	154

---

---

arp_app_callbacks Structure	155
ARP_REQ Macro	155
ARP_RESP Macro	155
MAX_REG_APPS Macro	155
Stack Members	156
ARPInit Function	156
ARPPProcess Function	156
Internal Members	157
ARPPut Function	158
SwapARPPacket Function	158
ARP_OPERATION_REQ Macro	159
ARP_OPERATION_RESP Macro	159
HW_ETHERNET Macro	159
ARP_IP Macro	159
Cache Variable	159
reg_apps Variable	160
Types	160
ARP_PACKET Structure	160
<b>BSD Sockets</b>	<b>161</b>
Public Members	162
accept Function	163
AF_INET Macro	164
bind Function	164
BSDSocket Structure	164
closesocket Function	165
connect Function	165
gethostname Function	166
in_addr Structure	167
INADDR_ANY Macro	167
INVALID_TCP_PORT Macro	168
IP_ADDR_ANY Macro	168
IPPROTO_IP Macro	168
IPPROTO_TCP Macro	168
IPPROTO_UDP Macro	168
listen Function	169
recv Function	169
recvfrom Function	170
send Function	171
sendto Function	171
SOCK_DGRAM Macro	172
SOCK_STREAM Macro	172
sockaddr Structure	172

---

---

SOCKADDR Type	173
sockaddr_in Structure	173
SOCKADDR_IN Type	173
socket Function	174
SOCKET Type	174
SOCKET_CNXXN_IN_PROGRESS Macro	174
SOCKET_DISCONNECTED Macro	175
SOCKET_ERROR Macro	175
Stack Members	175
BerkeleySocketInit Function	175
Internal Members	176
BSD_SCK_STATE Enumeration	176
BSDSocketArray Variable	177
gAutoPortNumber Variable	177
HandlePossibleTCPDisconnection Function	177
<b>DNS</b>	<b>178</b>
Public Members	178
DNSBeginUsage Function	179
DNSEndUsage Function	179
DNSResolve Function	180
DNSResolveROM Function	180
DNSIsResolved Function	181
DNS_TYPE_A Macro	181
DNS_TYPE_MX Macro	182
Internal Members	182
DNSPutString Function	183
DNSPutROMString Function	183
DNS_PORT Macro	184
DNS_TIMEOUT Macro	184
DNSHostName Variable	184
DNSHostNameROM Variable	184
Flags Variable	185
RecordType Variable	185
ResolvedInfo Variable	185
smDNS Variable	185
DNS_HEADER Structure	186
DNSDiscardName Function	186
<b>Dynamic DNS Client</b>	<b>187</b>
Public Members	187
DDNS_POINTERS Structure	188
DDNS_SERVICES Enumeration	189

---

---

DDNS_STATUS Enumeration	189
DDNSClient Variable	190
DDNSForceUpdate Function	190
DDNSGetLastIP Function	191
DDNSGetLastStatus Function	191
DDNSSetService Function	191
Stack Members	192
DDNSInit Function	192
DDNSTask Function	192
Internal Members	193
bForceUpdate Variable	194
ddnsServiceHosts Variable	194
ddnsServicePorts Variable	194
dwUpdateAt Variable	194
lastKnownIP Variable	194
lastStatus Variable	195
_checkIpSrvrResponse Variable	195
_updateIpSrvrResponse Variable	195
DDNS_CHECKIP_SERVER Macro	195
DDNS_DEFAULT_PORT Macro	196
<b>Hashes</b>	<b>196</b>
Public Members	196
HashAddData Function	197
HashAddROMData Function	197
MD5Calculate Function	198
MD5Initialize Function	199
SHA1Calculate Function	199
SHA1Initialize Function	199
HASH_SUM Structure	200
Stack Members	201
MD5AddROMData Function	201
SHA1AddROMData Function	202
SHA1AddData Function	202
MD5AddData Function	203
Internal Members	203
_MD5_k Variable	204
_MD5_r Variable	204
lastBlock Variable	204
HASH_TYPE Enumeration	204
SHA1HashBlock Function	205
MD5HashBlock Function	206

---

<b>Helpers</b>	<b>206</b>
Public Members	207
Base64Decode Function	208
Base64Encode Function	208
btohexa_high Function	209
btohexa_low Function	209
CalcIPBufferChecksum Function	210
CalcIPChecksum Function	210
ExtractURLFields Function	211
FormatNetBIOSName Function	214
GenerateRandomDWORD Function	214
hexatob Function	215
leftRotateDWORD Function	215
leftRotateDWORD Macro	216
Replace Function	216
ROMStringToIPAddress Function	217
ROMStringToIPAddress Macro	218
stricmppgm2ram Function	218
StringToIPAddress Function	218
strupr Function	219
strnchr Function	219
swapl Function	220
swaps Function	220
uitoa Function	221
ultoa Function	221
UnencodeURL Function	222
Functions	222
LFSRRand Function	222
LFSRSeedRand Function	223
Variables	224
dwLFSRRandSeed Variable	224
<b>HTTP2 Server</b>	<b>224</b>
Features	225
Dynamic Variables	225
Form Processing	227
Authentication	230
Cookies	232
Compression	232
Public Members	233
curHTTP Variable	234
HTTP_CONN Structure	234

---

HTTP_IO_RESULT Enumeration	235
HTTP_READ_STATUS Enumeration	235
HTTPCheckAuth Function	235
HTTPExecuteGet Function	236
HTTPExecutePost Function	237
HTTPGetArg Function	238
HTTPGetROMArg Function	239
HTTPNeedsAuth Function	239
HTTPPrint_varname Function	240
HTTPReadPostName Function	241
HTTPReadPostPair Macro	242
HTTPReadPostValue Function	242
HTTPURLDecode Function	243
sktHTTP Macro	244
Stack Members	244
HTTPInit Function	244
HTTPServer Function	245
Internal Members	245
curHTTPID Variable	246
HTTP_CACHE_LEN Macro	247
HTTP_FILE_TYPE Enumeration	247
HTTP_MAX_DATA_LEN Macro	248
HTTP_MAX_HEADER_LEN Macro	248
HTTP_MIN_CALLBACK_FREE Macro	248
HTTP_PORT Macro	248
HTTP_STATUS Enumeration	248
HTTP_STUB Structure	249
HTTP_TIMEOUT Macro	250
httpContentTypes Variable	250
httpFileExtensions Variable	250
HTTPHeaderParseAuthorization Function	250
HTTPHeaderParseContentLength Function	251
HTTPHeaderParseCookie Function	251
HTTPHeaderParseLookup Function	252
HTTPIncFile Function	252
HTTPLoadConn Function	253
HTTPMPFSUpload Function	253
HTTPProcess Function	254
HTTPReadTo Function	254
HTTPRequestHeaders Variable	255
HTTPResponseHeaders Variable	255
HTTPS_PORT Macro	255

---

---

HTTPSendFile Function	256
httpStubs Variable	256
SM_HTTP2 Enumeration	256
smHTTP Macro	257
RESERVED_HTTP_MEMORY Macro	257
<b>ICMP</b>	<b>257</b>
Public Members	258
ICMPBeginUsage Function	258
ICMPSendPing Function	259
ICMPSendPingToHost Function	259
ICMPSendPingToHostROM Function	260
ICMPGetReply Function	260
ICMPEndUsage Function	261
ICMPSendPingToHostROM Macro	261
Internal Members	262
ICMPProcess Function	262
ICMPFlags Variable	263
ICMP_PACKET Structure	263
ICMPState Variable	263
ICMP_TIMEOUT Macro	264
ICMPTimer Variable	264
StaticVars Variable	264
wICMPSequenceNumber Variable	265
<b>MPFS2</b>	<b>265</b>
Public Members	266
MPFS_HANDLE Type	267
MPFS_INVALID Macro	267
MPFS_INVALID_HANDLE Macro	267
MPFS_SEEK_MODE Enumeration	267
MPFSClose Function	268
MPFSFormat Function	268
MPFSGet Function	269
MPFSGetArray Function	269
MPFSGetBytesRem Function	270
MPFSGetEndAddr Function	270
MPFSGetFilename Function	271
MPFSGetFlags Function	271
MPFSGetID Function	272
MPFSGetLong Function	272
MPFSGetMicrotime Function	273
MPFSGetPosition Function	273

---

---

MPFSGetSize Function	273
MPFSGetStartAddr Function	274
MPFSGetTimestamp Function	274
MPFSOpen Function	275
MPFSOpenID Function	275
MPFSOpenROM Function	276
MPFSPutArray Function	276
MPFSSeek Function	277
MPFSPutEnd Function	277
Stack Members	278
MPFSInit Function	278
Internal Members	278
isMPFSLocked Variable	279
lastRead Variable	280
MAX_FILE_NAME_LEN Macro	280
MPFS_PTR Type	280
MPFS_STUB Structure	280
MPFS_WRITE_PAGE_SIZE Macro	281
MPFS2_FLAG_HASINDEX Macro	281
MPFS2_FLAG_ISZIPPED Macro	281
MPFSStubs Variable	281
MPFSTell Macro	282
ReadProgramMemory Function	282
_LoadFATRecord Function	282
_Validate Function	283
MPFS_FAT_RECORD Structure	283
fatCache Variable	283
fatCacheID Variable	284
numFiles Variable	284
MPFS_INVALID_FAT Macro	284
<b>NBNS</b>	<b>284</b>
Stack Members	285
NBNSGetName Function	285
NBNSPutName Function	286
NBNSTask Function	286
NBNS_HEADER Structure	287
NBNS_PORT Macro	287
<b>Performance Tests</b>	<b>287</b>
Stack Members	287
TCPPerformanceTask Function	288
UDPPerformanceTask Function	288

---

Internal Members	289
TCPRXPerformanceTask Function	289
TCPTXPerformanceTask Function	290
PERFORMANCE_PORT Macro	290
RX_PERFORMANCE_PORT Macro	290
TX_PERFORMANCE_PORT Macro	291
<b>SMTP Client</b>	<b>291</b>
Examples	291
Short Message	291
Long Message	292
Public Members	294
SMTP_CONNECT_ERROR Macro	295
SMTP_POINTERS Structure	295
SMTP_RESOLVE_ERROR Macro	297
SMTP_SUCCESS Macro	297
SMTPBeginUsage Function	297
SMTPClient Variable	298
SMTPEndUsage Function	298
SMTPFlush Function	298
SMTPIsBusy Function	299
SMTPIsPutReady Function	299
SMTPPut Function	300
SMTPPutArray Function	300
SMTPPutDone Function	301
SMTPPutROMArray Function	301
SMTPPutROMString Function	302
SMTPPutString Function	302
SMTPSendMail Function	303
Stack Members	303
SMTPTask Function	304
Internal Members	304
CRPeriod Variable	305
FindEmailAddress Function	305
FindROMEmailAddress Function	306
MySocket Variable	306
PutHeadersState Variable	306
ResponseCode Variable	307
RXParserState Variable	307
SMTP_PORT Macro	308
SMTP_SERVER_REPLY_TIMEOUT Macro	308
SMTPFlags Variable	308
SMTPServer Variable	308

---

SMTPState Variable	309
TransportState Variable	310
<b>Reboot</b>	<b>311</b>
Stack Members	311
RebootTask Function	311
REBOOT_PORT Macro	312
REBOOT_SAME_SUBNET_ONLY Macro	312
<b>SNMP</b>	<b>312</b>
Public Members	314
GENERIC_TRAP_NOTIFICATION_TYPE Enumeration	316
VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE Enumeration	316
SNMP_ACTION Enumeration	316
COMMUNITY_TYPE Enumeration	317
SNMP_VAL Union	317
TRAP_INFO Structure	318
gSendTrapFlag Variable	318
gSetTrapSendFlag Variable	318
gGenericTrapNotification Variable	319
gSpecificTrapNotification Variable	319
gOIDCorrespondingSnmplibID Variable	319
SNMPSendTrap Function	319
SNMPValidateCommunity Function	320
SNMPNotify Function	320
SNMPSetVar Function	321
SNMPGetVar Function	322
SNMPIsNotifyReady Function	322
SNMPNotifyPrepare Function	323
SNMPGetNextIndex Function	324
SNMP_ID Type	324
SNMP_INDEX Type	324
SNMP_COMMUNITY_MAX_LEN Macro	325
OID_MAX_LEN Macro	325
SNMP_START_OF_VAR Macro	325
SNMP_END_OF_VAR Macro	325
SNMP_INDEX_INVALID Macro	326
TRAP_TABLE_SIZE Macro	326
TRAP_COMMUNITY_MAX_LEN Macro	326
NOTIFY_COMMUNITY_LEN Macro	326
Internal Members	327
_SNMPDuplexInit Function	330
_SNMPGet Function	330

---

---

_SNMPGetTxOffset Macro	330
_SNMPPut Function	330
_SNMPSetTxOffset Macro	331
AGENT_NOTIFY_PORT Macro	331
appendZeroToOID Variable	331
ASN_INT Macro	331
ASN_NULL Macro	332
ASN_OID Macro	332
DATA_TYPE Enumeration	332
DATA_TYPE_INFO Structure	333
DATA_TYPE_TABLE_SIZE Macro	333
dataTypeTable Variable	333
FindOIDsInRequest Function	334
GET_BULK_REQUEST Macro	334
GET_NEXT_REQUEST Macro	334
GET_REQUEST Macro	334
GET_RESPONSE Macro	334
hMPFS Variable	335
INDEX_INFO Union	335
IS_AGENT_PDU Macro	335
IS_ASN_INT Macro	336
IS_ASN_NULL Macro	336
IS_GET_NEXT_REQUEST Macro	336
IS_GET_REQUEST Macro	336
IS_GET_RESPONSE Macro	336
IS_OCTET_STRING Macro	337
IS_OID Macro	337
GetDataTypeInfo Function	337
IS_SET_REQUEST Macro	337
IS_STRUCTURE Macro	338
IS_TRAP Macro	338
IsASNNull Function	338
MIB_INFO Union	338
OCTET_STRING Macro	339
OID_INFO Structure	339
PDU_INFO Structure	340
reqVarErrStatus Structure	340
SET_REQUEST Macro	341
SetErrorStatus Function	341
SNMP_AGENT_PORT Macro	341
SNMP_BIB_FILE_NAME Macro	342
SNMP_COUNTER32 Macro	342

---

---

SNMP_ERR_STATUS Enumeration	342
SNMP_GAUGE32 Macro	343
SNMP_IP_ADDR Macro	343
SNMP_NMS_PORT Macro	344
SNMP_NOTIFY_INFO Structure	344
SNMP_NSAP_ADDR Macro	344
IsValidLength Function	345
SNMP_OPAQUE Macro	345
SNMP_STATUS Union	345
SNMP_TIME_TICKS Macro	345
SNMP_V1 Macro	346
SNMP_V2C Macro	346
SNMPAgentSocket Variable	346
SNMPNotifyInfo Variable	346
snmpReqVarErrStatus Variable	347
SNMPRxOffset Variable	347
SNMPStatus Variable	347
SNMPTxOffset Variable	347
STRUCTURE Macro	347
TRAP Macro	348
trapInfo Variable	348
GetDataTypeInfo Function	348
GetNextLeaf Function	348
GetOIDStringByAddr Function	349
GetOIDStringByID Function	349
IsValidCommunity Function	349
IsValidInt Function	349
IsValidLength Function	350
IsValidOID Function	350
IsValidPDU Function	350
IsValidStructure Function	350
OIDLookup Function	351
ProcessGetSetHeader Function	351
ProcessHeader Function	352
ProcessSetVar Function	352
ProcessVariables Function	352
ReadMIBRecord Function	353
SNMPCheckIfPvtMibObjRequested Function	353
Stack Members	354
SNMPInit Function	354
SNMPTask Function	354
Functions	355

---

getSnmpV2GenTrapOid Function	356
ProcessGetBulkVar Function	356
ProcessGetNextVar Function	357
ProcessGetVar Function	357
ProcessSnmpv3MsgData Function	357
SNMPGetExactIndex Function	357
SNMPIdRecrdValidation Function	358
SNMPIsValidSetLen Function	359
Snmpv3AESDecryptRxedScopedPdu Function	359
Snmpv3BufferPut Function	359
Snmpv3FormulateEngineID Function	360
Snmpv3GetAuthEngineTime Function	360
Snmpv3GetBufferData Function	360
Snmpv3InitializeUserDataBase Function	360
Snmpv3MsgProcessingModelProcessPDU Function	361
Snmpv3Notify Function	361
Snmpv3ScopedPduProcessing Function	361
Snmpv3TrapScopedpdu Function	361
Snmpv3UserSecurityModelProcessPDU Function	362
Snmpv3UsmAesEncryptDecryptInitVector Function	362
Snmpv3UsmOutMsgAuthenticationParam Function	362
Snmpv3ValidateEngineId Function	362
Snmpv3ValidateSecNameAndSecLvl Function	363
Snmpv3ValidateSecurityName Function	363
Types	363
INOUT_SNMP_PDU Enumeration	363
SNMPNONMIBRECDINFO Structure	364
SNMPV3MSGDATA Structure	364
Variables	364
getZeroInstance Variable	365
gSnmpNonMibRecInfo Variable	365
gSNMPv3ScopedPduDataPos Variable	365
gSNMPv3ScopedPduRequestBuf Variable	365
gSNMPv3ScopedPduResponseBuf Variable	366
msgSecrtyParamLenOffset Variable	366
Macros	366
IS_SNMPV3_AUTH_STRUCTURE Macro	366
REPORT_RESPONSE Macro	367
SNMP_MAX_MSG_SIZE Macro	367
SNMP_MAX_NON_REC_ID_OID Macro	367
SNMP_V3 Macro	367
<b>SNTP Client</b>	<b>368</b>

Public Members	368
SNTPGetUTCSeconds Function	368
Stack Members	369
SNTPClient Function	369
Internal Members	369
NTP_PACKET Structure	370
dwLastUpdateTick Variable	371
dwSNTPSeconds Variable	371
NTP_EPOCH Macro	372
NTP_FAST_QUERY_INTERVAL Macro	372
NTP_QUERY_INTERVAL Macro	372
NTP_REPLY_TIMEOUT Macro	372
NTP_SERVER Macro	372
NTP_SERVER_PORT Macro	373
<b>SSL</b>	<b>373</b>
Generating Server Certificates	374
Public Members	377
SSL_INVALID_ID Macro	377
TCPAddSSLListener Function	378
TCPSSLIsHandshaking Function	378
TCPStartSSLClient Function	379
TCPisSSL Function	379
SSLStartSession Function	380
SSL_SUPPLEMENTARY_DATA_TYPES Enumeration	380
SSL_PKEY_INFO Structure	380
Stack Members	381
SSL_STATE Enumeration	382
SSLInit Function	382
SSLPeriodic Function	382
TCPRequestSSLMessage Function	383
TCPSSLGetPendingTxSize Function	383
TCPSSLHandleIncoming Function	384
TCPSSLHandshakeComplete Function	384
TCPSSLInPlaceMACEncrypt Function	385
TCPSSLPutRecordHeader Function	385
TCPStartSSLServer Function	386
SSL_MIN_SESSION_LIFETIME Macro	386
SSL_RSA_LIFETIME_EXTENSION Macro	387
Internal Members	387
CalculateFinishedHash Function	392
GenerateHashRounds Function	393
GenerateSessionKeys Function	393

HSEnd Function	394
HSGet Function	394
HSGetArray Function	395
HSGetWord Function	395
HSPut Function	396
HSPutArray Function	396
HSPutROMArray Function	397
HSPutWord Function	397
HSStart Function	398
isBufferUsed Variable	398
isHashUsed Variable	399
isStubUsed Variable	399
masks Variable	399
ptrHS Variable	399
RESERVED_SSL_MEMORY Macro	399
LoadOffChip Function	400
SaveOffChip Function	400
SM_SSL_RX_SERVER_HELLO Enumeration	401
SSL_ALERT Macro	401
SSL_ALERT_LEVEL Enumeration	401
SSL_APPLICATION Macro	402
SSL_BASE_BUFFER_ADDR Macro	402
SSL_BASE_HASH_ADDR Macro	402
SSL_BASE_KEYS_ADDR Macro	402
SSL_BASE_SESSION_ADDR Macro	403
SSL_BASE_STUB_ADDR Macro	403
SSL_BUFFER Union	403
SSL_BUFFER_SIZE Macro	403
SSL_BUFFER_SPACE Macro	404
SSL_CERT Variable	404
SSL_CERT_LEN Variable	404
SSL_CHANGE_CIPHER_SPEC Macro	404
SSL_HANDSHAKE Macro	405
SSL_HASH_SIZE Macro	405
SSL_HASH_SPACE Macro	405
SSL_KEYS Structure	405
SSL_KEYS_SIZE Macro	406
SSL_KEYS_SPACE Macro	406
SSL_MESSAGES Enumeration	406
SSL_RSA_EXPORT_WITH_ARCFOUR_40_MD5 Macro	407
SSL_RSA_WITH_ARCFOUR_128_MD5 Macro	407
SSL_SESSION Structure	408

---

SSL_SESSION_SIZE Macro	408
SSL_SESSION_SPACE Macro	408
SSL_SESSION_STUB Structure	409
SSL_SESSION_TYPE Enumeration	409
SSL_STUB Structure	409
SSL_STUB_SIZE Macro	411
SSL_STUB_SPACE Macro	411
SSL_VERSION Macro	411
SSL_VERSION_HI Macro	411
SSL_VERSION_LO Macro	411
SSLBufferAlloc Function	412
SSLBufferFree Function	412
sslBufferID Variable	413
SSLBufferSync Function	413
SSLFinishPartialRecord Macro	413
SSLFlushPartialRecord Macro	414
sslHash Variable	414
SSLHashAlloc Function	414
SSLHashFree Function	415
sslHashID Variable	415
SSLHashSync Function	415
sslKeys Variable	416
sslKeysID Variable	416
SSLKeysSync Function	416
SSLMACAdd Function	417
SSLMACBegin Function	417
SSLMACCalc Function	417
SSLRSAOperation Function	418
sslRSAStubID Variable	418
SSLRxAlert Function	418
SSLRxAntiqueClientHello Function	419
SSLRxCCS Function	419
SSLRxClientHello Function	420
SSLRxClientKeyExchange Function	421
SSLRxFinished Function	421
SSLRxHandshake Function	422
SSLRxRecord Function	422
SSLRxServerCertificate Function	423
SSLRxServerHello Function	423
sslSession Variable	424
sslSessionID Variable	424
SSLSessionMatchID Function	424

---

SSLSessionMatchIP Function	425
SSLSessionNew Function	425
sslSessionStubs Variable	426
SSLSessionSync Function	426
SSLSessionUpdated Macro	427
sslSessionUpdated Variable	427
SSLStartPartialRecord Function	427
sslStub Variable	428
SSLStubAlloc Function	428
SSLStubFree Function	428
sslStubID Variable	429
SSLStubSync Function	429
SSLTerminate Function	430
SSLTxCCSFin Function	430
SSLTxClientHello Function	431
SSLTxClientKeyExchange Function	431
SSLTxMessage Function	432
SSLTxRecord Function	432
SSLTxServerCertificate Function	433
SSLTxServerHello Function	433
SSLTxServerHelloDone Function	434
<b>TCP</b>	<b>434</b>
Public Members	436
INVALID_SOCKET Macro	437
UNKNOWN_SOCKET Macro	438
TCP_ADJUST_GIVE_REST_TO_RX Macro	438
TCP_ADJUST_GIVE_REST_TO_TX Macro	438
TCP_ADJUST_PRESERVE_RX Macro	438
TCP_ADJUST_PRESERVE_TX Macro	439
TCP_OPEN_IP_ADDRESS Macro	439
TCP_OPEN_NODE_INFO Macro	439
TCP_OPEN_RAM_HOST Macro	439
TCP_OPEN_ROM_HOST Macro	440
TCP_OPEN_SERVER Macro	440
TCPAdjustFIFOSize Function	440
TCPConnect Macro	441
TCPClose Function	441
TCPDiscard Function	442
TCPDisconnect Function	442
TCPFind Macro	443
TCPFindArray Macro	443
TCPFindArrayEx Function	443

---

TCPFindEx Function	444
TCPFindROMArray Macro	445
TCPFindROMArrayEx Function	445
TCPFlush Function	446
TCPGet Function	446
TCPGetArray Function	447
TCPGetRemoteInfo Function	447
TCPGetRxFIFOFree Function	448
TCPGetRxFIFOFull Macro	448
TCPGetTxFIFOFree Macro	449
TCPGetTxFIFOFull Function	449
TCPIsConnected Function	449
TCPIsGetReady Function	450
TCPIsPutReady Function	450
TCPListen Macro	451
TCPOpen Function	451
TCPPeek Function	453
TCPPeekArray Function	453
TCPPut Function	454
TCPPutArray Function	454
TCPPutROMArray Function	455
TCPPutROMString Function	455
TCPPutString Function	456
TCPRAMCopy Function	456
TCPRAMCopyROM Function	457
TCPWasReset Function	458
Stack Members	458
SOCKET_INFO Structure	459
TCB Structure	460
TCB_STUB Structure	461
TCP_SOCKET Type	462
TCP_STATE Enumeration	462
TCPInit Function	463
TCPProcess Function	464
TCPTick Function	464
TCPSSLDecryptMAC Function	465
TCPStartSSLClientEx Function	465
Internal Members	466
ACK Macro	468
CloseSocket Function	468
FIN Macro	468
FindMatchingSocket Function	469

---

---

HandleTCPSeg Function	469
hCurrentTCP Variable	470
LOCAL_PORT_END_NUMBER Macro	470
LOCAL_PORT_START_NUMBER Macro	470
MyTCB Variable	470
MyTCBStub Variable	471
PSH Macro	471
RST Macro	471
SendTCP Function	471
SENDTCP_KEEP_ALIVE Macro	472
SENDTCP_RESET_TIMERS Macro	472
SwapTCPHeader Function	472
SYN Macro	473
SyncTCB Function	473
SyncTCBStub Macro	473
SYNQueue Variable	473
TCBStubs Variable	473
TCP_AUTO_TRANSMIT_TIMEOUT_VAL Macro	474
TCP_WINDOW_UPDATE_TIMEOUT_VAL Macro	474
TCP_CLOSE_WAIT_TIMEOUT Macro	474
TCP_DELAYED_ACK_TIMEOUT Macro	474
TCP_FIN_WAIT_2_TIMEOUT Macro	475
TCP_HEADER Structure	475
TCP_KEEP_ALIVE_TIMEOUT Macro	476
TCP_MAX_RETRIES Macro	476
TCP_MAX_SEG_SIZE_RX Macro	476
TCP_MAX_SEG_SIZE_TX Macro	477
TCP_MAX_SYN_RETRIES Macro	477
TCP_MAX_UNACKED_KEEP_ALIVES Macro	477
TCP_OPTIMIZE_FOR_SIZE Macro	477
TCP_OPTIONS Structure	478
TCP_OPTIONS_END_OF_LIST Macro	478
TCP_OPTIONS_MAX_SEG_SIZE Macro	478
TCP_OPTIONS_NO_OP Macro	478
TCP_SOCKET_COUNT Macro	479
TCP_START_TIMEOUT_VAL Macro	479
TCP_SYN_QUEUE Structure	479
TCP_SYN_QUEUE_MAX_ENTRIES Macro	480
TCP_SYN_QUEUE_TIMEOUT Macro	480
URG Macro	480
Variables	480
NextPort Variable	480

---

<b>TFTP</b>	<b>485</b>
Public Members	486
TFTPClose Macro	488
TFTPCloseFile Function	488
TFTPGet Function	489
TFTPGetError Macro	489
TFTPisFileClosed Function	490
TFTPisFileOpened Function	490
TFTPisFileOpenReady Macro	491
TFTPisGetReady Function	491
TFTPisOpened Function	492
TFTPisPutReady Function	492
TFTPOpen Function	493
TFTPOpenFile Function	494
TFTPOpenROMFile Function	494
TFTPput Function	495
TFTP_ACCESS_ERROR Enumeration	495
TFTP_FILE_MODE Enumeration	495
TFTP_RESULT Enumeration	496
TFTPGetUploadStatus Function	496
TFTPUploadFragmentedRAMFileToHost Function	497
TFTPUploadRAMFileToHost Function	498
TFTP_CHUNK_DESCRIPTOR Structure	498
TFTP_UPLOAD_COMPLETE Macro	499
TFTP_UPLOAD_CONNECT Macro	499
TFTP_UPLOAD_CONNECT_TIMEOUT Macro	499
TFTP_UPLOAD_GET_DNS Macro	499
TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT Macro	500
TFTP_UPLOAD_RESOLVE_HOST Macro	500
TFTP_UPLOAD_SEND_DATA Macro	500
TFTP_UPLOAD_SEND_FILENAME Macro	500
TFTP_UPLOAD_SERVER_ERROR Macro	500
TFTP_UPLOAD_WAIT_FOR_CLOSURE Macro	501
Stack Members	501
TFTP_ARP_TIMEOUT_VAL Macro	501
TFTP_GET_TIMEOUT_VAL Macro	502
TFTP_MAX_RETRIES Macro	502
Internal Members	502
MutexVar Variable	503
TFTP_BLOCK_SIZE Macro	504
TFTP_BLOCK_SIZE_MSB Macro	504
TFTP_CLIENT_PORT Macro	504

---

TFTP_OPCODE Enumeration	504
TFTP_SERVER_PORT Macro	505
TFTP_STATE Enumeration	505
_tftpError Variable	505
_tftpFlags Variable	505
_tftpRetries Variable	506
_TFTPSendAck Function	506
_TFTPSendFileName Function	506
_TFTPSendROMFileName Function	507
_tftpSocket Variable	507
_tftpStartTick Variable	507
_tftpState Variable	507
smUpload Variable	507
uploadChunkDescriptor Variable	508
uploadChunkDescriptorForRetransmit Variable	508
vUploadFilename Variable	508
vUploadRemoteHost Variable	508
wUploadChunkOffset Variable	509
wUploadChunkOffsetForRetransmit Variable	509
<b>Tick</b>	<b>509</b>
Public Members	510
TICK Type	510
TICK_HOUR Macro	511
TICK_MINUTE Macro	511
TICK_SECOND Macro	511
TickConvertToMilliseconds Function	511
TickGet Function	512
TickGetDiv256 Function	512
TickGetDiv64K Function	513
Stack Functions	513
TickInit Function	513
TickUpdate Function	514
Internal Members	514
dwInternalTicks Variable	514
GetTickCopy Function	515
TICKS_PER_SECOND Macro	515
vTickReading Variable	515
<b>UDP</b>	<b>516</b>
Public Members	517
INVALID_UDP_PORT Macro	518
INVALID_UDP_SOCKET Macro	518

---

---

UDP_SOCKET Type	518
UDPOpenEx Function	519
UDPOpen Macro	520
UDPClose Function	521
UDPDiscard Function	521
UDPFflush Function	522
UDPGet Function	522
UDPGetArray Function	523
UDPisGetReady Function	523
UDPisPutReady Function	524
UDPPut Function	524
UDPPutArray Function	525
UDPPutROMArray Function	525
UDPPutROMString Function	526
UDPPutString Function	526
UDPSetRxBuffer Function	527
UDPSetTxBuffer Function	527
UDPisOpened Function	528
UDP_OPEN_IP_ADDRESS Macro	528
UDP_OPEN_NODE_INFO Macro	528
UDP_OPEN_RAM_HOST Macro	529
UDP_OPEN_ROM_HOST Macro	529
UDP_OPEN_SERVER Macro	529
Stack Members	529
UDPInit Function	530
UDPProcess Function	530
UDPTask Function	531
Internal Members	531
activeUDPSocket Variable	532
FindMatchingSocket Function	532
LastPutSocket Variable	533
LOCAL_UDP_PORT_END_NUMBER Macro	533
LOCAL_UDP_PORT_START_NUMBER Macro	533
SocketWithRxData Variable	533
UDP_HEADER Structure	534
UDP_PORT Type	534
UDP_SOCKET_INFO Structure	534
UDPRxCount Variable	535
UDPSocketInfo Variable	535
UDPTxCount Variable	535
wGetOffset Variable	535
wPutOffset Variable	536

---

Types	536
UDP_STATE Enumeration	536
<b>Wi-Fi API</b>	<b>537</b>
<b>Wi-Fi Connection Profile</b>	<b>540</b>
Connection Profile Public Members	540
WF_CPCreate Function	541
WF_CPDelete Function	542
WF_CPGetAdHocBehavior Function	542
WF_CPGetBssid Function	543
WF_CPGetDefaultWepKeyIndex Function	543
WF_CPGetElements Function	544
WF_CPGetIds Function	544
WF_CPGetNetworkType Function	545
WF_CPGetSecurity Function	546
WF_CPGetSsid Function	547
WF_CPSetAdHocBehavior Function	547
WF_CPSetBssid Function	548
WF_CPSetDefaultWepKeyIndex Function	548
WF_CPSetElements Function	549
WF_CPSetNetworkType Function	549
WF_CPSetSecurity Function	550
WF_CPSetSsid Function	551
WFCPElementsStruct Structure	551
Connection Profile Internal Members	552
LowLevel_CPGetElement Function	553
LowLevel_CPSetElement Function	553
<b>Wi-Fi Connection Algorithm</b>	<b>554</b>
Connection Algorithm Public Members	554
WF_CAGetBeaconTimeout Function	555
WF_CAGetBeaconTimeoutAction Function	556
WF_CAGetChannelList Function	557
WF_CAGetConnectionProfileList Function	557
WF_CAGetDeathAction Function	558
WF_CAGetElements Function	558
WF_CAGetEventNotificationAction Function	559
WF_CAGetListenInterval Function	559
WF_CAGetListRetryCount Function	560
WF_CAGetMaxChannelTime Function	560
WF_CAGetMinChannelTime Function	561
WF_CAGetProbeDelay Function	561

WF_CAGetRssi Function	562
WF_CAGetScanCount Function	562
WF_CAGetScanType Function	563
WF_CASetBeaconTimeout Function	563
WF_CASetBeaconTimeoutAction Function	564
WF_CASetChannelList Function	565
WF_CASetConnectionProfileList Function	565
WF_CASetDeauthAction Function	566
WF_CASetElements Function	566
WF_CASetEventNotificationAction Function	567
WF_CASetListenInterval Function	567
WF_CASetListRetryCount Function	568
WF_CASetMaxChannelTime Function	569
WF_CASetMinChannelTime Function	569
WF_CASetProbeDelay Function	570
WF_CASetRssi Function	570
WF_CASetScanCount Function	571
WF_CASetScanType Function	571
WFCAElementsStruct Structure	572
Connection Algorithm Internal Members	573
LowLevel_CAGetElement Function	574
LowLevel_CASetElement Function	574
SetEventNotificationMask Function	575
<b>Wi-Fi Connection Manager</b>	<b>576</b>
Connection Manager Public Members	576
WF_CMConnect Function	576
WF_CMDisconnect Function	577
WF_CMGetConnectionState Function	577
WF_CMInfoGetFSMStats Function	578
<b>Wi-Fi Scan</b>	<b>578</b>
Scan Public Members	578
WF_Scan Function	579
WF_ScanGetResult Function	580
<b>Wi-Fi Tx Power Control</b>	<b>580</b>
Tx Power Control Public Members	580
WF_TxPowerGetMinMax Function	581
WF_TxPowerSetMinMax Function	581
WF_TxPowerGetFactoryMax Function	582
<b>Wi-Fi Power Save</b>	<b>582</b>
Power Save Public Members	583
WF_GetPowerSaveState Function	583

---

WF_HibernateEnable Function	584
WF_PsPollDisable Function	585
WF_PsPollEnable Function	585
Power Save Internal Members	586
SendPowerModeMsg Function	586
SetPowerSaveState Function	586
<b>Wi-Fi Miscellaneous</b>	<b>587</b>
Wi-Fi Miscellaneous Public Members	587
WF_GetDeviceInfo Function	588
WF_GetMacAddress Function	588
WF_GetMacStats Function	589
WF_GetMultiCastFilter Function	589
WF_GetRegionalDomain Function	590
WF_GetRtsThreshold Function	591
WF_SetMacAddress Function	591
WF_SetMultiCastFilter Function	592
WF_SetRegionalDomain Function	592
WF_SetRtsThreshold Function	593
tWFDeviceInfoStruct Structure	593
WFMacStatsStruct Structure	594
<b>WF_ProcessEvent</b>	<b>595</b>
<b>Access Point Compatibility</b>	<b>597</b>
<b>WiFi Tips and Tricks</b>	<b>599</b>
<b>Hot Topics</b>	<b>600</b>
<b>Index</b>	<b>a</b>

# 1 Introduction

## Welcome to the Microchip TCP/IP Stack!

The Microchip TCP/IP Stack provides a foundation for embedded network applications by handling most of the interaction required between the physical network port and your application. It includes modules for several commonly used application layers, including HTTP for serving web pages, SMTP for sending e-mails, SNMP for providing status and control, Telnet, TFTP, Serial-to-Ethernet and much more. In addition, the stack includes light-weight and high-performance implementations of the TCP and UDP transport layers, as well as other supporting modules such as IP, ICMP, DHCP, ARP, and DNS.

This help file serves two purposes. The first is to be a guide for first-time users of the TCP/IP Stack. The Getting Started section begins a series of pages to help you become familiar with the stack and configure it for use on a Microchip development board.

The second purpose is to serve as a programmer's reference guide to the features and APIs available in the TCP/IP Stack.

## Updates

The latest version of the Microchip TCP/IP Stack is always available at <http://www.microchip.com/tcpip>. New features are constantly being added, so check there periodically for updates and bug fixes.

Wi-Fi<sup>®</sup> is a registered trademark of the Wi-Fi Alliance.

---

## 1.1 Getting Help

The TCP/IP Stack is supported through Microchip's standard support channels. If you encounter difficulties, you may submit ticket requests at <http://support.microchip.com>.

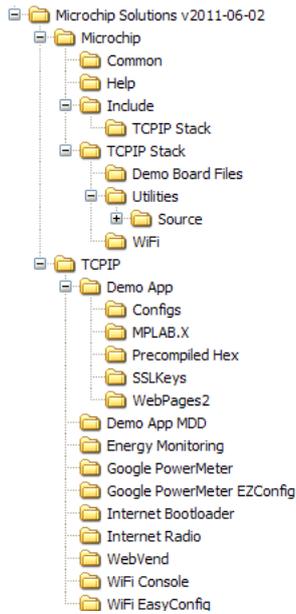
The Microchip forums are also an excellent source of information, with a very lively community dedicated specifically to Ethernet and TCP/IP discussions at <http://forum.microchip.com>.

Microchip also offers embedded network classes through Regional Training Centers. For more information, visit <http://www.microchip.com/rtc>.

---

## 1.2 Directory Structure

The TCP/IP Stack comes with many files, tools, documents, and project examples. Before getting started, take a moment to familiarize yourself with the directory structure so that you may find what you need quickly. Installing the stack will produce the following structure:



Several demonstration projects are installed into the **TCPIP** directory in the default **Microchip Solutions v20xx-xx-xx** directory. In your projects, you may wish to write your application code in a project folder located in the same place as the demo project folders. For more information about specific demos, see the list of demo projects (see page 81) in this help file. These project folders may contain additional subdirectories:

- A **Configs** subdirectory will contain alternative copies of the TCPIPConfig.h and HardwareProfile.h configuration files, pre-configured to work with different demo boards. The default copies of these files in the demo folder will include one of these alternative files based on a macro defined in the demo project.
- An **SSLKeys** subdirectory will contain sample security keys and certificates.
- A **WebPages2** subdirectory will contain sample web pages for use with the MPFS2 file system.
- An **MPLAB.X** project folder contains the MPLAB X project files for the demo.
- A **Precompiled Hex** subdirectory contains precompiled hex images of the demo project.

Other stack-specific folders include are:

- The **Microchip** folder contains stack files and components.
- The **Include** sub-folder under the **Microchip** folder contains header files for Microchip stack and library solutions. The **TCPIP Stack** folder in the **Include** folder contains headers for the TCP/IP Stack.
- The **TCPIP Stack** folder in the **Microchip** folder contains C source files, documentation, and stack utilities.
  - The **Demo Board Files** subdirectory contains information about the different demo boards (see page 62) that can run the TCP/IP stack.
  - The **Utilities** subdirectory contains PC-based utilities that can help when using the stack. See the Software (see page 57) section for more information. The source code for the Microchip TCP/IP Discoverer (see page 60), the MPFS2 (see page 57) tool, and the MPFS library is located in the **Source** subdirectory in the **Utilities** directory.

In most cases, it will not be necessary to modify source or header files in the **Microchip** directory.

## 2 SW License Agreement

**MICROCHIP IS WILLING TO LICENSE THE ACCOMPANYING SOFTWARE AND DOCUMENTATION TO YOU ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE FOLLOWING TERMS. TO ACCEPT THE TERMS OF THIS LICENSE, CLICK "I ACCEPT" AND PROCEED WITH THE DOWNLOAD OR INSTALL. IF YOU DO NOT ACCEPT THESE LICENSE TERMS, CLICK "I DO NOT ACCEPT," AND DO NOT DOWNLOAD OR INSTALL THIS SOFTWARE. BY DOWNLOADING AND INSTALLING THE SOFTWARE, LICENSEE AGREES TO BE BOUND BY THE TERMS OF THIS AGREEMENT.**

### **NON-EXCLUSIVE SOFTWARE LICENSE AGREEMENT FOR ACCOMPANYING MICROCHIP SOFTWARE AND DOCUMENTATION**

This Nonexclusive Software License Agreement ("Agreement") is a contract between you, your heirs, successors and assigns ("Licensee") and Microchip Technology Incorporated, a Delaware corporation, with a principal place of business at 2355 W. Chandler Blvd., Chandler, AZ 85224-6199, and its subsidiary, Microchip Technology (Barbados) II Incorporated (collectively, "Microchip") for the accompanying Microchip software including, but not limited to, Graphics Library Software, IrDA Stack Software, MCHPFSUSB Stack Software, Memory Disk Drive File System Software, mTouch(TM) Capacitive Library Software, Smart Card Library Software, TCP/IP Stack Software, MiWi(TM) DE Software, and/or any PC programs and any updates thereto (collectively, the "Software"), and accompanying documentation, including images and any other graphic resources provided by Microchip ("Documentation").

The Software and Documentation are licensed under this Agreement and not sold. U.S. copyright laws, international copyright treaties, and other intellectual property laws and treaties protect the Software and Documentation. Microchip reserves all rights not expressly granted to Licensee in this Agreement.

1. **Definitions.** As used in this Agreement, the following capitalized terms will have the meanings defined below:

1. **"Microchip Products"** means Microchip microcontrollers and Microchip digital signal controllers.
2. **"Licensee Products"** means Licensee products that use or incorporate Microchip Products.
3. **"Object Code"** means the Software computer programming code that is in binary form (including related documentation, if any), and error corrections, improvements, modifications, and updates.
4. **"Source Code"** means the Software computer programming code that may be printed out or displayed in human readable form (including related programmer comments and documentation, if any), and error corrections, improvements, modifications, and updates.
5. **"Third Party"** means Licensee's agents, representatives, consultants, clients, customers, or contract manufacturers.
6. **"Third Party Products"** means Third Party products that use or incorporate Microchip Products.

2. **Software License Grant.** Microchip grants strictly to Licensee a non-exclusive, non-transferable, worldwide license to:

1. use the Software in connection with Licensee Products and/or Third Party Products;
2. if Source Code is provided, modify the Software, provided that no Open Source Components (defined in Section 5 below) are incorporated into such Software in such a way that would affect Microchip's right to distribute the Software with the limitations set forth herein and provided that Licensee clearly notifies Third Parties regarding such modifications;
3. distribute the Software to Third Parties for use in Third Party Products, so long as such Third Party agrees to be bound by this Agreement (in writing or by "click to accept" (☑ see page 163)) and this Agreement accompanies such distribution;
4. sublicense to a Third Party to use the Software, so long as such Third Party agrees to be bound by this Agreement (in writing or by "click to accept" (☑ see page 163));
5. with respect to the **TCP/IP Stack Software**, Licensee may port the ENC28J60.c, ENC28J60.h, ENCX24J600.c, and ENCX24J600.h driver source files to a non-Microchip Product used in conjunction with a Microchip ethernet controller;
6. with respect to the **MiWi (TM) DE Software**, Licensee may only exercise its rights when the Software is embedded on a Microchip Product and used with a Microchip radio frequency transceiver or UBEC UZ2400 radio frequency transceiver which are integrated into Licensee Products or Third Party Products.

For purposes of clarity, Licensee may NOT embed the Software on a non-Microchip Product, except as described in this Section.

3. **Documentation License Grant.** Microchip grants strictly to Licensee a non-exclusive, non-transferable, worldwide license to use the Documentation in support of Licensee's authorized use of the Software
4. **Third Party Requirements.** Licensee acknowledges that it is Licensee's responsibility to comply with any third party license terms or requirements applicable to the use of such third party software, specifications, systems, or tools. Microchip is not responsible and will not be held responsible in any manner for Licensee's failure to comply with such applicable terms or requirements.
5. **Open Source Components.** Notwithstanding the license grant in Section 1 above, Licensee further acknowledges that certain components of the Software may be covered by so-called "open source" software licenses ("Open Source Components"). Open Source Components means any software licenses approved as open source licenses by the Open Source Initiative or any substantially similar licenses, including without limitation any license that, as a condition of distribution of the software licensed under such license, requires that the distributor make the software available in source code format. To the extent required by the licenses covering Open Source Components, the terms of such license will apply in lieu of the terms of this Agreement, and Microchip hereby represents and warrants that the licenses granted to such Open Source Components will be no less broad than the license granted in Section 1. To the extent the terms of the licenses applicable to Open Source Components prohibit any of the restrictions in this Agreement with respect to such Open Source Components, such restrictions will not apply to such Open Source Component.
6. **Licensee Obligations.**
  1. Licensee will ensure Third Party compliance with the terms of this Agreement, and will be liable for any breach of this Agreement committed by such Third Party.
  2. Licensee will not: (i) engage in unauthorized use, modification, disclosure or distribution of Software or Documentation, or its derivatives; (ii) use all or any portion of the Software, Documentation, or its derivatives except in conjunction with Microchip Products or Third Party Products; or (iii) reverse engineer (by disassembly, decompilation or otherwise) Software or any portion thereof.
  3. Licensee may not remove or alter any Microchip copyright or other proprietary rights notice posted in any portion of the Software or Documentation.
  4. Licensee will defend, indemnify and hold Microchip and its subsidiaries harmless from and against any and all claims, costs, damages, expenses (including reasonable attorney's fees), liabilities, and losses, including without limitation product liability claims, directly or indirectly arising from or related to: (i) the use, modification, disclosure or distribution of the Software, Documentation, or any intellectual property rights related thereto; (ii) the use, sale and distribution of Licensee Products or Third Party Products; and (iii) breach of this Agreement. THIS SECTION 3(d) STATES THE SOLE AND EXCLUSIVE LIABILITY OF THE PARTIES FOR INTELLECTUAL PROPERTY INFRINGEMENT.
7. **Confidentiality.** Licensee agrees that the Software (including but not limited to the Source Code, Object Code and library files) and its derivatives, Documentation and underlying inventions, algorithms, know-how and ideas relating to the Software and the Documentation are proprietary information belonging to Microchip and its licensors ("Proprietary Information"). Except as expressly and unambiguously allowed herein, Licensee will hold in confidence and not use or disclose any Proprietary Information and will similarly bind its employees and Third Party(ies) in writing. Proprietary Information will not include information that: (i) is in or enters the public domain without breach of this Agreement and through no fault of the receiving party; (ii) the receiving party was legally in possession of prior to receiving it; (iii) the receiving party can demonstrate was developed by the receiving party independently and without use of or reference to the disclosing party's Proprietary Information; or (iv) the receiving party receives from a third party without restriction on disclosure. If Licensee is required to disclose Proprietary Information by law, court order, or government agency, Licensee will give Microchip prompt notice of such requirement in order to allow Microchip to object or limit such disclosure. Licensee agrees that the provisions of this Agreement regarding unauthorized use and nondisclosure of the Software, Documentation and related Proprietary Rights are necessary to protect the legitimate business interests of Microchip and its licensors and that monetary damage alone cannot adequately compensate Microchip or its licensors if such provisions are violated. Licensee, therefore, agrees that if Microchip alleges that Licensee or Third Party has breached or violated such provision then Microchip will have the right to petition for injunctive relief, without the requirement for the posting of a bond, in addition to all other remedies at law or in equity.
8. **Ownership of Proprietary Rights.** Microchip and its licensors retain all right, title and interest in and to the Software and Documentation ("Proprietary Rights") including, but not limited to all patent, copyright, trade secret and other intellectual property rights in the Software, Documentation, and underlying technology and all copies and derivative works thereof (by whomever produced). Further, copies and derivative works will be considered works made for hire with ownership vesting in Microchip on creation. To the extent such modifications and derivatives do not qualify as a "work for hire," Licensee hereby irrevocably transfers, assigns and conveys the exclusive copyright thereof to Microchip, free and clear of any and all liens, claims or other encumbrances, to the fullest extent permitted by law. Licensee and Third Party use of such modifications and derivatives is limited to the license rights described in Sections this Agreement.
9. **Termination of Agreement.** Without prejudice to any other rights, this Agreement terminates immediately, without notice

by Microchip, upon a failure by Licensee or Third Party to comply with any provision of this Agreement. Upon termination, Licensee and Third Party will immediately stop using the Software, Documentation, and derivatives thereof, and immediately destroy all such copies.

10. **Warranty Disclaimers.** THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE. MICROCHIP AND ITS LICENSORS ASSUME NO RESPONSIBILITY FOR THE ACCURACY, RELIABILITY OR APPLICATION OF THE SOFTWARE OR DOCUMENTATION. MICROCHIP AND ITS LICENSORS DO NOT WARRANT THAT THE SOFTWARE WILL MEET REQUIREMENTS OF LICENSEE OR THIRD PARTY, BE UNINTERRUPTED OR ERROR-FREE. MICROCHIP AND ITS LICENSORS HAVE NO OBLIGATION TO CORRECT ANY DEFECTS IN THE SOFTWARE. LICENSEE AND THIRD PARTY ASSUME THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE AND DOCUMENTATION PROVIDED UNDER THIS AGREEMENT.
11. **Limited Liability.** IN NO EVENT WILL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL OR EQUITABLE THEORY FOR ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS. The aggregate and cumulative liability of Microchip and its licensors for damages hereunder will in no event exceed \$1000 or the amount Licensee paid Microchip for the Software and Documentation, whichever is greater. Licensee acknowledges that the foregoing limitations are reasonable and an essential part of this Agreement. LICENSEE ACKNOWLEDGES AND AGREES THAT IT IS SOLELY RESPONSIBLE FOR THE LICENSEE PRODUCTS AND THIRD PARTY PRODUCTS, INCLUDING DETERMINING WHETHER SUCH PRODUCTS INFRINGE A PATENT, COPYRIGHT OR OTHER PROPRIETARY RIGHT OF ANY THIRD PARTY. LICENSEE AGREES THAT MICROCHIP HAS NO OBLIGATION TO INDEMNIFY OR DEFEND LICENSEE IN THE EVENT THAT A THIRD PARTY MAKES A CLAIM REGARDING LICENSEE PRODUCTS OR THIRD PARTY PRODUCTS.
12. **General.** THIS AGREEMENT WILL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF THE STATE OF ARIZONA AND THE UNITED STATES WITHOUT REGARD TO CONFLICTS OF LAWS PROVISIONS. Licensee agrees that any disputes arising out of or related to this Agreement, Software or Documentation will be brought in the courts of the State of Arizona. The parties agree to waive their rights to a jury trial in actions relating to this Agreement. If either the Microchip or Licensee employs attorneys to enforce any rights arising out of or relating to this Agreement, the prevailing party will be entitled to recover its reasonable attorneys' fees, costs and other expenses. This Agreement will constitute the entire agreement between the parties with respect to the subject matter hereof. It will not be modified except by a written agreement signed by an authorized representative of the Microchip. Microchip's authorized representatives will have the right to reasonably inspect Licensee's premises and to audit Licensee's records and inventory of Licensee Products in order to ensure Licensee's adherence to the terms of this Agreement. If any provision of this Agreement will be held by a court of competent jurisdiction to be illegal, invalid or unenforceable, that provision will be limited or eliminated to the minimum extent necessary so that this Agreement will otherwise remain in full force and effect and enforceable. No waiver of any breach of any provision of this Agreement will constitute a waiver of any prior, concurrent or subsequent breach of the same or any other provisions hereof, and no waiver will be effective unless made in writing and signed by an authorized representative of the waiving party. Licensee agrees to comply with all export laws and restrictions and regulations of the Department of Commerce or other United States or foreign agency or authority. The indemnities, obligations of confidentiality, and limitations on liability described herein, and any right of action for breach of this Agreement prior to termination, will survive any termination of this Agreement. Neither this Agreement nor any rights, licenses or obligations hereunder, may be assigned by Licensee without the prior written approval of Microchip except pursuant to a merger, sale of all assets of Licensee or other corporate reorganization, provided that assignee agrees in writing to be bound by the Agreement. Any prohibited assignment will be null and void. Use, duplication or disclosure by the United States Government is subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause of FAR 52.227-19 when applicable, or in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, and in similar clauses in the NASA FAR Supplement. Contractor/manufacturer is Microchip Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85224-6199.

If Licensee has any questions about this Agreement, please write to Microchip Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85224-6199 USA. ATTN: Marketing.

Copyright (c) 2011 Microchip Technology Inc. All rights reserved.

License Rev. No. 03-060111

## 3 Release Notes

\*\*\*\*\*

Microchip TCP/IP Stack Version Log:

\*\*\*\*\*

### Remarks

Please file bug-reports/bug-fixes at <http://support.microchip.com/> or post to the Microchip TCP/IP -> Ethernet Forum at <http://forum.microchip.com/> Look for stack updates at <http://www.microchip.com/mal/>

\*\*\*\*\*

v5.36 2 June 2011

\*\*\*\*\*

Changes:

1. Because of changes to the SHOUTcast server configuration, the Internet Radio demo is no longer functional. This demo has been retained in the stack distribution to provide a TCP/IP code example.
2. The UDP module will now perform address resolution and DNS querying automatically. As a result, the UDP module APIs have changed. The UDPOpenEx (see page 519) function provides this additional functionality. Please consult the TCP/IP Stack Help File's "Stack API > UDP" topic for more information.
3. The UDPOpen (see page 520) macro has been added to conform to the legacy UDPOpen (see page 520) interface.
4. The Announce (see page 149), BerkeleyAPI, DHCP client/server, DNS client/server, NBNS, Reboot (see page 311), SNMP, SNTP, TFTPc, UDPPerformanceTest, and ZeroConf modules have been updated to use the new UDP API. The iPerf demo has also been updated.
5. The MPFS Classic and HTTP(1) modules have been removed from the stack. Function-ality to support these modules has also been removed from the TCP/IP Stack software tools. MPFS2 and HTTP2 are still supported.
6. The UARTConfig demo module has been updated to upload MPFS2 images to the demo board in place of MPFS Classic images.
7. To facilitate linking on PIC18 platforms, the number of UDP sockets in demo projects has been reduced from 10 to 8.
8. The SNMP Stack application and mib2bib.jar PC utility now both support 1024 dynamic IDs.
9. SNMP\_DEMO\_TRAP is a new dynamic variable added to the snmp.mib file to support SMlv2 with TRAPv2. This will correct a previously existing issue viewing traps with the iReasoning MIB browser. As per those changes, the mchp.mib file has been modified to support the SMlv2 standard. This mib includes MODULE-IDENTITY which will provide MICROCHIP and MIB information. snmp.mib also includes MODULE-IDENTIY(1), a new number (1) to the existing OID after ENTERPRISE-ID(17095).
10. Added a preprocessor check that will include the ultoa function if a version of the C32 compiler earlier than 1.12 is used.
11. Modified the WiFi module to use separate retry counters for AdHoc and Infrastructure modes.
12. Modified Berkeley API module to accept (see page 163) IPPROTO\_IP (see page 168) as a valid protocol for the socket function. The code will determine the protocol type from the socket type (datagram or stream).
13. Created MPLAB X projects corresponding to most MPLAB 8 projects and configurations. These projects are located in the MPLAB.X subfolder in the associated demo project directory. The MPLAB X import wizard can be used to create MPLAB X projects from MPLAB 8 projects that don't have an analogue in the new demo project folders.
14. Added project support for the dsPIC33E and PIC24E architectures.
15. All TCP/IP Stack demo projects have been moved to the "TCPIP" subdirectory in the stack installation directory.
16. Created Java versions of several TCP/IP tools to provide cross-platform support. The TCP/IP Configuration Wizard has not been ported to Java; thus, it is only available for Windows users.

17. To prevent issues with path length, MPLAB 8 project names have been changed. A list of the abbreviations used in the project names is available in the MAL help folder (Microchip Solutions/Microchip/Help/Abbreviations.htm). The names of the HardwareProfile and TCPIPConfig configuration files have been abbreviated as well.
18. Changed the configuration inclusion macros used by the TCP/IP Stack demo projects to match the terms used in the project/configuration names.
19. The "Alternative Configurations" subfolders in most demo projects has been renamed to "Configs."
20. Added a Google PowerMeter demo for use with the PIC18F87J72 Energy Monitoring PICtail.
21. The Web Preview tool is no longer included with the stack.

Fixes:

1. Fixed a DHCP Server (DHCPs.c) lease leak problem that would occur when STACK\_USE\_ZEROCONF\_LINK\_LOCAL was defined. This problem would have resulted in the DHCP server stop giving out any leases until being rebooted.
2. Updated the PIC32MX6XX/7XX external PHY SMSC 8720LAN reference design.
3. Fixed bug with window expecting MACGetFreeRxSize() to return values < 32KB.
4. Fixed a type casting bug with the CalcIPChecksum (see page 210) function that would cause an incorrect TX checksum if the checksum value overflowed again after adding the carry bits to the checksum value.
5. Fixed a bug in the AutoIP module that may have prevented the module from correctly defending its own address.
6. Added a check to the Announce (see page 149) module to ensure the MAC layer is linked before attempting to transmit an Announce (see page 149) message.
7. Fixed a bug in the ETH97J60 MACPut function.
8. Added an additional preprocessor check in a debug menu setting in WF\_Spi.c to prevent a build error.
9. Added a fix to the Google PowerMeter demo code to restore SNTP timestamp sourcing if SNTP is enabled. Previously, it would be overwritten by a possibly invalid HTTP timestamp.

Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenale it's DHCP server.
3. LCDBlocking.c timing for the LCD\_E\_IO enable signal is too fast to meet published data sheet limits for various LCD controllers when compiling for a PIC32 running at > 43MHz. Despite this potential timing violation, the LCD does normally work correctly on the precompiled PIC32 demos for the Explorer 16 at 80MHz.
4. For PIC32 implementations, depending on the configuration, it is possible that the MACGetFreeRxSize() returns a value greater than 64 KB. For backward compatibility reasons the stack uses a 16 bit value to check the returned value and it won't work correctly.
5. Limiting the number of UDP sockets to 8 in the stack demos may prevent SNMP trap functionality. If this occurs, you can increase the MAX\_UDP\_SOCKETS definition in TCPIPConfig.h to 10 (if your system will support the increased data memory usage) to fix this issue.
6. The SNMP mib file's date and version parameter does not match the date/version of the current stack release.

\*\*\*\*\*

v5.31 19 October 2010

\*\*\*\*\*

Changes:

1. Reorganized demo projects. The TCPIP ENCX24J600 Demo App, TCPIP PIC32 ETH Demo App, and TCPIP WiFi Demo App projects in stack versions 5.25 and prior have all been merged into the TCPIP Demo App folder. All four of these projects were almost identical to each other, with the primary difference being the network interface controller the projects

were preconfigured to support. In this 5.31 TCP/IP Stack release, each hardware combination now has its own MPLAB IDE project in the TCPIP Demo App folder.

2. Reorganized HardwareProfile.h and TCPIPConfig.h structure for the TCPIP Demo App, TCPIP Google PowerMeter Demo App, and TCPIP WebVend App projects. Now, instead of having one massive HardwareProfile.h file that supports a multitude of hardware platforms simultaneously, simple individual hardware profiles have been created for specific hardware combinations and placed in the "Alternative Configurations" sub-folder. The correct hardware profile or TCPIP config file is selected by #including "HardwareProfile.h" or "TCPIPConfig.h" as in previous stack versions. However, the active hardware profile or config file from the Alternative Configurations folder is selected by a preprocessor macro defined in the MPLAB project settings (passed on the command line to the compiler).
3. Added HTTP\_SAVE\_CONTEXT\_IN\_PIC\_RAM option to TCPIPConfig.h (HTTP2 server module). This option, when enabled, will increase HTTP2 server performance when servicing multiple simultaneous connections at the expense of using more PIC RAM.
4. Added automatic TPIN+/- polarity swapping logic to ETH97J60.c driver for PIC18F97J60 family devices. Some 3rd party Ethernet switches, routers, and end devices have their TX+/- pins wired backwards such that the remote TX+ signal connects to the PIC TPIN- pin and the remote TX- signal connects to the PIC TPIN+ pin. Because 10BaseT Ethernet signaling is polarized and the PIC18F97J60 family does not implement an auto-polarity feature, this normally prevents all RX communications with these non-IEEE 802.3 compliant link partners. To work around this incompatibility, it is possible to implement circuitry to swap the RX+ and RX- signals before reaching the TPIN+ and TPIN- pins. The PICDEM.net 2 Rev 6 reference design includes this necessary circuitry (U6, U7, R54, and RX\_SWAP GPIO output pin from PIC). This stack version automatically controls the RX\_SWAP signal based on the ETH\_RX\_POLARITY\_SWAP\_TRIS and ETH\_RX\_POLARITY\_SWAP\_IO definitions in HardwareProfile.h. If these macros are undefined, then the automatic polarity swapping feature is disabled in the ETH97J60.c driver.
5. Added portable LFSRRand (see page 222)() and LFSRSeedRand (see page 223)() public functions to Helpers.c and removed all references to C rand() and srand() functions. The C rand() function returns a 15 bit integer on 8 and 16 bit PICs, but a 31 bit integer on PIC32s. The LFSRRand (see page 222)() function returns a 16-bit integer, regardless of which PIC you are using.
6. Added support for various SST/Microchip brand SST25xxxx SPI Flash chips to SPIFlash.c driver. Previously only devices supporting the Auto Address (see page 141) Increment (AAI) Word Program opcode (0xAD) would work. Now, devices such as the SST2525VF010A should work, which require the AAI Byte program opcode (0xAF) instead.
7. Removed support for Spansion brand SPI Flash chips in the SPIFlash.c driver. If your application is already using one of these devices, continue to use the SPIFlash.c/.h files from TCP/IP Stack 5.25. These older files are API compatible with the current version, so can be dropped in by simply overwriting the SPIFlash.c and SPIFlash.h files.
8. Removed a -4 offset from the advertised TCP Maximum Segment Size option (MSS) and TCP\_MAX\_SEG\_SIZE\_RX (see page 476) configuration value in TCP.c. The default TCP MSS advertised is now 536 instead of 532.
9. For Wi-Fi projects in the TCPIP Demo App folder, changed MY\_DEFAULT\_LIST\_RETRY\_COUNT to WF\_RETRY\_FOREVER instead of 3. This changes default connection behavior to keep trying to connect (see page 165) instead of just trying 3 times which makes more sense for demonstration.
10. Changed WF\_Connect() beacon timeout to 40.
11. IFConfig command in TCPIP WiFi Console Demo App modified to return application-perspective MAC address from the AppConfig structure, and not the Wi-Fi serialized MAC address (they may not match if user desired custom MAC).
12. Updated the TCP/IP Configuration Wizard. The user can now configure wireless settings and stack settings separately. Because of the changes to the TCPIPConfig.h file, the user must now select the specific copy of TCPIPConfig.h (or any of its variants) instead of selecting a project directory. Added the ability to select WF\_RETRY\_FOREVER in the Wi-Fi configuration settings. Added a selection parameter for BSD socket count. Added validation to check for the proper number of Berkeley sockets and TCP performance test sockets in the socket configuration screen (Advanced Settings) if either of these features are enabled. Added the ability to create sockets of the same type with different TX/RX buffer sizes in the socket configuration screen.
13. Updated the TCPIP WebVend Demo App to support Wi-Fi in several configurations.
14. Modified the Google PowerMeter demo to automatically determine the date/time from the HTTP module if the date/time cannot be obtained from the SNTP module.
15. Added a new Google Map project example to the Combo Demos folder. This example runs on a PIC24FJ256DA210 Development Board + Fast 100Mbps Ethernet PICtail Plus (or Ethernet PICtail Plus) + Truly 3.2" 240x320 display, TFT\_G240320LTSW\_118W\_E (or Powertip 4.3" 480x272 display, PH480272T\_005\_I11Q). It also can run on the PIC32 Multimedia Expansion Board + PIC32 Ethernet Starter Kit. This demo connects to the Internet, sends an HTTP query for a specific map tile to the Google Static Maps API, and then displays the compressed tile to the graphics display. For more information, see the "Getting Started - Running the Graphics Google Map Demo.htm" file in the Combo DemosGoogle Map folder.

16. Added preliminary SNMPv3 module. This module, enabled with the `STACK_USE_SNMPV3_SERVER` option in `TCPIPConfig.h`, implements the Simple Network Management Protocol, version 3. Among other things, SNMPv3 adds secure authentication and cryptographic privacy as compared to SNMPv2C. This implementation currently only supports AES encryption (no DES support). It also has only been tested with the PIC32 Ethernet Starter Kit (TCPIP Demo App - C32 - PIC32\_ENET\_SK\_DM320004\_INTERNAL\_ETHERNET.mcp MPLAB IDE project). SNMPv3 on PIC18, PIC24, and dsPIC platforms are not supported at this time. Because AES encryption has specialized United States export requirements, this TCP/IP Stack release does not include the required AES library to enable SNMPv3. To obtain the needed AES library, you must purchase SW300052 v2.6 or later. Older v2.5 and previous versions include AES related files on them, but do not include the new AES files required by SNMPv3. For more information on using SNMP, refer to the TCP/IP Stack Help (Demo Information -> Available Demos -> TCPIP Demo App -> Demo Modules -> Network Management (SNMP) Server).
17. Altered the `SaveAppConfig()` function in `MainDemo.c` to store a more robust signature to EEPROM/SPI Flash when saving the `AppConfig` structure. In v5.25 and prior stack versions, when EEPROM or SPI Flash memory was available, the stack would automatically write a one byte marker character to address `0x000000` in the EEPROM/Flash indicating if a valid `AppConfig` structure was stored in the non-volatile memory. This resulted in the EEPROM/Flash contents being organized like the following: Address (see page 141) Data Contents =====  
 ===== 0x000000: Marker Byte 0x000001: AppConfig structure  
 MPFS\_RESERVE\_BLOCK: Start of MPFS/MPFS2 image containing web pages In this stack version, EEPROM/Flash contents will now contain: Address (see page 141) Data Contents =====  
 ===== 0x000000: Length of AppConfig structure (16-bit integer)  
 0x000002: IP checksum of the AppConfig default values, as defined in `TCPIPConfig.h` and `WF_Config.h` (16-bit integer).  
 0x000004: IP checksum of the subsequent EEPROM/Flash bytes of the AppConfig values. 0x000006: AppConfig structure MPFS\_RESERVE\_BLOCK: Start of MPFS/MPFS2 image containing web pages

The additional checksums allow automatic detection to occur if you change one of the values in `TCPIPConfig.h` or `WF_Config.h` that affects `AppConfig`. If you change one of the values in code, then upon boot up, the application will automatically detect this change and start using the values that you selected in code. If, at run time, you decide to change the `AppConfig` values and commit the changes to EEPROM/Flash, then the stack will subsequently use the run-time saved values on future reboots. The checksum at offset `0x000004` ensures that if any corrupted `AppConfig` contents are found in EEPROM/Flash (ex: power is lost between writing the signature structure and actual `AppConfig` structure, or code unintentionally overwrites something in the `AppConfig` memory area), then the original defaults defined in `TCPIPConfig.h` and `WF_Config.h` will be used instead of the corrupted values. This EEPROM/SPI Flash change affects all projects except TCPIP Internet Radio App, TCPIP Internet Bootloader App, and all PIC32 Starter Kit projects since these projects do not have or use external EEPROM or SPI Flash memory.

#### Fixes:

- Fixed a UDP bug in which a transmitted packet would have been addressed to the wrong destination node if the UDP socket received a broadcast packet from a different remote node from the last received packet, but using the same source port number as the last received packet. The `FindMatchingSocket()` function in `UDP.c` will now always change the local socket parameters to send to the most recent remote node's unicast IP address, regardless of if the last received packet was addressed to a multicast or broadcast destination. Thanks go to Billy Walton for reporting this erroneous behavior. If you wish to change the destination IP/MAC addresses or port number for a UDP packet that you are ready to send, write the new parameters to the `UDPSocketInfo[SocketHandle]` global structure before calling `UDPFlush()`. This structure contains `remoteNode` and `remotePort` parameters for the remote IP address/MAC address and remote UDP port, respectively. You can also read these values to obtain the remote addressing parameters for the last received packet on the given UDP socket. Note that "SocketHandle" refers to the UDP socket handle returned by the `UDPOpen()` API, not the literal string "SocketHandle".
- Fixed ADC state save/restore bug in `GenerateRandomDWORD()` function in `Helpers.c`. PIC24, dsPIC, and PIC32 platforms require the ADC ON/ADON bit to be cleared before modifying certain other ADC register contents.
- Fixed an ENC28J60 MAC/MII register write timing violation when using a PIC24H or dsPIC at over 33MIPS. There was inadequate Chip Select hold time provided, violating the 210ns minimum specified in the ENC28J60 data sheet. This violation may have resulted in certain devices losing the ability to receive packets (due to the MARXEN bit, `MACON1<0>`, getting cleared unintentionally).
- Fixed an ENC24J600.c driver bug in which operating at 100Mbps with the ENC424J600/624J600 Ethernet controller, it would be possible for the `MACGetHeader()` function to issue a `Reset()` operation under rare circumstances. The PIC would reset whenever the PHY detected an illegal symbol during 4B5B decoding but guessed the correct 4B symbol such that no data corruption or CRC error occurred. This condition results in a valid packet being received but with the Received Ok Receive Status Vector bit being clear (`RSV<23> == 0`). This issue would become more probable when using very long Ethernet cables (ex: 100 meters) and receiving a lot of data.
- Fixed a TCP bug in which calling `TCPDisconnect()` to close a connection when the remote node's RX window was 0 bytes would have caused the stack to enter an infinite loop sending duplicate ACK packets.

6. Fixed Wi-Fi bug that caused assert condition if too many management messages were being received during data traffic.
7. Fixed Wi-Fi bug that caused WF\_EVENT\_CONNECTION\_REESTABLISHED event case to send the wrong notification to the app.
8. Fixed Wi-Fi bug that caused assert failure with Scratch move failure.
9. Fixed Wi-Fi bug in WF\_CAGetChannelList (see page 557)() and WF\_CAGetSecurity that caused failure.
10. Fixed Wi-Fi EasyConfig bug that required development boards to be manually reset even after new network was selected.
11. Fixed MRF24WB0 bug that caused assert if invalid WPA/WPA2 key was entered.
12. Fixed Wi-Fi power management bit behavior in PS-Poll frame that was causing some AP's to never send data or disconnect when in power save mode.
13. Fixed a TCP bug in which attempting to open a client TCP socket to a remote server, specified by IP address (not DNS address), that was offline, but who's MAC address was already cached by the ARP client, would result in endless back-offs. For example, when attempting to contact the remote node (that was not responding), the TCP module would have transmitted a SYN at time T=0 seconds, T=1s, T=3s, T=7s, T=15s, T=31s, T=63s, T=127s, T=255s, etc. The exponential back-off between retransmissions would grow indefinitely until the retransmission interval would have grown so large that effectively no-retransmissions would be occurring. Assuming the application wasn't written with its own timeout to prevent endless waiting, this would prevent the socket from automatically establishing the connection to the remote server once the server came back online. With this TCP fix, the exponential back off now saturates after TCP\_MAX\_RETRIES (see page 476) (5) back offs and continues to retransmit using the same interval. By default, this means SYN transmissions will occur at T=0 seconds, T=1s, T=3s, T=7s, T=15s, T=31s, T=63s, T=95s, T=127s, etc. After 5 back-offs the retransmission interval stops growing and stays constant at 32 seconds.
14. Fixed an RSA computation bug that would cause the RSA module to never complete if you attempted to compute  $y = x^e \% n$  where  $e = 3$  (or similar number < 256 with only 0, 1, or 2 bits set). Thanks go to Kevin Maidment for pointing this error out and suggesting a solution. Note, that this fix to RSA.c is not distributed with the ordinary TCP/IP Stack due to United States export restrictions. To get this fix, you must repurchase SW300052. This fix is included in SW300052 v2.6 or later. If you don't have CD media to identify the SW300052 version that you have, you can test the RSA.c file that you have. RSA.c in SW300052 v2.6 has a CRC32 checksum of 0x91F66711. RSA.c in v2.5 and prior had a checksum of 0xB1E8B0CC.

#### Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.
3. LCDBlocking.c timing for the LCD\_E\_IO enable signal is too fast to meet published data sheet limits for various LCD controllers when compiling for a PIC32 running at > 43MHz. Despite this potential timing violation, the LCD does normally work correctly on the precompiled PIC32 demos for the Explorer 16 at 80MHz.

\*\*\*\*\*

v5.25 07 May 2010

\*\*\*\*\*

#### Changes:

1. Added support for the Microchip MRF24WB0 802.11 WiFi controller (module part number MRF24WB0MA). This product is intended as a backwards compatible, drop in replacement to the ZG2100. The MRF24WB0MA should work with previous TCP/IP Stack versions as if it were a ZG2100M, but when the MRF24WB0MA is used with this (5.25) and future TCP/IP Stack versions, feature improvements inside the MRF24WB0 allow the TCP/IP Stack code/RAM size to be smaller and run faster.
2. Dropped support for the ZeroG ZG2100 802.11 WiFi controller. Applications that must stay with this device should continue to use TCP/IP Stack version 5.20b or earlier. All new projects or preexisting projects undergoing updates should

be developed with the MRF24WB0 instead.

3. The WiFi connection management state machines now run on the MRF24WB0 instead of the PIC host, freeing up code and data space. Connection profiles can be created and the connection algorithm fine-tuned by the PIC application. In the WiFi demos see the WF\_Connect function in MainDemo.c for an example of how to configure and then establish a WiFi connection. The programming model has changed to an API model which is documented in 'TCPIP Stack Help.chm'.
4. Changed "VERSION" macro definition in TCPIP.h to "TCPIP\_STACK\_VERSION". "VERSION" is overly generic and will likely conflict with other identical tokens one may use in their application code or source libraries.
5. Added support for the PIC24FJ256GA110, PIC24FJ256GB110 and PIC24FJ256GB210 PIMs for the Explorer 16. Note that when using the PIC24FJ256GA110 general purpose PIM, the Ethernet PICtail Plus, Fast 100Mbps Ethernet PICtail Plus, MRF24WB0MA Wi-Fi PICtail Plus, or other SPI PICtail daughter board should be installed in the middle SPI2 slot of the Explorer 16, not the ordinary topmost SPI1 slot used by other PIMs, including the PIC24FJ256GB110 and PIC24FJ256GB210 ones. The software is set up to use SPI2 for the PIC24FJ256GA110 PIM to avoid incompatibilities with silicon revision A3, which does not allow the SCK1 pin to be configured as a PPS output.
6. Added support for the PIC24FJ256DA210 Development Board.
7. Added TFTPUploadRAMFileToHost (see page 498()), TFTPUploadFragmentedRAMFileToHost (see page 497()) and TFTPGetUploadStatus (see page 496()) APIs to the TFTPc.c file. These APIs provide a very simple means of uploading a whole file to a remote TFTP server without requiring a great deal of application state machine logic. These APIs require the DNS client module to be enabled (STACK\_USE\_DNS must be defined, in addition to STACK\_USE\_TFTP\_CLIENT).
8. Added a dummy DNS Server module. This server always sends the local IP address in response to all queries received. When using the PIC DHCP server, its purpose is to allow a user to type anything into a web browser (ex: http://asdf/) and still receive the web page provided by the PIC, much as a hotel or airport WiFi router will serve to you before you've paid or agreed to the network's terms of service. This DNS server module is implemented in DNSs.c, requires one UDP socket, and is enabled via the STACK\_USE\_DNS\_SERVER option in TCPIPConfig.h.
9. Changed SPIFlash.h file defaults to target an SST SPI Flash with 4096 byte sectors instead of a Spansion Flash with 65536 byte sectors. These new defaults are, among other reasons, in support of the PIC24FJ256DA210 Development Board, which has an SST SST25VF016B on it.
10. Made TCP Keep-Alive packets consistently get sent TCP\_KEEP\_ALIVE\_TIMEOUT (see page 476) (default 10 seconds) after the last socket TX or RX activity. In earlier stack versions, if the local node transmitted some data and then let the socket go idle, the first Keep-Alive packet sent would use the TCP\_START\_TIMEOUT\_VAL (see page 479) (default 1 second) timer value before getting sent. While benign in terms of application behavior, these faster than normal keep-alive transmissions were distracting when viewed in Wireshark or other packet capture tools.
11. Disabled STACK\_USE\_DYNAMICDNS\_CLIENT option in TCPIPConfig.h by default for the TCPIP Demo App and TCPIP ENCX24J600 Demo App projects. This option was enabled by default in earlier stack releases. This was done to save code size and allow out-of-box compilation on devices with 128KB of Flash when not using compiler optimizations. The TCPIP PIC32 ETH Demo App project continues to have this option enabled by default.
12. Added SNMP v2 TRAP PDU format. Macro SNMP\_STACK\_USE\_V2\_TRAP is used to enable the SNMP v2 trap format. New API function SNMPV2TrapDemo() is included to support more than one variable binding to the SNMPv2 TRAP. This API can be used for a single SNMPv2 TRAP variable varbind and is part of CustomSNMPApp.c. A multiple variable binding demo can be enabled MainDemo.c. One should not enable both SNMPTrapDemo and SNMPV2TrapDemo simultaneously. Global flag "gSetTrapSendFlag (see page 318)" is used to indicate the start and end of SNMPv2 trap varbinds. If gSetTrapSendFlag (see page 318) is FALSE, then very next variable varbind for the SNMPv2 TRAP, is the last or only one variable varbind. If gSetTrapSendFlag (see page 318) is TRUE, then there is another variable varbind available to be part of the SNMPv2 TRAP PDU.
13. Added support for PIC32MX6XX/7XX external PHY's: SMSC 8700LAN and National DP83640.
14. Added schematics and BOM for the PIC32 Ethernet Starter Kit.
15. Added the Google PowerMeter demo project. Consult the "Reference Implementation for Google PowerMeter.chm" help file for more information.
16. Modified the SSL and TCP modules to create the TCPStartSSLClientEx (see page 465) function. This function will enable the SSL module to store supplementary data (currently only SSL Certificate Public Keys) in a structure.
17. Moved the HTTP\_PORT (see page 248), HTTPS\_PORT (see page 255), HTTP\_MAX\_DATA\_LEN (see page 248), and HTTP\_MIN\_CALLBACK\_FREE (see page 248) macros from HTTP2.c to TCPIPConfig.h.

#### Fixes:

1. The SPIFlashEraseSector() function in the SPIFlash.c file incorrectly erased the sector specified by the current write pointer (set by calling SPIFlashBeginWrite()) instead of the specified dwAddr parameter address. This error had no impact on any TCP/IP Stack code as these parameters always matched. However, application code using the API would

have been affected. Thanks go to Marc Boon for reporting this issue on the Microchip Ethernet forum.

2. Fixed ENC424J600/624J600 driver for PSP modes 2, 4, 6, and 10. The PIC's PMP PMMODE<9:8> bits were not set correctly.
3. Removed from lingering references to TickGetDiff() in FTP.c, TFTPc.c and UARTConfig.c.
4. Fixed DNS client module from returning the DNS server IP address if the DNS query failed due to a server error (i.e. DNS did respond, but did not return any records, such as when attempting to resolve a name that isn't in the DNS). DNSIsResolved (see page 181)() will now return 0.0.0.0 on any non-recoverable DNS error or timeout.
5. Fixed HTTP2 MPFS upload page being accessible from URLs that weren't an exact match. For example, in 5.20 and earlier, accessing <http://mchpboard/mpfsuploadASDF> would still have opened the mpfsupload page. Thanks go to Andrea Rivolta on the Microchip Ethernet Forum for identifying this error.
6. Improved UDP TX checksum code for the special case when the computed checksum was 0x0000. According to the UDP RFC, for this corner case, the checksum should be converted to 0xFFFF before transmission to differentiate from the checksum disabled case, improving error detection by a minuscule amount.
7. Fixed GetCLKOUT() function in ENCX24J600.c driver file. Previously, 0x00 would always be returned, regardless of the value in the COCON bits of ECON2. The function documentation for SetCLKOUT() and GetCLKOUT() was also corrected (had obsolete information ported over from ENC28J60 driver file).
8. Fixed DHCP client rebinding bug in which the DHCP client would request the wrong IP address if an unrelated DHCP OFFER or ACK message were received after we transmitted a DHCP REQUEST but before we received our DHCP ACK. Under rare conditions, this would have resulted in the TCP/IP stack reverting to the static or AutoIP assigned address for a few seconds between DHCP lease renewals.
9. Fixed TFTP Internet Bootloader bug in which uncommon .hex files containing a certain data pattern could not be uploaded correctly to the PIC18F97J60 family device. For these problem .hex files, a block of 32 program words (64 bytes) would remain unprogrammed (left as 0xFFFF) due to a parsing error in the bootloader's DecodeHex() function. The TFTP upload operation would succeed without reporting a programming error. The problem can be detected by using an ICD3 or similar ICSP programmer and reading the program Flash out of a device that is programmed with the bootloader and application .hex files. Compare the resulting memory dump to a device programmed only with the application .hex file. If you have devices deployed in the field with the previous bootloader and happen to generate a problem application .hex file, you can potentially work around the bootloader bug by opening the application .hex file with Notepad and appending dummy address records to the beginning to move the data around in the file. For example, at the very top of the .hex file, add lines containing ":020000040000FA" until the bootload process works correctly. You may alternatively try adding spaces at the end of any line, although this may make the .hex file incompatible with some programming utilities. Thanks go to Jonathan Seidmann for identifying and reporting this bug.
10. Fixed SNMPv2 TRAP format issue where SNMP browser was displaying all the SNMPv2 traps as SNMP version 1. SNMP v2 TRAP pdu format is rectified. Macro SNMP\_STACK\_USE\_V2\_TRAP is used to form and send a SNMPv2 TRAP PDU. SNMPTrapDemo API is used for both SNMPv1 and SNMPv2 single variable varbind trap.
11. Fixed an HTTP2.c server module initialization bug when using the PIC32MX7XX/6XX series internal Ethernet module. During initialization the HTTPLoadConn (see page 253)() function would overwrite over 100 bytes of PIC RAM past the end of the reserved memory allocated for the HTTP2 module. This problem would manifest itself by locking up the TCP/IP PIC32 ETH Demo App-C32 demo shortly after power up if you compiled TCP/IP Stack version 5.20 with the MPLAB C Compiler for PIC32 MCUs (C32) version 1.11.
12. Fixed SSL client from incorrectly parsing for the server's public key in rare cases where the RSA Public Key Algorithm identifier was received, but the key hadn't been received by TCP yet. Thanks go to Kevin Maimdnet for identifying this error in SSL.c and reporting it via <http://support.microchip.com/>.
13. Fixed Tick.c TickGet (see page 512)(), TickGetDiv256 (see page 512)() and TickGetDiv64K (see page 513)() APIs sometimes returning the wrong value on PIC32 platforms. On the PIC32MX3XX/4XX family devices a wrong return result would sometimes occur if using -O3 compiler optimizations (maximum speed) with the Microchip MPLAB C Compiler for PIC32 MCUs (C32). On the PIC32MX5XX/6XX/7XX family devices, such as the PIC32MX795F512L device used on the PIC32 Ethernet Starter Kit, wrong values could be returned, regardless of the compiler optimization level.

#### Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenale it's DHCP server.

3. HI-TECH PICC-18 compilers are not supported in this release. The supplied HI-TECH PICC-18 MPLAB projects usually will not compile and/or link.
4. LCDBlocking.c timing for the LCD\_E\_IO enable signal is too fast to meet published data sheet limits for various LCD controllers when compiling for a PIC32 running at > 43MHz. Despite this potential timing violation, the LCD does normally work correctly on the precompiled PIC32 demos for the Explorer 16 at 80MHz.

\*\*\*\*\*

v5.20 18 November 2009

\*\*\*\*\*

#### Changes:

1. Added PIC32MX7XX/6XX Family integrated Ethernet controller support. The "TCPIP PIC32 ETH Demo App" folder was added to compile for the PIC32 Ethernet Starter Kit. Ethernet driver files "ETHPIC32ExtPhy.c" and "ETHPIC32IntMac.c" were added, in addition to the "ETHPIC32ExtPhyDP83848.c" file, which is a specific driver file for the National DP83848 10/100 PHY.
2. Added RFC 3927 Auto IP module. This module will automatically assign a local IP address to the node in the 169.254.xxx.xxx private address range (subnet mask 255.255.0.0) if a DHCP server is not present on the network or the DHCP client is disabled. The exact IP address chosen will be pseudo-random, but as required by the protocol, it will perform gratuitous ARPs to avoid clobbering anyone else's IP address. Also, unless there is an address collision with a preexisting node on the network, the IP address generated by the Auto IP module will not change between power cycle events (random number generator is seeded by local MAC address). To enable this module, STACK\_USE\_AUTO\_IP must be defined in TCPIPConfig.h. When compiled in, the module defaults to enabled, but will automatically yield to the DHCP client module, which has higher priority.
3. Added "TCPIP MDD Demo App" beta application projects. Projects in this folder store the HTTP2 web pages in external FAT16/FAT32 formatted SD card or USB Mass Storage media instead of an MPFS2 formatted EEPROM or SPI Flash. For more information on these projects, see the "Running the TCPIP MDD Demo App (Beta Release).pdf" file in the MicrochipHelp folder.
4. Expanded XEEReadArray() API's third length parameter from a BYTE to a WORD.
5. Converted all variable declarations and type casts of TICK data type to DWORD. The TICK typedef is now deprecated and will be removed in a future stack release. This data type conflicts with the TICK structure used in certain other Microchip software libraries.
6. Added TCP\_WINDOW\_UPDATE\_TIMEOUT\_VAL (see page 474) option to the TCP.c file (default 200ms). This timeout controls the time after calling TCPGet (see page 446)() or TCPGetArray (see page 447)() before the stack will transmit a RX window update to the remote node. Historically, the TCP\_AUTO\_TRANSMIT\_TIMEOUT\_VAL (see page 474) value was used for this purpose (default 40ms). This change decreases the net window update transmission overhead. If this adversely affects your application RX performance (unlikely, but possible for certain communications patterns), set TCP\_WINDOW\_UPDATE\_TIMEOUT\_VAL (see page 474) equal to or shorter than TCP\_AUTO\_TRANSMIT\_TIMEOUT\_VAL (see page 474) to get the same or better behavior relative to previous stack versions.
7. Split TCP\_MAX\_SEG\_SIZE configuration constant in TCP.c into separate TCP\_MAX\_SEG\_SIZE\_TX (see page 477) and TCP\_MAX\_SEG\_SIZE\_RX (see page 476) configuration constants. Previously, TCP\_MAX\_SEG\_SIZE was used to limit both the maximum size of transmit and receive packets. In cases where large TX FIFOs are allocated, and the remote node advertises a large Maximum Segment Size TCP option, this change improves TCP transmit performance by roughly 10%.
8. Renamed "Internet Radio App", "Internet Bootloader App" and "WiFi Iperf App" folders to "TCPIP Internet Radio App", "TCPIP Internet Bootloader App" and "TCPIP WiFi Iperf App" respectively. These new names ensure consistent folder placement when viewing the Microchip Solutions folder with other Microchip Application Libraries installed.

#### Fixes:

1. Fixed SSL functionality (ex: HTTPS server) from failing when using the ENC424J600 and ENC624J600 Ethernet controllers. In stack versions 5.00 and 5.10, the BFSReg() and BFCReg() functions were being incorrectly used to set and clear CRYPTEN (EIR<15>). ENC424J600/624J600 silicon errata #6 on production silicon revision A2 prevents BFSReg() and BFCReg() from being able to modify CRYPTEN. This resulted in the SSL RSA encrypt/decrypt operations from ever finishing. The ENC424J600/624J600 errata #6 workaround is now implemented in the ToggleCRYPTEN() function in ENCX24J600.c.

2. Fixed an RSA padding error in the ENCX24J600.c's version of RSASetData() and RSASetE() functions. This fixes the Bad Record MAC problem when using SSL client APIs with the ENC424J600 and ENC624J600, as mentioned in the 5.10 stack release notes' Known Problems section. Although unknown at the time of release this problem also occurred in stack version 5.00.
3. Fixed DNS client from mishandling DNS responses that did not use name compression. Thanks go to Will Stone on the Microchip Ethernet forum for identifying this bug.
4. Fixed an ExtractURLFields (see page 211)() API bug which would incorrectly parse the URLs containing other URLs. Ex: "http://www.google.com/search?q=http://www.microchip.com/"
5. Fixed TickGet (see page 512)(), TickGetDiv256 (see page 512)(), and TickGetDiv64K (see page 513)() APIs from potentially returning an incorrect time value (0x10000 ticks less than it should have) on rare occasions when using a PIC32 and with compiler optimizations turned on. The Tick.c module was also revised so that the IEC0 register does not get written to via a load-modify-store operation on PIC32s so that it is now possible for other application ISR functions to write to IEC0 without risking state corruption.
6. Fixed PIC32 Starter Kit Debugger losing access to the PIC32 target when the project was run. JTAG was being disabled at run time, but the PIC32 Starter Kit Debugger requires JTAG to communicate with the debug executive. JTAG is now conditionally disabled on PIC32s when the `__MPLAB_DEBUGGER_PIC32MXSK` macro is undefined.
7. Fixed a Berkeley sockets API bug in which calling closesocket (see page 165)() on a SOCK\_STREAM (see page 172) type socket (TCP) did not actually close the socket if the remote node did not first send a FIN to the local node. This would leak a TCP socket each time the affected API calling sequence occurred and result in no FIN getting transmitted to the remote node.
8. Fixed an HTTP2 filename parsing bug that would occur when a web browser submitted a request for a file with hex encoded characters in it. For example, with stack version 5.10 and Firefox 3.5.3, typing "http://mchpboard/%70rotect" into the URL field would have resulted in an HTTP 404 not found error when "http://mchpboard/protect/index.htm" should have been returned instead. Thanks go to Steve Tuttle for reporting this issue and suggesting a solution.
9. Fixed a Berkeley sockets API bug in which calling recvfrom (see page 170)() on a datagram type socket (UDP) would return an incorrect remote IP address and port number when the from pointer was non-NULL.
10. Fixed HTTP2 server bug in which the HTTPReadPostName (see page 241)() function was failing to convert the field name from URL encoding to plain-text. If the browser posted, for example, a field named "Stock Remaining", it would have been incorrectly returned from HTTPReadPostName (see page 241)() as "Stock+Remaining".
11. In Stack 5.10, any new values you saved into AppConfig via the Network Configuration demo web page would have been mishandled for WiFi projects. HTTPPostConfig (see page 88)() in CustomHTTPApp.c of the TCPIP WiFi Demo App and TCPIP lperf Demo App projects were corrected so that they now write a magic 0x61 marker into EEPROM/SPI Flash address 0x0000 to indicate that the AppConfig structure is valid in EEPROM/SPI Flash. This prevents the InitAppConfig() function in MainDemo.c from restoring the default settings when changing the values through the Network Configuration page.
12. For WiFi projects, a Gratuitous ARP Work-around was implemented to work around cases where access points send broadcast messages at data rates that the ZG2100 cannot listen (see page 169) to. The define `USE_GRATUITOUS_ARP` (in TCPIPConfig.h) turns this feature on or off.

#### Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenale it's DHCP server.
3. HI-TECH PICC-18 compilers are not supported in this release. The supplied HI-TECH PICC-18 MPLAB projects usually will not compile and/or link.
4. ENC624J600 PSP modes 2, 4, 6, and 10 do not work at this time. Some Parallel Bit Bang modes may not work either. Some minor firmware changes are needed.
5. TCPIP ENCX24J600 Demo App-C18.mcp project does not compile by default using MPLAB C Compiler for PIC18 MCUs (C18) version 3.34. There is not quite enough program memory available on the PIC18F97J60 for the large number of selected stack features to allow linking. To get this project to compile, turn on compiler optimizations or disable one of the modules in TCPIPConfig.h (ex: comment out `STACK_USE_DYNAMICDNS_CLIENT`).

\*\*\*\*\*

v5.10 29 July 2009

\*\*\*\*\*

## Changes:

1. Added SSL capability to the Telnet (see page 481) server. If STACK\_USE\_SSL\_SERVER is defined, the Telnet (see page 481) server will now listen (see page 169) on port 992 for SSL secured connections. If you do not have a telnet client, you can use an SSL proxy, such as stunnel (<http://www.stunnel.org/>) to add SSL/TLS support to any telnet client.
2. Moved a number of string pointer tables in the HTTP.c, HTTP2.c, FTP.c, and DynDNS.c files to allocate in ROM instead of RAM. This reduces around 120 bytes of RAM usage in the HTTP2 server when compiled for the PIC18 or PIC24/dsPIC platforms. The gains are even greater on PIC32 platforms.
3. Added redefinition of SPIRAM\*(), SPIFlash\*(), and XEEPROM\*() functions so that when compiled and used without proper HardwareProfile.h definitions, a more descriptive linker error will be generated instead of a mysterious symbol not found error.
4. Added several new APIs:
  - ExtractURLFields (see page 211)() in Helpers.c. This function provides an easy means of parsing an URL string and extracting the protocol, hostname, port, file path, etc. Currently, this function is commented out to save code space as no stack modules require it. However, it should work correctly if you simply uncomment it (remove the #if 0...#endif around it).
  - strchr (see page 219)() in Helpers.c. Finds the first occurrence of a character within a string but limited to a maximum length before giving up.
  - TCPPeek (see page 453)() and TCPPeekArray (see page 453)() in TCP.c. Reads from a TCP socket's RX FIFO buffer without removing the data from the stream.
  - TCPClose (see page 441)() in TCP.c. Disconnects a socket from the remote node (if connected) and closes the socket handle, including for server type sockets. This function is identical to the TCPDisconnect (see page 442)() API except for the handling of server sockets. TCPDisconnect (see page 442)() returns server sockets to the listening state and maintains the socket handle. TCPClose (see page 441)() closes the socket and frees all associated resources, invalidating the socket handle.
5. Updated the DHCP client module:
  - Modified so that it wouldn't attempt to transmit DHCP Discover packets when the MAC layer reports no link (MACIsLinked() == FALSE). This avoids main() while(1) loop performance degradation when you unplug the Ethernet cable or lose association to your access point.
  - Added capability of performing DHCP discovers and requests without setting the BOOTP broadcast flag. Now, the DHCP client module will start up and attempt to obtain an IP address with the broadcast flag set, but if it fails the next DHCP retry will attempt to obtain the IP address with the broadcast flag cleared. The flag will toggle back and forth between unicast mode and broadcast mode if no DHCP server responds. This feature improves compatibility with certain DHCP servers and WiFi access points.
  - Added several new APIs including DHCPInit(), DHCPEnabled(), DHCPStateChanged(), DHCPBound(), and DHCPisServerDetected().
  - Removed the DHCPFlags DHCP\_CLIENT\_FLAGS global variable. Use the above named APIs now to get equivalent functionality.
  - Removed the DHCPBindCount global variable. To detect if the DHCP state has changed, poll the new DHCPStateChanged() function.
  - Removed the DHCPReset() API. To perform this operation, now call the DHCPInit() API. Use 0x00 for the vInterface parameter.
6. Removed deprecated TickGetDiff() macro. To get a tick difference, just subtract the two values in-line. This macro was removed because it promoted confusing code. Ex: a-b is different from b-a. However, it was not contextually obvious which of the two was returned when TickGetDiff(a, b) was called.
7. Added PIC32MX460F512L USB and dsPIC33FJ256GP710 PIM support to the Explorer 16 hardware profile for the TCPIP WiFi Demo App and WiFi IPerf App projects.
8. Added all files needed for SSL (assuming the crypto libraries are present) to the TCPIP WiFi Demo App-C30 and TCPIP WiFi Demo App-C32 projects.

9. Converted TCPIP Demo App, TCPIP WebVend App, Internet Radio App, and Internet Bootloader App MPLAB Build Directory Policy to compile in the project folder instead of the source folder. This reduces the dependencies on the MPLAB project include path and allows new projects to be created by copying one of the pre-existing folders (ex: copy "TCPIP Demo App" to "My App") without having problems including the wrong HardwareProfile.h and TCPIPConfig.h files.
10. Changed EEPROM/SPI Flash AppConfig record valid flag from 0x60 to 0x61 in the TCPIP WiFi Demo App and WiFi Iperf App projects. This will force the various EEPROM settings to get erased when switching between Ethernet and WiFi projects. This is done since the AppConfig structure changes when using WiFi (SSID string is added).
11. The Wifi Iperf App and TCPIP WiFi Demo App projects have been optimized for better performance.

#### Fixes:

1. Fixed a TCPDisconnect (see page 442)() API bug in which the last few bytes of data (up to the TCP socket's TX FIFO size less 532 bytes) was not transmitted and no FIN was sent out if the TX FIFO was full of data when TCPDisconnect (see page 442)() was called. This problem could have only occurred for TCP sockets with a large TX FIFO (>=532 bytes). This problem could have been observed in stack version 5.00's "TCPIP Demo App-C32 EXPLORER\_16 32MX360F512L ENC624J600 PSP 9.hex" precompiled application, among others, if you connected to the TCPPerformanceTest.c module and then attempted to simultaneously access the web server. The web server was returning data very slowly and failing to send the last parts of each file requested by the browser.
2. Eliminated a potential buffer overflow vulnerability from the HTTPHeaderParseContentLength (see page 251)() function in HTTP2.c. If an oversized or malformed Content-Length header is sent from the web client, the function will now gracefully fail by returning an HTTP 400 Bad Request error page. Thanks go to Mark Philipp for identifying this error and suggesting a solution.
3. Fixed a TCPOpen (see page 451)() problem in which the stack would continuously flood the network with nearly back-to-back ARP query packets if a client socket was created that specified a non-reachable remote IP address (ex: local gateway was offline, or for destinations on the same subnet, the actual remote node was offline). This problem would occur only after a few minutes (<10) had passed since the PIC was last reset. Thanks go to Sergey of DPS TELECOM for reporting this problem.
4. Fixed linking problem with BigInt\_helpers.S (PIC24/dsPIC only) when targeting a PIC with more than 8KB of RAM. The interface registers (\_iA, \_xA, \_iB, \_xB, \_iR, and \_wC) are now forced into near RAM.
5. Cleaned up some uninitialized variable warnings in SNMP.c.
6. Fixed a sequence variable traversal bug in SNMP.c.
7. Cleaned up a large number of unsigned integer to signed integer comparison warnings produced by the MPLAB C Compiler for PIC18 MCUs (C18) version 3.32. With earlier versions of this compiler, these warnings would only be generated as messages, so they did not get displayed by default.
8. Some ENCX24J600 parallel bit bang modes work now. PSP Mode 5 indirect has been tested.
9. SSL client and server capabilities now work when using the ZeroG ZG2100M WiFi interface. In the 5.00 stack release, attempting to enable the STACK\_USE\_SSL\_CLIENT or STACK\_USE\_SSL\_SERVER TCPIPConfig.h options with this network controller would have resulted in an error trap. If an LCD was present, the LCD would display "encRdPtrRAWId = encWrPtrRAWId" when the error occurred.
10. The WiFi Iperf App demo locked up when an invalid command was entered at the serial port console. This is now fixed.
11. The WiFi Iperf App demo locked up when running with a PIC32 if iwconfig was typed at the serial port console. This is now fixed.
12. The Wifi Iperf App demo, when running on the PIC24 and PIC32, and compiled with the -Os option (min code size optimization), did not work. This is now fixed.
13. Change a lot of BerkeleyAPI.c internals. This may fix a number of BSD API problems.
14. Fix a problem with SNMP variables being inaccessible with certain unique PEN numbers.

#### Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenale it's DHCP server.
3. HI-TECH PICC-18 compilers are not supported in this release. The supplied HI-TECH PICC-18 MPLAB projects usually

will not compile and/or link.

4. ENC624J600 PSP modes 2, 4, 6, and 10 do not work at this time. Some Parallel Bit Bang modes may not work either. Some minor firmware changes are needed.
5. SSL client code doesn't work with ENC424J600/624J600 devices. The remote server terminates the connection reporting a bad record MAC (Message Authentication (see page 84) Code). The SSL client does work with other controllers.

\*\*\*\*\*

v5.00 27 April 2009

\*\*\*\*\*

#### Changes:

1. Added ZeroG ZG2100 802.11 WiFi controller support. The new TCPIP WiFi Demo App and WiFi Iperf App projects have been added, which default to using this controller.
2. Added Microchip ENC424J600/624J600 10/100 Ethernet controller support. Support for this controller is provided by the new ENCX24J600.c/.h files which perform the same role as the ENC28J60.c/.h or ETH97J60.c/.h files. Precompiled .hex files for the ENC624J600 controller require the use of the new Fast 100Mbps Ethernet PICTail Plus daughter card (AC164132). This product is not available at the time of the 5.00 TCP/IP stack release. However, it is anticipated to be available for purchase on [www.microchipdirect.com](http://www.microchipdirect.com) in CQ3 2009.
3. Significantly updated the Internet Radio App project. Previously, radio stations were hard coded into program memory at compile time. Now, a dynamic Shoutcast directory client has been implemented which allows retrieval of radio stations at run time, offering endless stations you can tune into. The web pages for the radio have also been updated to allow control and status reporting of the board from a web browser.
4. Update SNMP Server (Agent) module to support SNMPv2C. The default Demo App web pages now include an SNMP reconfiguration capability to set the read and write community strings.
5. Added ICMPSendPingToHost (see page 259)() and ICMPSendPingToHostROM (see page 261)() APIs to ICMP (ping) client module. These two APIs are available only when STACK\_USE\_ICMP\_CLIENT and STACK\_USE\_DNS is defined in TCPIPConfig.h. These functions allow ping of DNS hostnames directly without the need for the application to convert the hostname to an IP address first by manually calling the DNS client APIs. With this addition, the PingDemo.c file was updated to ping the hostname "ww1.microchip.com" instead of a static IP address. Previously, the PingDemo (see page 97) would stop working a couple of months after the stack was released, due to the IP address of the www.microchip.com server dynamically changing. If the DNS module is not enabled, the ping demo will instead ping the local gateway IP address instead of ww1.microchip.com.
6. Updated TCPPerformanceTest.c code. The previous version would generate incorrect speed calculations at high data rates (ex: >1Mbyte/sec).
7. Added multiple connection support to Telnet (see page 481) server example module. To allow multiple connections, define MAX\_SIMULTANEOUS\_CONNECTIONS in Telnet.c greater than 1 and create an equal number of TCP\_PURPOSE\_TELNET type TCP sockets in the TCPSocketInitializer[] definition in TCPIPConfig.h.
8. Added more randomness to the local port selection when opening a client-mode TCP socket. This reduces the risk of reusing a previously used port number if the user power cycles the device.
9. Updated XEE\* SPI EEPROM API functions. Writes are no longer required to start on an EEPROM page boundary, and writes can now be arbitrarily long without having to call XEEEndWrite() at each page boundary. Additionally, the XEEWriteArray() API has been added, which performs a similar operation to the SPIFlashWriteArray() API (but with no special erase cases to worry about).
10. Decoupled AppConfig storage in external SPI EEPROM or SPI Flash option from MPFS\_USE\_EEPROM and MPFS\_USE\_SPI\_FLASH options. MainDemo.c will now save the AppConfig structure in external non-volatile memory, even if MPFS is unused (no HTTP or SNMP server modules enabled) or MPFS is using internal Flash program memory to store web pages/bib information. This change also allows the XEE\*() and SPIFlash\*() non-volatile read/write functions to be available at all times (even if MPFS is unused), as long as the appropriate hardware pinout definitions are present in HardwareProfile.h. SPI Flash and SPI EEPROM are no longer mutually exclusive with each other. However, if both are enabled simultaneously, AppConfig will be stored in the EEPROM, not the SPI Flash.
11. Added required SSL files to TCPIP Demo App MPLAB projects. SSL capabilities can now be turned on directly via the STACK\_USE\_SSL\_SERVER and STACK\_USE\_SSL\_CLIENT options in TCPIPConfig.h for these projects, assuming appropriate crypto libraries are installed (SW300052 available from <https://www.microchipdirect.com/>). With this change,

the historical "SSL Demo App" folder has been removed.

13. Updated HardwareProfile.h files. This includes the addition of PIC18 Explorer board support, removal of the PICDEM Z profile, changes to the HI-TECH PICC-18 profiles for newer compilers, among other changes.
14. Added a TCP and UDP performance test measurements table to TCPIP Stack Help (TCPIP Stack Help.chm). Access this from the "Microchip TCP/IP Stack" book, "Stack Performance" page.
15. Updated MPFSlib project (Microchip.MPFS.dll file) so that C18 and C32 output from the MPFS2.exe utility is now identical for MPFS2 images. The generated .c file is now compatible with both C18 and C32 compilers simultaneously. Previously, the images generated for C18 would compile successfully for C32 projects, but would potentially operate incorrectly when compiler optimizations were turned on. Images generated for C32 would compile successfully and work on C18 projects, but the C18 compiler would take a very long time to process the file each time you rebuilt your MPLAB project. Now, the image generated for C18 matches the image generated for C32 and it will compile fast and work correctly on both platforms, even with compiler optimizations turned on.
16. Added schematics and BOMs for the Ethernet PICtail, Ethernet PICtail Plus, Fast 100Mbps Ethernet PICtail Plus, Internet Radio, PICDEM.net 2, and ZeroG ZG2100M PICtail development boards to the "MicrochipTCPIP StackDemo Board Files" folder.

#### Fixes:

1. Fixed a denial of service vulnerability in the NBNSGetName (see page 285)() function of the NBNS.c file. Previously, if a deliberately malformed packet was received, the PIC RAM could have become corrupted. Thanks go to David Talmage for finding this vulnerability.
2. Fixed Timer1 interrupt flag clearing code on PIC32 products. Previously, the Tick.c module was clearing the interrupt flag in an unsafe manner which could have corrupted other interrupt flags in the IFS0 register. Thanks go to Leon van Snippenberg working on the AVIX-RT RTOS for pointing this error out on the Microchip forums.
3. Fixed SNMP up-time variable. Previously the CustomSNMPApp.c module would respond with the number of Tick API ticks that elapsed, not the number of 10ms time slices that elapsed. The SNMP standard uses 10ms as its time base.
4. Fixed BigInt\_helper.asm's \_masBI() and \_masBIROM() functions when the Br parameter's length modulo 4 was equal to 1 or 2. This bug previously caused the BigIntMod() function to sometimes go into an endless calculation loop on PIC18 products when using the SSL libraries and certain combinations of modulus data and length were used. Thanks go to Vasil Stoianov on the Microchip Ethernet forum for running into this defect and reporting it.
5. Fixed SSLSessionNew (see page 425)() so that it wouldn't "lose" SSL sessions after waiting a few hours. This would previously make it impossible to make new SSL connections after a while, but then after a few more hours, the sessions would become free again. Thanks go to Jim Stephens for identifying this issue and finding the solution.
6. Fixed an SSL 2.0 antique client hello record length calculation bug occurring when a received record was > 255 bytes.
7. Added retransmission capability to SendNotification (see page 110)() function in CustomSNMPApp.c. Previously, if an SNMP trap were sent, but the initial ARP query or response was lost on the network, the SendNotification (see page 110)() code would have deadlocked, and suppressed all future transmission of SNMP traps.
8. Fixed DNS client timeout if the DNS server is unable to be ARPed. Previously, the DNS client would retry ARPing the DNS server indefinitely if it was offline. Now, the DNS client will correctly abort if too many attempts to ARP the DNS server fail. Thanks go to Phil "andersop" on the Microchip Ethernet forum for identifying this error.
9. Suppressed transmission of a TCP RST packet to an unknown IP or MAC address if the TCPDisconnect (see page 442)() function was called on a client mode socket that was not finished with ARP or DNS resolution yet. Thanks go to Phil "andersop" on the Microchip Ethernet forum for pointing this behavior out.
10. Fixed TCP socket from disconnecting if the remote receive window was zero and TCPFlush (see page 446)() was still called. Thanks go to Bob Topper for identifying this issue and suggesting a solution.
11. Fixed Tick.c module returning incorrect values when TickGet (see page 512)() or other API was used with compiler optimizations turned on. Wrong values were observed when using MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs version 3.12.
12. Fixed a number of SPI communications problems that could occur when compiler optimizations were turned on. The ENC28J60 was observed to not work correctly on the dsPIC33FJ256GP710 processor when compiled with MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs version 3.12.
13. Fixed possible MPFS2 error when using an ASM30 .s image where MPFS\_Start would be read using the wrong PSVPAG setting. You must rebuild your MPFS2 image file (ex: MPFSImg2.s) with this stack version's MPFS2.exe utility to get this correction applied.

#### Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will

likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.

2. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenale it's DHCP server.
3. HI-TECH PICC-18 compilers are not supported in this release. The supplied HI-TECH PICC-18 MPLAB projects usually will not compile and/or link.
4. ENC624J600 PSP modes 2, 4, 6, and 10 do not work at this time. Parallel Bit Bang mode does not work either. Some minor firmware changes are needed.

\*\*\*\*\*

v4.55 10 November 2008

\*\*\*\*\*

SSL Note: RSA.c and ARCFOUR.c have not changed between the 4.50 and 4.55 releases. Although the precompiled SSL Demo App .hex files will differ, you can continue to use the previous TCP/IP Stack v4.50 Encryption Add-on with this 4.55 stack version.

Changes:

1. Added DNS client support for a secondary DNS server address. Previously, the AppConfig.SecondaryDNSServer setting was unused. Now, the DNS client module will automatically swap the AppConfig.PrimaryDNSServer and AppConfig.SecondaryDNSServer values after any DNS query timeout (or ARP timeout for the DNS server) and attempt the query with the alternative server. If AppConfig.SecondaryDNSServer is disabled by setting it to the IP address 0.0.0.0, the DNS client will only use the AppConfig.PrimaryDNSServer value and never swap the values. With this change, the DHCP client was also updated. If the DHCP server does not specify a secondary DNS server, then the DHCP client will now set the AppConfig.SecondaryDNSServer value to 0.0.0.0. Previously, it would change the AppConfig.SecondaryDNSServer setting only if the remote DHCP server offered a secondary DNS server.

Fixes:

1. Updated Internet Bootloader App project to correctly detect if the configuration bits are being changed or not. Previously, the bootloader always thought the configuration bits were being changed and thus had to always erase the last Flash page (largest memory address) twice for each firmware update. This did not cause any functional problems or specification violations, but it would decrease the effective Flash endurance of the last page.
2. Fixed a TCP socket memory corruption bug that would occur if TCPGetRemoteInfo (see page 447)() API was called twice with different socket handles without an intermediate call to any other TCP API that takes a TCP\_SOCKET (see page 462) input. Thanks go to Bob Topper for identifying this problem and suggesting a solution.
3. Fixed the UDPsGetReady (see page 523)() function so that it returns the number of bytes remaining in the packet based on the current read location. This is the same behavior as stack versions 4.18 and earlier. In stack versions 4.50 and 4.51, the UDPsGetReady (see page 523)() function would always return the total number of bytes in the current packet, regardless of how many bytes the read pointer had been advanced through the UDPGet (see page 522)() and UDPGetArray (see page 523)() functions. Thanks go to Bob Topper for identifying this problem and suggesting a solution.
4. Fixed demo admin web page in TCPIP Demo App project so that the last byte of the MAC address can be changed, independent of the format it was entered by the user.
5. Fixed a buffer overflow bug that would occur when using the SSL server during hashing of the server certificate for the initial handshake. This error previously caused several bytes of random variables elsewhere in the project to get overwritten for each SSL connection.
6. BSD sockets API was updated to fix some issues.
7. LCDBlocking.c was updated to relax start up timing. This timing fix is specifically needed to support Explorer 16 boards with a Truly TSB1G7000 display (Novatek NT7603H controller).
8. Removed four uses of Arial Black font in MPFS2.exe utility. On some rare PC configurations, the use of this font caused the executable to not run.

## Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenale it's DHCP server.
3. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC\_EXPLORER + Ethernet PICtail.
4. MAC.h RXSIZE precompiler test for proper range doesn't work. This is not a functional problem, just a compile-time configuration test. Ensure that you don't over allocate TCP\_ETH\_RAM\_SIZE or MAX\_HTTP\_CONNECTIONS.
5. HI-TECH PICC-18 STD 9.51PL1 cannot compile DynDNS.c. It raises an "inconsistent type" error while trying to perform a ROM pointer to integer cast. The older 9.50PL3 compiler release is required to compile this file.
6. HI-TECH PICC-18 STD 9.50PL3 does not initialize several static variables correctly on reset. This behavior breaks many stack modules in the TCPIP Demo App and TCPIP WebVend App projects. Additionally, string printing functions do not work correctly, so the supplied "TCPIP Demo App-HITECHPICC18 PICDEMNET2 18F97J60.hex" and "TCPIP WebVend App-HITECHPICC18 PICDEMNET2 18F97J60.hex" files may not correctly print the board's DHCP assigned IP address on the board's LCD (if present) and UART. To avoid these severe problems, use the Microchip MPLAB C Compiler for PIC18 MCUs. A free student edition can be downloaded from <http://www.microchip.com/c18>.

\*\*\*\*\*

v4.51 24 July 2008

\*\*\*\*\*

IMPORTANT NOTE: You must use MPLAB 8.10 or higher to successfully open the MPLAB projects.

SSL Note: RSA.c and ARCFOUR.c have not changed between the 4.50 and 4.51 releases. Although the precompiled SSL Demo App .hex files will differ, you can continue to use the previous TCP/IP Stack v4.50 Encryption Add-on with this 4.51 stack version.

Changes: None. This release includes bug fixes only. It is very important that applications using the ENC28J60 get fix item 7, below.

## Fixes:

1. TCPOpen (see page 451)() was previously failing if you used it to start a connection with a remote hostname, but the DNS module failed to resolve the remote address on the first try. This, for example, would occur if you powered up your board and tried to connect (see page 165) to a remote server before the Ethernet cable was attached. Once the Ethernet cable was attached, the socket would attempt to resolve and connect (see page 165) to a garbage address. The Internet Radio application would sometimes not begin playing the default station upon power up because of this problem.
2. Set SEQ.ACK = 0 for outbound TCP SYN packets. This fixes a connection compatibility problem with certain paranoid TCP/IP stacks that would validate this field even though the ACK flag was clear. This problem would previously cause the Microchip TCP/IP stack to be unable to connect (see page 165) client-mode TCP sockets to certain rare servers/services. Thanks go to Jean LE TUTOUR for finding one of these problem servers.
3. MPFSOpen (see page 275)() and MPFSOpenROM (see page 276)() for MPFS2 could leak a file handle if a name hash matched but no complete file name did. This has been corrected to prevent potential DOS attacks on the HTTP2 web server. Thanks to David Tan on the Microchip Ethernet forum for identifying this issue.
4. Fixed a bug in MPFS2.1 that caused compile errors when MPFS Classic images were generated for ASM30 containing files whose length was either zero or a multiple of 12.
5. Fixed an issue in HTTPPostConfig (see page 88)() that caused it to ignore the flag that was set when invalid IP address input was detected. This issue only affects the example configuration page and only exists in v4.50 (prior versions functioned correctly). Also corrected an issue where user input could potentially overflow into part of the shadow

AppConfig in the same function. Thanks to prinznroll3 on the Microchip Ethernet forums for identifying both of these issues.

6. Implemented Explorer 16 development board 5V LCD errata workaround to LCDBlocking.c. This corrects the A/D converter from returning erratic readings on certain Explorer 16 boards. LCD I/O pins are now continuously driven by the microcontroller instead of going high impedance when idle.
7. Fixed a critical ENC28J60 revision B7 errata workaround problem in the ENC28J60.c, MACFlush() function. Previously, the code was checking for an EREVID register value of 0x07 for silicon revision B7. This was incorrect. Silicon revision B7 actually has an EREVID value of 0x06. Note that this problem was caused by an incorrect EREVID value published in DS80349A, the B7 silicon errata documentation. Make sure to use DS80349B or later.

#### Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenables its DHCP server.
3. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC\_EXPLORER + Ethernet PICtail.
4. MAC.h RXSIZE precompiler test for proper range doesn't work. This is not a functional problem, just a compile-time configuration test. Ensure that you don't over allocate TCP\_ETH\_RAM\_SIZE or MAX\_HTTP\_CONNECTIONS.
5. HI-TECH PICC-18 STD 9.51PL1 cannot compile DynDNS.c. It raises an "inconsistent type" error while trying to perform a ROM pointer to integer cast. The older 9.50PL3 compiler release is required to compile this file.
6. HI-TECH PICC-18 STD 9.50PL3 does not initialize several static variables correctly on reset. This behavior breaks many stack modules in the TCPIP Demo App and TCPIP WebVend App projects. Additionally, string printing functions do not work correctly, so the supplied "TCPIP Demo App-HITECHPICC18 PICDEMNET2 18F97J60.hex" and "TCPIP WebVend App-HITECHPICC18 PICDEMNET2 18F97J60.hex" files may not correctly print the board's DHCP assigned IP address on the board's LCD (if present) and UART. To avoid these severe problems, use the Microchip MPLAB C Compiler for PIC18 MCUs. A free student edition can be downloaded from <http://www.microchip.com/c18>.

\*\*\*\*\*

v4.50 02 June 2008

\*\*\*\*\*

**IMPORTANT NOTE:** You must use MPLAB 8.10 or higher to successfully open the MPLAB projects. Also, ensure that the latest C compiler is used. This release was tested against MPLAB C Compiler for PIC18 MCUs version 3.20, MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs version 3.10, MPLAB C Compiler for PIC32 MCUs version 1.01, and HI-TECH PICC-18 version 9.50PL3 (STD). Earlier compilers may not be able to compile this TCP/IP stack release.

#### Changes:

1. Added SSL 3.0 client capabilities, including SMTP over SSL. The SSL modules supports up to 1024-bit RSA handshakes and 128-bit ARCFOUR bulk encryption. This can be demonstrated using the SMTP client. SSL server support is functional, but a key generation utility is not yet provided and support over HTTPS is not yet reliable with all browsers. **IMPORTANT:** Encryption software is covered by US Export Control law, so it is not directly downloadable from the Microchip website. To use the encryption modules, you must order SW300052 from microchipDIRECT [ <https://www.microchipdirect.com/> ] and install the required libraries.
2. Added Berkeley Sockets (see page 146) Distribution (BSD) API translation layer. You can now call the well know Berkeley APIs instead of or in addition to the Microchip specific APIs. To use this new functionality, define STACK\_USE\_BERKELEY\_API and configure BSD\_SOCKET\_COUNT in TCPIPConfig.h. Three new source code demos are provided to demonstrate this API: BerkeleyTCPClientDemo.c, BerkeleyTCPServerDemo.c, and BerkeleyUDPClientDemo.c. The TCP client demo is identical to the GenericTCPClient.c demo, but implemented using Berkeley Sockets (see page 146). The UDP client demo is similarly identical to the SNTP.c client. The TCP server

demo listens on TCP port 9764 and will echo any traffic received back to the sender. It allows up to 3 simultaneous connections when there are an adequate number of sockets defined in the TCPSocketInitializer[] array in TCPIPConfig.h.

3. Added support for Dynamic DNS services. See the Dynamic DNS Client module in the TCP/IP Stack Help for details. Presently, dyndns.org, dyndns.com, no-ip.com, and dns-o-matic.com are supported.
4. Added the Microchip TCP/IP Configuration Wizard to the Utilities folder, facilitating easier configuration of the TCP/IP Stack through a graphical application.
5. Restructured TCPIPConfig.h to remove rule-enforcement logic, placing the removed sections in TCPIP.h. Many other project structure changes were also made to clean up the general distribution appearance.
6. Increased DHCP Server default lease duration to 60 seconds instead of 15 seconds. Some computers were losing their IP lease before performing a renew operation with only a 15 second lease.
7. Removed CLOCK\_FREQ, INSTR\_FREQ, and PERIPHERAL\_FREQ macro definitions. GetSystemClock(), GetInstructionClock(), and GetPeripheralClock() now return these respective values. This change was made for compatibility with other Microchip software libraries.
8. Added TCP Fast Retransmission capability. Whenever three duplicate ACK packets arrive, the stack will now immediately perform a retransmit operation. This greatly improves recovery latency whenever the network loses a packet for applications that stream TX data using TCP.
9. Improved TCP Keep Alive mechanism to automatically close TCP sockets which do not receive any keep-alive responses for TCP\_MAX\_UNACKED\_KEEP\_ALIVES (see page 477) (default 6) times. This means that, by default, any connection that catastrophically breaks without notifying us (ex: user unplugs cable, Internet connection goes down, etc.) will time out and automatically close after 60 seconds (TCP\_MAX\_UNACKED\_KEEP\_ALIVES \* TCP\_KEEP\_ALIVE\_TIMEOUT (see page 476)). Server oriented sockets will return to the listening state. Client oriented sockets will close, but the TCP\_SOCKET (see page 462) handle will continue to remain valid until the application calls TCPDisconnect (see page 442)(). Applications can check if the socket became disconnected and reset by calling TCPWasReset (see page 458)() or TCPIsConnected (see page 449)(). Note that this keep alive implementation will only close sockets that are broken (remote node is not responding to TCP requests). It will not close or otherwise interfere with idle connections in which the application is not transmitting or receiving data and wishes to keep the connection open.
10. Added a TCP RX SYN queue of depth TCP\_SYN\_QUEUE\_MAX\_ENTRIES (see page 480) (default 3). This queue automatically saves incoming SYN packets destined for a local server port which is already connected to a different client. When the client disconnects, the SYN data is pulled out of the queue and the socket immediately attempts to connect (see page 165) to the next client. This improves connect (see page 165) time performance since the remote client no longer has to retransmit the SYN request if it was unserviceable the first time around. This is most apparent with the HTTP/HTTP2 servers which previously performed poorly with certain modern web browsers which attempt to open many simultaneous connections to the web server, such as Mozilla Firefox 3 beta 5 and Apple Safari 3.1. Entries in the queue automatically time out after TCP\_SYN\_QUEUE\_TIMEOUT (see page 480) (default 3 seconds) so as to prevent the queue from filling up permanently if several connection requests arrive for a service that is in use and will not be available for an extended period.
11. Modified the structure of the MPFS2 FAT (now known as MPFS2.1) to include name hashes first. This speeds up opening files by 25%, and makes opening index files nearly instant.
12. Updated the MPFS2 Utility. MPFS2.1 now supports the new FAT structure and provides a cleaner interface. It also writes images to disk as they are created, which eliminates the IndexOutOfBounds exceptions some users had reported. Finally, uploads are now truly multi-threaded.
13. Source code to the MPFS2.exe PC utility is now released. Find it in the Microchip Solutions\MicrochipTCPIP StackUtilitiesSource\MPFS21 folder. This project is designed to compile with Microsoft Visual C# 2008 Express Edition.
14. Added support for SST25VFxxx serial flash parts in 2, 4, 8, 16, and 32Mbit densities. These parts can be used to replace EEPROMs for storing MPFS images (both versions) and custom data.
15. Added HTTPReadPostName (see page 241), HTTPReadPostValue (see page 242), and HTTPReadPostPair (see page 242) functions to facilitate easier processing of data arriving via POST.
16. Split HTTPAuthenticate API into separate functions: HTTPNeedsAuth (see page 239) and HTTPCheckAuth (see page 235). This function was already split internally, and didn't make sense as a single API.
17. Updated DHCP client to close its UDP socket when idle (bound state) to save a small amount of resources.
18. Removed LED\_IO macro from all hardware profiles because it is not suitable for use on certain hardware platforms that have non-contiguous LEDs or reversed bit ordering. Use the new LED\_GET() and LED\_PUT(val) macros to read and write to all of the LEDs at once.
19. Added Ethernet Hash Table Calculator.exe to the Utilities folder and start menu. This tool will calculate the correct bit that you must set in the EHT0-EHT7 registers on the ENC28J60 and PIC18F97J60 family devices for using the Hash Table RX filter. This is useful only for fixed MAC addresses known at design time. For addresses that are known at run time, use the SetRXHashTableEntry() function in the ENC28J60.c or ETH97J60.c files to set the correct EHT0-EHT7 bit.

Fixes:

1. Fixed a buffer overflow data corruption issue in the FTP module that arises when too many parameters were passed on the command line.
2. Moved TCPWasReset (see page 458) checking in HTTP2 to execute for every socket on every loop. Previously, it would only execute when a socket reconnected, which caused the RX buffer to not resize until after data was received. Some platforms (notably FF2 on Ubuntu) would stall if the initial advertised RX window was too small, and this change corrects that issue.
3. Updated SendSystemReset() and MACInit() initialization routine in ENC28J60.c. Previously, if the ENC28J60 was placed into sleep mode by calling MACPowerDown(), the SendSystemReset() command would not work anymore. This would leave the ENC28J60 in power down if the host PIC was ever reset. SendSystemReset() should work for all conditions with this update. Thanks go to Rob Haverkort on the Microchip Ethernet forum for identifying this problem.
4. Fixed an alignment bug in HTTP2 that caused redirects to fail when the MPFS2 image was stored in Flash program memory. Thanks to Todd Boaz on the Microchip Ethernet forum for identifying this bug, and Chen Qu for posting a solution.
5. Fixed SNTP client from losing accuracy if you called SNTPGetUTCSeconds (see page 368)() 10s of thousands of times since the last server synchronization. Thanks go to "pic123" on the Microchip Ethernet forum for noticing this error.
6. Fixed a TickGet (see page 512)\*() API problem where the returned tick value could be off by 64K ticks occasionally on PIC24, dsPIC30/33, and PIC32 processors. This bug was previously fixed in stack versions 4.13 and 4.16, but it was unintentionally recreated in 4.18 due to PIC32 changes.
7. Fixed UART2TCPBridge module from failing to connect (see page 165) to a remote server when USE\_REMOTE\_TCP\_SERVER was defined.
8. Fixed an issue that prevented SNMP SETs on 16 and 32 bit parts when using MPFS2. Thanks go to Milena K on the Microchip Ethernet forum for identifying this problem.
9. Fixed a rare buffer corruption issue that could occur with UDP if TCP was also enabled.
10. Fixed a Tick rollover error in HTTP2. Thanks go to Paul Bixel on the Microchip Ethernet forum for identifying this problem.
11. Fixed an MPFS2 bug in which an excessive value to MPFS\_SEEK\_REWIND may have failed to return an error. Thanks go to Paul Bixel on the Microchip Ethernet forum for identifying this problem as well.
12. SMTP Client now sends EHLO when using authentication. Previously, the HELO command was used, even with authentication enabled. Using HELO with authentication creates incompatibilities with certain SMTP servers.
13. Improved Internet Bootloader robustness by retransmitting ACKs in response to data retransmissions by the remote sending node. Previously, if an ACK packet was lost before reaching the sending node, the TFTP upload would fail and need to be restarted. Thanks go to "coolvibe" Dave Collier on the Microchip Ethernet forum for identifying this behavior.
14. Fixed TFTP Internet Bootloader from not being accessible from Linux TFTP clients which were setting the IP header "Don't Fragment" flag bit.
15. Changed TCP so that unsent data that is automatically flushed by the TCP\_AUTO\_TRANSMIT\_TIMEOUT\_VAL (see page 474) timer includes the PSH flag. This improves GUI responsiveness for certain applications which rely on this automatic flush feature, such as the UART2TCPBridge module.
16. Fixed TCP socket loss issue which could occur if the TCP TX FIFO size was greater than 536 bytes (TCP\_MAX\_SEG\_SIZE). Before the fix, the socket would have gotten tied up indefinitely performing retransmissions every 1.0 seconds without detecting that the remote node was disconnected.
17. Fixed TCP socket hang issue that would occur if the PIC sent out a FIN and the remote node never responded with a corresponding FIN. The socket would have gotten stuck indefinitely in the TCP\_FIN\_WAIT\_2 state. Thanks go to Mr. Kyle Strickland with AW North Carolina for identifying this bug.
18. Fixed UDPSetRxBuffer (see page 527)() function from not working if it was called before having called UDPGet (see page 522)() or UDPGetArray (see page 523)() at least once.
19. Fixed an offset error of +2 milliseconds being returned from TickConvertToMilliseconds (see page 511)(). Thanks go to Andrés ("saturn") on the Microchip Ethernet forum for finding this error. Note that due to integer truncation during division, this function can be off by 0.2% or so, depending on the value returned by GetPeripheralClock().
20. Updated DelayMs() macro for MPLAB C Compiler for PIC18s to work correctly when a large parameter was given. You should now be able to delay between 0 and 65535 milliseconds across all supported compilers without ending up with an unexpectedly short delay.

#### Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.

3. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC\_EXPLORER + Ethernet PICtail.
4. MAC.h RXSIZE precompiler test for proper range doesn't work. This is not a functional problem, just a compile-time configuration test. Ensure that you don't over allocate TCP\_ETH\_RAM\_SIZE or MAX\_HTTP\_CONNECTIONS.
5. HI-TECH PICC-18 STD 9.51PL1 cannot compile DynDNS.c. It raises an "inconsistent type" error while trying to perform a ROM pointer to integer cast. The older 9.50PL3 compiler release is required to compile this file.
6. HI-TECH PICC-18 STD 9.50PL3 does not initialize several static variables correctly on reset. This behavior breaks many stack modules in the TCPIP Demo App and TCPIP WebVend App projects. Additionally, string printing functions do not work correctly, so the supplied "TCPIP Demo App-HITECHPICC18 PICDEMNET2 18F97J60.hex" and "TCPIP WebVend App-HITECHPICC18 PICDEMNET2 18F97J60.hex" files may not correctly print the board's DHCP assigned IP address on the board's LCD (if present) and UART. To avoid these severe problems, use the Microchip MPLAB C Compiler for PIC18 MCUs. A free student edition can be downloaded from <http://www.microchip.com/c18>.

\*\*\*\*\*

v4.18 28 November 2007

\*\*\*\*\*

Changes:

1. Added C32 and PIC32MX support. Some things were cleaned up in the process.
2. Removed linker scripts from C30 MPLAB projects. MPLAB IDE 8.00 can automatically select the correct linker script for 16-bit and 32-bit products.
3. Updated TCPPerformanceTest.c module. Now it automatically calculates the TX throughput and displays it for you. Also, there is now an RX throughput testing mode, which listens on a separate TCP socket (port 9763) when a TCP socket of type TCP\_PURPOSE\_TCP\_PERFORMANCE\_RX is allocated in TCPIPConfig.h. The RX socket is by default not enabled to save memory, so you must create a TCP\_PURPOSE\_TCP\_PERFORMANCE\_RX socket in TCPIPConfig.h and ensure that enough memory is allocated to accommodate it to test the RX performance test. When connected to port 9763, send a large amount of data and the PIC microcontroller will send back a count of how many bytes were received per second.
4. UDPPerformanceTest.c module now transmits 1024 packets on start up and then stops to prevent continually broadcast flooding your network. To transmit more packets after 1024 is reached, hold down BUTTON3 (left-most button on most boards).
5. Significantly improved the speed of the MD5 and SHA-1 functions. Gains for the 8-bit compilers were 50-75%, while 16-bit parts saw more modest improvements (~10%).
6. Reimplemented TCP\_CLOSE\_WAIT TCP state ("CLOSE WAIT" in RFC793). Now, TCP sockets that receive a FIN from the remote node will hold off transmitting a FIN back to the remote node until the TCP\_CLOSE\_WAIT\_TIMEOUT (see page 474) (defined at the top of TCP.c) elapses or immediately when the application calls the TCPDisconnect (see page 442)() function. This makes it possible for the application to transmit a response back to the remote node before the socket becomes closed on our end. Similarly, it simplifies application usage of the last RX bytes received as these bytes are now assured to still be in the RX FIFO for at least TCP\_CLOSE\_WAIT\_TIMEOUT (see page 474) seconds. TCP\_CLOSE\_WAIT\_TIMEOUT (see page 474) defaults to 200ms in this stack version.
7. Pushed the SNTP query on failure timeout up some. It was ~14 seconds and is now ~20 seconds.
8. Added TFTPOpenROMFile (see page 494)() API to complement TFTPOpenFile (see page 494)() when using PIC18 products.
9. Added a fourth parameter to newAJAXCommand() in mchp.js, allowing data to be POSTed along with the AJAX request.
10. Deprecated the TCP Loopback functions, which includes TCPOpenLoopback, TCPCLoseLoopback, TCPIsLoopback, TCPInject, and TCPSteal. These functions were added in 4.10 for future SSL support, but have since become unnecessary. They are of limited usefulness, and so are being removed to save code space. The functions are still available in this version, but will be removed in the next release.
11. Added SMTPClient.ServerPort (see page 94) field to the SMTP API. This allows the remote server port number to be specified dynamically at run time instead of being hard coded to the SMTP\_PORT (see page 308) value defined at the top of SMTP.c. SMTP\_PORT (see page 308) is now only a default.
12. Added web interface to the SMTP module in the TCPIP Demo App applications. You can now configure the SMTP

module and send emails directly from within your web browser. The HTTPPostEmail (see page 89)() function in CustomHTTPApp.c also demonstrates how to send MIME encoded attachments in emails. The default demo will send button states, LED states, and the current potentiometer reading as a CSV file attached to the email. 13.Changed SMTPDemo (see page 92)() in MainDemo.c to trigger on BUTTON2 and BUTTON3 simultaneously held down instead of BUTTON0 only.

#### Fixes:

1. Fixed an ENC28J60.c MACGetArray() bug which would overwrite one byte of memory at address 0xFFFFFFFF if you provided NULL for the destination address pointer.
2. Fixed an MPFS2.c MPFSGet (see page 269)() bug which would overwrite memory address 0x00000000 if a NULL pointer was provided as the destination.
3. Fixed a bug in the HTTP2 server accessing incorrect sockets if an inadequate number of sockets were available on POR.
4. Fixed Internet Bootloader project from failing with a timeout if an ARP packet arrived during the Erase/Write operation.
5. Fixed DHCP client RFC non-compliance where it would send the ciaddr field in the initial SELECTING state. Also, in the RENEWING state, the Requested IP Address (see page 141) option was being sent, which is illegal. These changes may fix compatibility problems with certain DHCP servers.
6. Fixed TFTP Client's TFTPcloseFile (see page 488)() function from sending data using a wrong UDP socket if StackTsk() was called after TFTPisFileOpened (see page 490)() was last called.
7. Added two zero bytes to the ICMP echo request payload to improve compatibility with some buggy NAT routers.

#### Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenale it's DHCP server.
3. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC\_EXPLORER + Ethernet PICtail.
4. MAC.h RXSIZE precompiler test for proper range doesn't work. This is not a functional problem, just a compile-time configuration test. Ensure that you don't over allocate TCP\_ETH\_RAM\_SIZE or MAX\_HTTP\_CONNECTIONS.

\*\*\*\*\*

v4.16 06 November 2007

\*\*\*\*\*

#### Changes:

1. Added Internet Radio application. This is a TCP client application which downloads streaming MP3 audio from a Shoutcast server and then plays it back to stereo earphones via a VLSI VS1011 audio decoder.
2. Added SPIRAM.c module. This module is intended for interfacing to an AMI Semiconductor N256S0830HDA SPI RAM chip. The TCP module can now interface directly to this SPIRAM module to store TCP socket FIFO buffers and other TCB data in the external RAM.
3. Added TCP\_OPTIMIZE\_FOR\_SIZE (see page 477) compile time configuration macro to TCP.c file. When optimizing for small code size, the TCP module ROM footprint shrinks up to 6KB, but performance may slow down on some processors (namely PIC18s, where the penalty is approximately 15%).
4. Added USE\_EEPROM\_25LC1024 compile time configuration macro to TCPIPConfig.h. Enable this definition if you are storing your MPFS[2] on a 1Mbit 25LC1024 or similar EEPROM device that uses 24-bit addressing and a 256 byte write page size.
5. Changed LCDBlocking.c module initialization code. It should now be possible to use 4-bit mode on certain "unusual" LCD

controllers, like the Samsung S6A0032. Most PICDEM.net 2 and Explorer 16 boards use an LCD with this controller.

6. SNTP client now attempts to requery the SNTP server about every 14 seconds if the last query attempt fails. This allows the internal time value to become valid quickly should the board be powered up before an Ethernet cable is attached or if the DHCP client doesn't obtain an IP address quickly enough. Previously, it would take up to 10 minutes after plugging the Ethernet cable in to get a correct time value from the SNTP server.
7. Added `UDP_USE_TX_CHECKSUM` compile time configuration macro to `TCPIPConfig.h`. When enabled, all UDP packets will have a correct UDP checksum computed and inserted into the UDP header of outbound packets. If you do not define this macro, the UDP checksum will be disabled (left as `0x0000`), which is how previous stack versions operated. Note that enabling checksum generation cuts your maximum UDP TX throughput by nearly half due to the required computations.
8. Substantially changed TCP socket RX and TX FIFO allocation. Now, sockets can be stored either in Ethernet RAM, PIC RAM, or external (SPI) RAM. Previously, sockets could only be allocated in Ethernet RAM, which was not scalable.
9. Added `TCPOpen` (see page 451)() API function. This replaces `TCPListen` (see page 451)() and `TCPCConnect` (see page 441)(). `TCPOpen` (see page 451)() supports a large number of options that will make the creation of client mode sockets much easier. You can specify the remote node as a hostname that needs DNS and ARP resolution, an IP address that only needs ARP resolution, or legacy `NODE_INFO` pointer for direct compatibility with the previous `TCPListen` (see page 451)() and `TCPCConnect` (see page 441)() APIs. `TCPOpen` (see page 451)() also supports a socket type parameter which will allow you to use the new TCP socket RAM allocation system.
10. Added TCP Keep Alive mechanism defined by RFC 1122 section 4.2.3.6 to the TCP module. This helps automatically detect lost connections. If the remote node sends back an RST, this immediately closes the lost connection on our end. Currently, no action is taken if the keep alive gets no response. Note that this feature deviates from the standard by defaulting to only 10 seconds instead of over 2 hours. Also deviating from the standard, this feature is enabled by default. To disable it, undefine `TCP_KEEP_ALIVE_TIMEOUT` (see page 476) at the top of `TCP.c`.
11. Moved `TCPPerformanceTest.c` module from default port 12345 to 9762.
12. Moved `UDPPerformanceTest.c` module from default port 12345 to 9, the "discard" protocol port.

#### Fixes:

1. The DHCP client now specifically requests the previous IP address when a DHCP renewal occurs.
2. The SNTP client now correctly maintains time when repetitively calling `SNTPGetUTCSeconds` (see page 368)() between an NTP requery event. Thanks go to Rob Haverkort on the Microchip Ethernet forum for noticing the time value incrementing far faster than it should have.
3. TCP module will not transmit a bunch of unnecessary duplicate ACK packets when data is ready to be transmitted but the remote RX window is zero. This previously didn't cause anything to break, but would waste CPU time and bandwidth sometimes.
4. TCP sockets will no longer automatically close if the remote RX window stays zero for several seconds.
5. Fixed TFTP Internet Bootloader project from corrupting the configuration fuses. Previously, this would result in the Watchdog timer being enabled and causing an unintentional reboot every few minutes with the demo TCP/IP stack.

#### Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft `telnet.exe` utility which is provided with Microsoft Windows.
2. TFTPc module has not been tested with this version.
3. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenale it's DHCP server.
4. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the `HPC_EXPLORER + Ethernet PICtail`.
5. `MAC.h` `RXSIZE` precompiler test for proper range doesn't work. This is not a functional problem, just a compile-time configuration test. Ensure that you don't over allocate `MAX_TCP_SOCKETS`, `TCP_TX_FIFO_SIZE`, `TCP_RX_FIFO_SIZE`, or `MAX_HTTP_CONNECTIONS`.

\*\*\*\*\*

v4.13 02 October 2007

\*\*\*\*\*

## Changes:

1. Added command line support to the MPFS2.exe tool. You can now generate MPFS output files using batch scripts or other console applications.
2. Added dynamic variable parameter capabilities to the MPFS2 utility. To use, add the parameters you wish to pass to the end of the dynamic variable. All parameters are passed as WORD values. (ex: ~myArray(2,5)~ )
3. Added TCPWasReset (see page 458)() API to allow the application layer to be notified if an underlying socket reset has occurred (ex: remote node disconnects, cable is disconnected and times out, user calls TCPDisconnect (see page 442)(), etc.). The reset state is latching, which allows the application layer to detect if a remote node disconnects and a new connection occurs on the same socket before the application can detect the original disconnection through the TCPIsConnected (see page 449)() API.
4. Added a counter to the UDPPerformanceTest module and made it suppress transmission if an Ethernet link is not present.
5. Added TCPIP WebVend App example application to the main stack distribution. This corresponds to three new Microchip Webinars being published on the HTTP2 server usage topic.

## Fixes:

1. Fixed MPFS2.exe PC utility from crashing if you attempt to generate an MPFS classic .bin/.c/.s output file.
2. Fixed RCONbits definition for HPC\_EXPLORER hardware profile when using the HI TECH PICC-18 compiler.
3. Fixed a MPFSGetFilename (see page 271)() bug when using C30 and MPFS2 images stored in program memory. Thanks to Billy Walton on the Microchip Ethernet forum for identifying this issue.
4. Fixed a TCP RX FIFO corruption problem which would occur if the remote node sent more data than could fit in our RX FIFO in a single packet. The GeneticTCPClient.c module was subject to experiencing this problem when connected to www.google.com's servers.
5. Fixed a DHCP client UDP socket leak if you called DHCPDisable() after the DHCP client had already obtained a UDP socket. Thanks go to Matthew Kendall on the Microchip Ethernet forum for identifying this problem.
6. Fixed a SNMP Server module bug testing a string length (with respect to SNMP\_COMMUNITY\_MAX\_LEN (see page 325)) being off by one, resulting in possible memory corruption. Thanks go to Matthew Kendall on the Microchip Ethernet forum for identifying this problem.
7. Cleaned up some C30 compiler warnings related to macro definitions with inadequate parenthesis in them.
8. Fixed HTTP2 module sometimes returning a 501 error instead of a correct web page when being bombarded with new connection requests.
9. Fixed a TickGet (see page 512)\*() API problem where the returned tick value could be off by 64K ticks occasionally on PIC24 and dsPIC processors.
10. Fixed SMTP client module failing to send email when attempting to send an email with a `CC` or `BCC` field that was in ROM while the `To` field was in RAM or visa versa.
11. Fixed TCP module sending an incorrect sequence number in RST packets sent when in the TCP\_SYN\_SENT state and an invalid segment arrives. In prior stack versions, some TCP client applications might take a very long time to recover in the event of a power failure, reset, and subsequent reconnect to a remote server that still thinks the old connection is still active. With this fix, reconnections should be possible almost immediately after a power failure because the correct RST packet will cause the old connection to get closed right away.
12. Fixed a TCP socket leak problem that would occur over if the local PIC called TCPDisconnect (see page 442)() and the remote node didn't send us a correct FIN response. Sockets (see page 146) could previously get lost in the TCP\_FIN\_WAIT\_2 state and wouldn't recover unless the application called TCPDisconnect (see page 442)() a second time with the same socket handle.

## Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. TFTPc module has not been tested with this version.

3. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenables its DHCP server.
4. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC\_EXPLORER + Ethernet PICtail.
5. HI-TECH PICC-18 projects will not correctly set the processor configuration fuses through code using the `__CONFIG()` macro. Ensure that the configuration fuses are manually set correctly via the MPLAB IDE Configuration Bits dialog. This problem has been observed with compiler version 9.50PL3.
6. MAC.h RXSIZE precompiler test for proper range doesn't work. This is not a functional problem, just a compile-time configuration test. Ensure that you don't over allocate `MAX_TCP_SOCKETS`, `TCP_TX_FIFO_SIZE`, `TCP_RX_FIFO_SIZE`, or `MAX_HTTP_CONNECTIONS`.
7. GenericTCPClient (see page 93) example of downloading a web page from `www.google.com` is extremely slow. The default TCP socket has too little RX space to accept (see page 163) a full packet sent from Google's servers, so the remote server must retransmit a lot of data, slowing the transfer down a lot. Making `TCP_RX_FIFO_SIZE` 536 bytes or bigger and correspondingly shrinking `MAX_TCP_SOCKETS` will correct this problem.

\*\*\*\*\*

v4.11 27 August 2007

\*\*\*\*\*

IMPORTANT NOTE: You must use MPLAB 7.62 or higher to successfully open the MPLAB projects.

Changes:

1. Added a Microchip TCP/IP Stack Users' Guide to document the stack features/modules/and APIs and address the stale AN833 documentation. Note that this is a work in progress. Many modules have yet to be documented in the Users' Guide.
2. Added HTTP2 module. This HTTP module includes a whole new API and supreme new features, such as POST support, cookies support, browser authentication support, and more.
3. Added MPFS2 module. This module is required for the new HTTP2 module and performs better while having fewer limitations. Long filenames and folders are now supported.
4. Added a new GUI based MPFS2.exe PC utility. The older MPFSv2.exe GUI application and MPFS.exe command line tool has been retired. The new utility has advanced features, such as MPFS2 file format support, GZIP compress, etc.
5. Added a TFTP bootloader. This is a stand alone project and currently only supports the PIC18F97J60 family of PIC processors with internal Ethernet.
6. Added `UART2TCPBridge.c` file and `STACK_USE_UART2TCP_BRIDGE` option to `TCPIPConfig.h`. This new module acts as a TCP and UART bridge, with a high priority UART interrupt and dedicated UART TX and RX FIFOs for minimum UART latency and maximum performance. By default, the bridge acts as a TCP server and listens on port 9761. The UART baud rate defaults to 19200. The bridge can be reconfigured to act as a TCP client.
7. Added Simple Network Time Protocol (SNTP) client. This module automatically obtains the current time (date) from the Internet. Enable this module by defining `STACK_USE_SNTP_CLIENT` in `TCPIPConfig.h`. Obtain the current time (in seconds since 00:00:00 1970) by calling the `SNTPGetUTCSeconds` (see page 368)() API.
8. Added support functions `Base64Encode` (see page 208)() and `Base64Decode` (see page 208)() in `Helpers.c`. Base 64 is required for the new HTTP2 module, but of general use to many applications.
9. Added SMTP Authentication (see page 84) support to the SMTP Client. To use this, set the `SMTPClient.Username` and `SMTPClient.Password` string pointers to a non-NULL value before calling `SMTPSendMail` (see page 303)(). Applications implementing email transmission capabilities should expose these options to the end-user for configuration. To use SMTP servers that do not support the AUTH LOGIN authentication command, simply leave the `SMTPClient.Username` and `SMTPClient.Password` pointers as their default NULL value.
10. Converted `DHCPDisable()` from a macro to a real function and added the complementary `DHCPEnable()` function. These two functions can be used at run time to dynamically switch between using a static IP address and configuration and DHCP

assigned IP address and configuration. 11.Updated StringToIPAddress (see page 218)() to work more robustly, including the ability to decode host name strings and determine if they contain a valid IP address or not. Also, the complementary ROMStringToIPAddress (see page 218)() function was added. 12.Updated the DNS module. Now, if you give it an IP address string to resolve, it will convert the string to an IP address and immediately return without querying the DNS. 13.Shrunk the advertised TCP Maximum Segment Size from 576 bytes to 528 bytes. This might improve compatibility if your TCP data has to propagate over nodes with small MTUs and you have a correspondingly large TCP RX FIFO defined. 14.Performed some maintenance on the FTP.c file. No significant functionality has been changed, but some potential problems were corrected. 15.Altered Tick.c file and API. Now, the Tick module can operate maximum precision, returning the value of the actual Timer as it is counting, without disturbing the timer count by writing to it or disabling it. Three new APIs were added, TickGetDiv256 (see page 512)(), TickGetDiv64K (see page 513)(), and TickConvertToMilliseconds (see page 511)(). Internally the tick counter is now 48-bits wide and as accurate as your Timer clock source, allowing you to use it as a Real Time Clock. 16.Added PIC24FJ64GA004\_PIM hardware profile. This hardware profile is intended for use with the PIC24FJ64GA004 PIM on the Explorer 16 development board. In this mode, BUTTON2 and BUTTON3 and several of the LEDs do not work correctly due to lack of I/O pins on this device. Also, you cannot have the POT and TEMP jumpers on the PIM bridged because these signals are multiplexed with the SDO1/SDI1 pins needed for the Ethernet PICtail Plus. 17.Removed most ROM APIs when using a 16-bit compiler (C30). PIC24s and dsPICs usually don't need separate ROM functions since the Program Space Visibility feature maps ROM into RAM space. All ROM APIs are still supported, but they are now macros to base RAM APIs. This change saves a couple of kilobytes of code space on PIC24 and dsPICs. 18.Improved MyTCB structure caching. This should reduce TCP packet processing overhead with the ENC28J60 where TCBS are stored in the Ethernet RAM. 19.MAX\_RETRY\_COUNTS TCP configuration option has been renamed to TCP\_MAX\_RETRIES (see page 476). 20.FTP server is no longer enabled by default. HTTP2 now supports POST, so you can upload new webpages through the /mpfsupload page now. FTP required two precious TCP sockets. 21.Began adding hooks for an SSL/TLS transport for secure HTTPS and other future stack modules. Note that these cryptographic modules are not available at this time. Configuration options such as MAX\_SSL\_CONNECTIONS do nothing and should not be modified. 22.Username has changed for all of the modules. Now all modules have a default username of "admin" and password of "microchip". Previously, the FTP and Telnet (see page 481) modules used "ftp" and "telnet" respectively for the usernames.

#### Fixes:

1. Fixed a SendFile() bug in HTTP.c where parsing dynamic cgi files could send garbage back to the web browser sometimes. Thanks go to Matt Watkins on the Microchip Ethernet forum for identifying this issue.
2. Fixed an off by one error in the calculation of RESERVED\_TCP\_MEMORY. Previously, the last TCP socket's RX FIFO would incorrectly overlap with the Ethernet RX buffer, causing incoming packets to occasionally be corrupted or the incoming data on the last socket to get corrupted.
3. Fixed the QWORD\_VAL's dword struct element types. dword.LD and dword.HD were incorrectly defined as WORDs instead of DWORDs. Thanks go to Iñaki Esparza on the Microchip Ethernet forum for identifying this issue.
4. Fixed the incorrect processing of received IP fragments with a non-zero offset. This stack does not support IP packet reconstruction due to the limited amount of available RAM. Thanks go to Iñaki Esparza on the Microchip Ethernet forum for noticing this behavior.
5. Board now only responds to ping requests to our IP address, the directed subnet broadcast address, or the broadcast address of 255.255.255.255. Previously, it would respond to any ping request to any IP address, assuming the MAC address was correct.
6. Fixed a memory corruption/UDP packet loss problem when handling incoming UDP packets. Previously, StackTask() would incorrectly continue processing more packets if it came upon a UDP packet. Thanks go to Iñaki Esparza on the Microchip Ethernet forum for identifying this issue.
7. Fixed the SMTPClient.ROMPointers.Server flag having an inverted meaning. Previously, the SMTP client module would treat the SMTPClient.Server pointer as a ROM pointer if this bit was cleared. In most cases, this would cause the SMTP client to return an error code of 0x8000 when the SMTPClient.SMTPServer (see page 308) address pointer was set.
8. Fixed the DHCP Server module from incorrectly parsing received packets which had a DHCP\_PARAM\_REQUEST\_IP\_ADDRESS option followed by more options. Previously due to the length miscalculation, the parser would enter a random state, depending on the packet's contents. Thanks go to Iñaki Esparza on the Microchip Ethernet forum for identifying this issue.
9. Fixed potential incorrect results when UDPIsGetReady (see page 523)() was called and a previous application did not call UDPDiscard (see page 521)() on an RX packet. Now, StackTsk() calls UDPDiscard (see page 521)() as appropriate to let it know when it's old RX data is being thrown away. This fixes a potential bug in the DHCP Server

module and makes the UDP API more robust. Thanks go to Iñaki Esparza on the Microchip Ethernet forum for identifying the potential DHCP server issue.

10.Fixed a potential ARP bug where the Gateway's MAC address would be returned for an IP address on the local subnet. This unusual case would occur when two application tasks were using the ARP module at the same time and the second application was trying to resolve an IP address off of our subnet. Thanks go to Iñaki Esparza on the Microchip Ethernet forum for pointing this issue out. 11.Fixed an PIC18F97J60 family MAC layer bug where MACGetArray() might not correctly increment the Ethernet read pointer if a NULL pointer was given for the destination. The C compiler might have optimized the function so that it would increment the read pointer one less than it was supposed to. 12.The TCP module now acknowledges TCP Keep-Alive packets which will help prevent connection loss if the remote node fills up our RX FIFO and then our window-update packet gets lost on the network/Internet. In stack version 4.02, a zero-window probe would have been required to restore the communications. 13.Fixed a TCP RX FIFO corruption issue that would occur in (uncommon) circumstances when too many out-of-order segments arrived such that a second "hole" would have been required to accommodate the data. Thanks go to Iñaki Esparza and his eagle eyes on the Microchip Ethernet forum for finding this corner case bug. 14.Inline assembly in the ETH97J60.c file has been modified to accommodate the C18 Extended mode and C18 Auto default storage class. Previously, the Ethernet module would transmit garbage packets when using the C18 parameter stack. 15.Fixed potential buffer overflow in NBNS.c's NBNSGetName (see page 285)() function where an unexpected string length retrieved from the packet could cause random memory corruption. 16.Fixed some potential PIC18F97J60 family Ethernet module transmit lockup conditions that occur on some networks. Previously blocking while() loops would wait indefinitely for the ECON1bit to become clear by hardware, which the hardware might never have done. 17.In MainDemo.c, a call to DelayMs() was being made using a value of 100ms. This was too long for the underlying Delay1KTCYx() C18 function and would result in a shorter than expected delay when compiled with C18. This has been fixed with a loop. Thanks go to Andy123 on the Microchip Ethernet forum for pointing this problem out. 18.Fixed a potential C18 memory overlaying problem in the TickUpdate (see page 514)() function. Previously, the local variable used in this function might have been overlayed on other memory, resulting in random memory corruption as the ISR occurred. 19.The demo AJAX web pages in the TCPIP Demo AppWebPages folder now correctly display and self-refresh in Firefox 2. Previously, it would work in Firefox 1.5 and Microsoft Internet Explorer, but not Firefox 2. Thanks go to "gohsthb" on the Microchip Ethernet forum for identifying this correction. 20.Rewrote the GenericTCPServer.c example to not use an application RAM FIFO for buffering. Since the TCP module implements its own FIFOing, the application has limited need for its own FIFO too. This fixes a previous bug where the GenericTCPServer (see page 96) was not checking the number of incoming bytes with the remaining size available of the App FIFO. This would have previously resulted in a buffer overflow, corrupting the RX data if too much arrived all at once. 21.Fixed a potential MPFS classic inline ASM30 assembly code problem where web pages stored in internal Flash and C30 with optimizations enabled could result in data corruption. 22.Fixed a UDPPut (see page 524)() tracking problem that would result in extra bytes being appended to the end of a packet if the UDPSetTxBuffer (see page 527)() function was used. This previously caused the SNMP module to send some junk data at the end of its packets. 23.Fixed a potential TCP problem where transmitted FIN packets might not get retransmitted properly if the remote node never acknowledged the data that was transmitted just before the FIN was sent. 24.Fixed a NetBIOS Name Service bug where the response packet would sometimes get sent to an incorrect address. It now consistently responds to the unicast MAC/IP address of the NBNS query packet. 25.Added padding to all transmitted DHCP messages to make the minimum UDP payload at least 300 bytes. This fixes compatibility with some older BOOTP relay devices which discard smaller packets. Thanks go to Dave Collier on the Microchip Ethernet forum for pointing this problem out. 26.Substantially shrunk the number of retransmission attempts made in the TCP\_SYN\_RECEIVED state. This improves recovery time when attacked by a SYN flood Denial of Service event. The recovery time is now 7 seconds (3 total packets) instead of 31 seconds (6 total packets) 27.Fixed the possibility of the NetBIOS Name Service module giving out the board's static IP address before a DHCP lease could be obtained. NBNS requests are now only serviced when originating from nodes on the same subnet. 28.Fixed storage of MPFS classic in internal program memory when using the HI-TECH PICC-18 compiler. 29.Substantially revised TCP.c, fixing many TCP bugs and possibly adding new ones. Thanks go to Michael Rubinstein for finding several of these TCP problems. 30.The DNS client module will now time out and return failure if the DNS server cannot be ARPed or does not respond to the DNS query. Each timeout is set to 1 second and 3 total ARP and 3 total DNS query attempts are possible. Previously, it would retry indefinitely, causing the calling application to deadlock.

#### Known Problems:

1. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.

2. TFTPc module has not been tested with this version.
3. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenale it's DHCP server.
4. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC\_EXPLORER + Ethernet PICtail.
5. HI-TECH PICC-18 projects will not correctly set the processor configuration fuses through code using the \_\_CONFIG() macro. Ensure that the configuration fuses are manually set correctly via the MPLAB IDE Configuration Bits dialog. This problem has been observed with compiler version 9.50PL3.

#### Testing and Performance Notes:

1. Make sure to use MPLAB IDE 7.62 or higher with this version. Versions below 7.61 will not work. Version 7.62 has cool new features like C auto-word complete and function parameter tooltips that can be enabled (disabled by default).
2. Testing was done using MPLAB C18 version 3.12, MPLAB C30 version 3.01, and HI-TECH PICC-18 version 9.50PL3. Make sure to upgrade your tools to at least these versions.

\*\*\*\*\*

v4.02 10 April 2007

\*\*\*\*\*

IMPORTANT NOTE: You must use MPLAB 7.41 or higher to successfully open the MPLAB projects. IMPORTANT NOTE2:If an external serial EEPROM memory is used to store AppConfig, it's contents will be invalidated the first time you run this version, restoring the AppConfig defaults. The AppConfig structure has been optimized. IMPORTANT NOTE3:If an external serial EEPROM memory for MPFS, you will need to recreate the MPFS image and program your EEPROM. A 32 bit addressing format is now used.

#### Changes:

1. Implemented TCP RX packet order correction logic. The stack can now accept (see page 163) TCP frames that arrive out-of-order without requiring the remote node to go through a retransmit cycle. This dramatically improves RX performance when communicating over the Internet.
2. UDPOpen (see page 520)() now can handle a NULL pointer for remoteNode. In this case, the broadcast IP/MAC addresses will be used for the remoteNode (destination address of outbound packets).
3. Recreated MPLAB projects for the HI-TECH PICC-18 compiler. These were temporarily absent from 4.00RC. This project works with the PIC18F97J60 with internal Ethernet module, assuming the correct compiler version is present.
4. Moved all the headers around. Most of them are in "Microchip SolutionsMicrochipIncludeTCPIP Stack" now. This change was made to again be more compatible with other (future) Microchip software libraries.
5. New UDPPut (see page 524)() behavior. Now, if space in the Ethernet TX buffer runs out, the packet will not automatically be transmitted. You must call UDPFlush (see page 522)() to cause the packet to be transmitted.
6. Added UDPGetArray (see page 523)(), UDPPutArray (see page 525)(), UDPPutROMArray (see page 525)(), UDPPutString (see page 526)() and UDPPutROMString (see page 526)() user API functions. These functions perform substantially better than calling UDPPut (see page 524)() successively and allow greater application programming flexibility.
7. Changed TCPPutString (see page 456)() and TCPPutROMString (see page 455)() APIs to now return an updated string pointer instead of a count of bytes successfully placed in the TX buffer.
8. Added UDPPerformanceTest.c. By default this module causes UDP packets containing 1024 bytes of application data to be broadcasted on UDP port

12345. Use a packet sniffer, such as Wireshark (<http://www.wireshark.com/>)

to capture and derive stack overhead/UDP TX performance characteristics with this module. Note that this test uses the UDPPutROMArray (see page 525)() function. Applications which use successive calls to UDPPut (see page 524)() will be slower. To enable this module, #define STACK\_USE\_UDP\_PERFORMANCE\_TEST in TCPIPConfig.h.

### 9. Added TCPPerformanceTest.c. By default this module listens on TCP port

12345. When a remote client connects, this server module will be

transmitting the maximum possible amount of application data that it can, given your TCP TX FIFO size. Use a packet sniffer, such as Wireshark (<http://www.wireshark.com/>) to capture and derive stack overhead/TCP TX performance characteristics with this module. Any TCP client can be used, including readily available utilities such as the telnet.exe utility available on Microsoft Windows XP. To use it to connect (see page 165) to the test module, run: "telnet.exe xxx.xxx.xxx.xxx 12345" where xxx.xxx.xxx.xxx is the board's IP address. Note that this test uses the TCPPutROMArray (see page 455)() function. Applications which use successive calls to TCPPut (see page 454)() will be slower. To enable this module, #define STACK\_USE\_TCP\_PERFORMANCE\_TEST in TCPIPConfig.h. 10. Added Reboot.c module. By default, this module listens on UDP port 30304. If the application byte 0x00 arrives on this port, the PIC will reset. This is primarily useful for remote Bootloader entry. #define STACK\_USE\_REBOOT\_SERVER in TCPIPConfig.h to enable this module. Note that since no encrypted challenge/response algorithm is currently implemented, this module is a Denial of Service vulnerability, so it should not be enabled unless there is a specific need for it. 11. Made the TickUpdate (see page 514)() ISR routine execute in the low priority ISR instead of the default high priority ISR. The Microchip TCP/IP stack does not need any interrupts except this low priority timer. 12. Renamed STACK\_USE\_DHCP macro to STACK\_USE\_DHCP\_CLIENT 13. Added STACK\_USE\_MPFS macro. 14. Changed UDPIsPutReady (see page 524)() to return a WORD instead of a BOOL. The WORD is the number of bytes that can be put into the buffer. 15. Changed MACGetArray() to accept (see page 163) a NULL pointer. If NULL, the retrieved data will simply be discarded. This also changes the behavior of UDPGetArray (see page 523)() and TCPGetArray (see page 447)() to match, throwing bytes away if a NULL pointer is given. 16. Added a very simple DHCP Server module. This module has limitations and is useful for a single client only. Its purpose is to allow you to directly connect (see page 165) the board to a standard PC through a crossover cable (no other network nodes attached). The server is coded to automatically disable itself if the DHCP client is also enabled and another DHCP server is detected on the network. This allows both the DHCP server and DHCP client to coexist without any manual reconfiguration. 17. Added DNSResolveROM (see page 180)() function for resolving host names that are stored in program memory, ex: literal strings. 18. Added a TCP automatic transmit/window update timer. It defaults to TCP\_AUTO\_TRANSMIT\_TIMEOUT\_VAL (see page 474) (40ms) after the first get or put operation following the last automatic transmit/window update. This timer enhances performance, especially when streaming data over the Internet where round trip times can be several tens to low hundreds of milliseconds. This also improves application coding flexibility as TCPFlush (see page 446)() need not be called anymore. 19. Added TCP delayed ACKnowledgement timer. This conserves bandwidth by transmitting fewer ACKs and prevents inadvertently influencing remote slow start/collision avoidance and fast retransmit algorithms. 20. Completely rewrote ICMP (ping) server module. It is now much smaller (ROM and RAM), faster, and can handle packets of 576 bytes or larger, if no IP fragmentation occurs. 21. Rewrote StackTsk() stack manager. It is much simpler now. 22. Added TCPFind (see page 443)(), TCPFindArray (see page 443)(), and TCPFindROMArray (see page 445)() user API functions. These functions peek inside a given TCP socket's RX FIFO (without removing anything) and looks for a particular byte or array of bytes. This should greatly simplify the creation of application code whenever variable length fields are used (ex: text strings terminated by \r\n). It supports case insensitive text searching or binary searching, as well as an offset to start searching at. 23. Added TCPGetRxFIFOFree (see page 448)() user API. It returns the number of bytes of free space in the TCP's RX FIFO. 24. Changed default TICK resolution to 1ms (from 10ms) and improved accuracy. 25. Added outbound ping capabilities (i.e. board can now ping another board or a PC). To enable these features, define STACK\_USE\_ICMP\_CLIENT. This will enable several new APIs, including ICMPBeginUsage (see page 258)(), ICMPSendPing (see page 259)(), ICMPGetReply (see page 260)(), and ICMPEndUsage (see page 261)(). The functions should be called in this order. See the PingDemo (see page 97)() function in MainDemo.c for an example of how to use them. By default, pushing BUTTON3 (left-most one) will cause a ping to be sent to 4.78.194.159 (ww1.microchip.com). The response time will be displayed on the LCD (assuming your development board has an LCD). 26. Cleaned up C30 3.00 signed/unsigned warnings. 27. Removed PIC18F97J60\_TEST\_BOARD hardware profile support. This stack no longer supports it due to the old beta silicon (with errata) mounted on these boards. 28. Added support for ROM pointers for all of the SMTP strings (To, From, CC, Subject, etc.). If you use a ROM string, you must also set the corresponding SMTPClient.ROMPointers.xxx bit to let the SMTP module know which type of pointer was provided. See the SMTPDemo (see page 92)() code in MainDemo.c for an example calling sequence using both ROM and RAM strings for the various fields.

#### Fixes:

1. Fixed a critical TCP buffer corruption issue where the start of a TCB header overlapped with the last byte of the RX FIFO from the previous socket. This bug affected version 4.00RC only.

2. ETH97J60.c, TCPIP.h, and TCPIP Stack Version.txt were correctly readed to the TCPIP Demo App-C18 project using relative paths instead of absolute paths.
3. UDPOpen (see page 520)() now dynamically assigns a local port number if you call it and give it a 0x0000 port number. This should fix some UDP applications from not working (ex: DNS Client module) with some computers/routers/networks which throw away traffic originating from the invalid port 0x0000 value.
4. Fixed a ENC28J60 bank selection error that would occur if an application called GetCLKOUT() in ENC28J60. By default, this function is not called.
5. UnencodeURL (see page 222)() function in Helpers.c is now tested and working.
6. Fixed a TCP Window Update problem when TCPGetArray (see page 447)() was used. Before the problem was fixed, performance could have been terrible on reception.
7. Fixed a unintended TCP connection close if the socket was idle for about a minute. Now, TCP sockets will remain open indefinitely if there is no traffic going on.
8. Serial numbers >32K are now displayed correctly on the serial port as a positive value when C18 is used and the board is placed in configuration mode (BUTTON0 is depressed on power up).
9. HI-TECH PICC-18 compiler would previously incorrectly initialize the AppConfig structure.
10. Previously a processor reset was possible when accessing items in the AppConfig structure on 16 bit MCUs (PIC24, dsPIC) due to unaligned word accesses. This was fixed by reordering the Flags byte in the APP\_CONFIG structure.
11. Rewrote DHCP client state machine, fixing the previously known problem where it would not perform a new discovery if it was trying to renew a lease with an offline DHCP server.
12. Fixed a critical deadlock problem in the ETH97J60.c MAC layer driver for the PIC18F97J60 family Ethernet controller. Previously, it was possible (although rare) that the DMAST or TXRTS bits would get stuck set if too much Ethernet traffic was received within a short interval. Previously, the MACFlush() function was unnecessarily setting TXRST, which it should not do while the Ethernet interface or DMA is being used.
13. Fixed an HTTP server state machine problem where a new connection occurring too soon on a previously used socket could cause the HTTP server to no longer respond.
14. Fixed a potential memory corruption error in the HTTPGetVar() callback which would exceed the bounds of the VarString array when returning the VAR\_STACK\_DATE variable.
15. Fixed a TCP transmission sequence tracking problem whenever data is retransmitted and new unflushed data is also in the TX FIFO. Thanks go to Matt Watkins on the Microchip Ethernet forum for identifying this issue.

#### Known Problems:

1. RTL8019AS MAC layer driver has not been updated for new TCP module. Users requiring RTL8019AS support should continue to use stack version 3.75.
2. I2CEEPROM.c has not been tested or completed. Continue to use I2CEEPROM.c from stack version 3.75 if this file is needed.
3. Telnet (see page 481) server module does not implement a lot of Telnet (see page 481) functions. As a result, it will likely not display correctly or work at all with some Telnet (see page 481) clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
4. TFTPc module has not been tested with this version.
5. The default demo web pages which use AJAX do not automatically refresh themselves when viewed in Firefox 2.0.0.1. Earlier Firefox versions (1.5ish) probably work without any problem.
6. Files may be inaccessible in your MPFS if compiled with C18 for internal flash program memory and your total MPFS content is large (around 64KB or larger). The code attempts to access the ROM memory using a near rom pointer when a far rom pointer is needed.
7. If using MPLAB 7.52 all .s files that are compiled with C30 will not have the corresponding object file get stored in the correct directory. As a result, if you are compiling with C30 and with MPFS\_USE\_EEPROM not defined (i.e. storing web pages in internal program memory), the project won't link (throws a undefined reference to `MPFS\_Start'). As a workaround, remove the Intermediates Directory in the MPLAB project. Alternatively upgrade MPLAB to a newer version. MPLAB IDE 7.60+ may have this fixed.
8. If the DHCP client and DHCP server are used at the same time and you connect (see page 165) two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to renable it's DHCP server.
9. HI-TECH PICC-18 projects may not compile when MPFS\_USE\_EEPROM is not defined and you are trying to store web page data in internal FLASH program memory.

10. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC\_EXPLORER + Ethernet PICtail.

Testing and Performance Notes:

1. This stack version was compiled and tested with the following tool versions: -MPLAB IDE 7.52 -Microchip C30 version 3.00 -Microchip C18 version 3.10 -HI-TECH PICC-18 version 9.50PL3
2. Using the UDPPerformanceTest.c module, the stack can transmit around 220KBytes/second (1.75Mbits/second) of UDP application data on the PIC18F97J60 with internal Ethernet @ 41.66667MHz core clock, compiled using C18 3.10 with debug optimization settings.
3. Using the UDPPerformanceTest.c module, the stack can transmit around 392KBytes/second (3.14Mbits/second) of UDP application data on the PIC24HJ256GP610 with external ENC28J60 @ 40 MIPS, compiled using C30 3.00 with debug optimization settings.
4. Using the TCPPerformanceTest.c module, the stack can transmit around 58KBytes/second (464Kbits/second) of TCP application data on the PIC18F97J60 with internal Ethernet @ 41.66667MHz core clock, compiled using C18 3.10 with debug optimization settings, over Ethernet when using a tiny 200 byte TX TCP FIFO. Note that performance can be improved significantly by increasing the FIFO size and performance will drop significantly if the round trip TCP acknowledgement time is increased (ex: testing over the Internet instead of Ethernet).
5. Using the TCPPerformanceTest.c module, the stack can transmit around 69KBytes/second (558Kbits/second) of TCP application data on the PIC24HJ256GP610 with external ENC28J60 @ 40 MIPS, compiled using C30 3.00 with debug optimization settings, over Ethernet when using a tiny 200 byte TX TCP FIFO. Note that performance can be improved significantly by increasing the FIFO size and performance will drop significantly if the round trip TCP acknowledgement time is increased (ex: testing over the Internet instead of Ethernet).
6. Using the TCPPerformanceTest.c module, the stack can transmit around 178KBytes/second (1.42Mbits/second) of TCP application data on the PIC24HJ256GP610 with external ENC28J60 @ 40 MIPS, compiled using C30 3.00 with debug optimization settings, over Ethernet when using a larger 2000 byte TX TCP FIFO. Note that performance will drop significantly if the round trip TCP acknowledgement time is increased (ex: testing over the Internet instead of Ethernet).

\*\*\*\*\*

v4.00RC 28 December 2006

\*\*\*\*\*

IMPORTANT NOTE: If an external serial EEPROM memory is used to store AppConfig, it's contents will be invalidated the first time you run this version, restoring the AppConfig defaults. The AppConfig structure has been optimized. IMPORTANT NOTE2: If an external serial EEPROM memory for MPFS, you will need to recreate the MPFS image and program your EEPROM. A 32 bit addressing format is now used.

Changes:

1. Added Simple Mail Transfer Protocol (SMTP) client module and updated MainDemo.c to exercise the Email transmission functionality when a user pushes BUTTON0.
2. Added beta Telnet (see page 481) server module. See Known Problems section.
3. Completely revamped the TCP module. A real transmit FIFO and receive FIFO are allocated for each TCP socket now. This greatly enhances RFC compliance, communications robustness, and makes application development easier. New APIs were added for putting and getting arrays and strings (including ROM variants). Several TCP related bugs are now fixed as a result. Please report any bugs found in the new implementation.
4. Added TCPPutArray (see page 454)() API.
5. Added TCPPutROMArray (see page 455)() API.
6. Added TCPPutString (see page 456)() API.
7. Added TCPPutROMString (see page 455)() API.
8. Added TCPGetArray (see page 447)() API.
9. Changed TCPIsPutReady (see page 450)() API. Instead of returning a BOOL, it now returns a WORD. The WORD is a count of the number of bytes that TCPPut (see page 454)(), TCPPutArray (see page 454)(), etc. can immediately place in the output buffer. MAKE SURE THAT YOUR CODE DOES NOT COMPARE THE RETURN RESULT OF

TCPIsPutReady (see page 450)() DIRECTLY TO TRUE. For example, "if(TCPIsPutReady (see page 450)(MySocket (see page 306)) == TRUE){...}" must be converted over to: "if(TCPIsPutReady (see page 450)(MySocket (see page 306))){...}".

10.Changed TCPIsGetReady (see page 450)() API. Instead of returning a BOOL, it now returns a WORD. The WORD is a count of the number of bytes that TCPGet (see page 446)() or TCPGetArray (see page 447)() can immediately obtain. MAKE SURE THAT YOUR CODE DOES NOT COMPARE THE RETURN RESULT OF TCPIsGetReady (see page 450)() DIRECTLY TO TRUE. For example, "if(TCPIsGetReady (see page 450)(MySocket (see page 306)) == TRUE){...}" must be converted over to: "if(TCPIsGetReady (see page 450)(MySocket (see page 306))){...}".

11.Changed TCPDiscard (see page 442)() return type from BOOL to void.

12.Removed TCP\_NO\_WAIT\_FOR\_ACK option. It was defaulted to disabled in the last two releases of the stack and is not needed with the new TCP module.

13.Updated DNS module to include two new required APIs: DNSBeginUsage (see page 179)() and DNSEndUsage (see page 179)(). These functions control a one bit ownership semaphore to allow multiple applications to use the DNS module in series. If invoked correctly, this will prevent unintended bugs resulting from two applications trying to use the DNS module at the same time. Old applications, such as those based around the GenericTCPClient.c example must be updated to use these functions.

14.Started using a new project structure and folders. You must use MPLAB 7.41 or higher (stack is tested on MPLAB 7.50) to use the default workspaces/projects, which include files using relative paths. This should improve compatibility with some future code libraries released by Microchip. StackTsk.h was broken into TCPIPConfig.h, HardwareProfile.h, and StackTsk.h. TCPIPConfig.h now includes all stack configuration options and HardwareProfile.h contains all hardware options. No macros need be globally defined in MPLAB project now. TCPIP.h is the only header applications must include now, for any/all modules used.

15.Combined ARP.c/ARP.h and ARPTsk.c/ARPTsk.h into a single file pair: ARP.c/ARP.h. Applications built using a prior stack revision must remove all instances including "ARPTsk.h" and replace it with "ARP.h" instead. The ARP module is now simpler, more linear (easier to read), and being in one source file, allows the C compiler to optimize better.

16.Added PIC18F67J60\_TEST\_BOARD hardware profile to HardwareProfiles.h. This hardware profile is designed for 05-60091 (Rev 1), a development board that is not in production at this time.

17.Added DSPICDEMNET1 and DSPICDEMNET2 hardware profiles to HardwareProfiles.h for eventual support of the Microchip dsPICDEM.net 1 and dsPICDEM.net 2 demo boards. These two boards use the RTL8019AS Ethernet controller and a 24LC515 EEPROM. These changes are currently incomplete and these profiles cannot be used.

18.Began rewriting I2CEEPROM.c to support 16 bit CPUs, including the dsPIC30F6014 used on the dsPICDEM.net 1 and 2 demo boards. Note that work here is incomplete and cannot be used as a result -- see Known Problems section.

19.Partially updated RTL8019AS.c to support 16 bit CPUs, including the dsPIC30F6014 used on the dsPICDEM.net 1 and 2 demo board. Note that work here is incomplete and cannot be used as a result -- see Known Problems section.

20.Updated SNMP.c to use new typedefs in GenericTypedefs.h. Also SNMP was tested in this version. SNMP.mib was updated some to better reflect current hardware.

21.Added AN870 SNMP callbacks to MainDemo.c (a feature that was missing in 3.xx releases). This code will get compiled when STACK\_USE\_SNMP\_SERVER is defined in TCPIPConfig.h.

22.Removed all instances of MPFS\_USE\_PGRM for storing in internal FLASH program memory. Storage in internal program memory is now the default. Define MPFS\_USE\_EEPROM to override the default and store MPFS in an external EEPROM memory.

23.Decreased program memory needed for Announce.c module by about 180 bytes. Multiple inline calls to UDPPut (see page 524)() were removed.

24.UDP checksum checking logic has been improved. The UDP layer now avoids writing the pseudo header checksum in the RX buffer.

25.Swapped endianness of the returned checksum from CalcIPBufferChecksum (see page 210)(). Rewrote CalcIPBufferChecksum (see page 210)() in Helpers.c. This improves consistency.

26.Improved swapl() in Helpers.c.

27.Improved USART baud rate (SPBRG) calculation for PIC18s. Rounding is now done to chose the most optimal value and the code will automatically select high baud rate mode (BRGH=1) if possible. Additional improvements can be made if using a newer PIC18 with the 16 bit baud rate generator.

28.Added GenericTCPServer.c example file to complement GenericTCPClient.c. The server is enabled by defining STACK\_USE\_GENERIC\_TCP\_SERVER\_EXAMPLE in TCPIPConfig.h.

29.Renamed STACK\_USE\_GENERIC\_TCP\_EXAMPLE definition to STACK\_USE\_GENERIC\_TCP\_CLIENT\_EXAMPLE for consistency with new server example.

30.Defaulted MPFS.exe to generate binary MPFS images using 32 bit addressing. MPFS.h has been modified to also default to use 32 bit addressing of external EEPROM images. You must rebuild any old MPFS images and reprogram them if upgrading from a previous TCP/IP stack revision, which defaulted to use 16 bit addressing.

31.Updated MPFS.exe to #include "TCPIP.h" instead of ".HeadersCompiler.h" in C files generated by the utility.

32.Added MPFSv2.exe PC utility for generating large MPFS images in program memory (ASM30 code) for C30 users. Previously, the C30 compiler placed a limit of less than 32KB of total MPFS size due to the PSV window size limitation on PIC24/dsPIC devices. To get around the limitation, use the new MPFSv2.exe utility to generate an .s file which can be included in your project instead of the .c file generated by the traditional MPFS.exe utility.

## Fixes:

1. Fixed a bug in ARPPProcess (see page 156)() which would incorrectly send an ARP response to an incorrect MAC & IP address if a TX buffer wasn't immediately available.
2. Fixed a TCP bug where TCPIsGetReady (see page 450)() would return TRUE even if no data was left in the received packet. Previously you had to call TCPGet (see page 446)() one last time and have it fail before TCPIsGetReady (see page 450)() would return FALSE.
3. Modified TCP state machine. Established connections will no longer automatically close if left idle for approximately 45 seconds. Note that your application needs to ensure that no sockets unintentionally get lost (For example: a server socket that received data only is established and the cable breaks while connected. In this case, the socket would never be detected as being disconnected since the server never attempts to transmit anything).
4. Stopped overclocking dsPIC33 and PIC24H devices. Previously PLLFBD was incorrectly set to 39 instead of 38 to yield a resulting Fosc of 84MHz (42MIPS) instead of 80MHz (40MIPS) with the default Explorer 16 development board. Thanks go to Matt Watkins on the Microchip Ethernet Forum for pointing this error out.
5. Corrected a bug in IP.c where IPHeaderLen would not be properly initialized if a NON\_MCHP\_MAC was used (ex: RTL8019AS) and IPSetRxBuffer() was called. This bug did not affect ENC28J60 or PIC18F97J60 family support. Thanks go to Darren Rook for identifying this issue.
6. Updated checksum checking code in ENC28J60.c for latest silicon DMA checksum errata.
7. Declared TickCount in Tick.c/Tick.h as volatile and implemented an interrupt safe reading procedure in TickGet (see page 512)(). Since this multibyte variable is modified in the ISR and read in the mainline code, these changes are needed to prevent rare inconsistency bugs.
8. Fixed Announce.c so the unicast remoteNode of the requesting packet would be used rather than the remoteNode of the last received packet, which may not be correct when transmitting. Thanks go to Brett Caulton for identifying this issue.
9. Fixed a DHCP bug which would cause DHCP renewals to continually occur after only 60 seconds once the original lease expired. Thanks go to Brett Caulton for identifying this issue and fix.
10. Fixed a potential TCP socket leak in the FTP module. Previously FTPDataSocket would not be reliably initialized nor closed if the connection was killed forcefully (user killed application, cable disconnected while transferring, etc.).

## Known Problems:

1. RTL8019AS MAC layer driver has not been updated for new TCP module. Users requiring RTL8019AS support should continue to use stack version 3.75.
2. I2CEEPROM.c has not been tested or completed. Continue to use I2CEEPROM.c from stack version 3.75 if this file is needed.
3. Telnet (see page 481) server module is still in development. No user authentication features are currently implemented. Some telnet clients may render the telnet server output incorrectly (in the wrong locations or wrong colors). Testing has only been done with the Microsoft Windows telnet.exe utility that comes Windows XP.
4. DHCP will continually send out DHCP Request packets when the lease expires and the original DHCP server that gave the lease is offline. The board will continue to use the expired IP address until the DHCP server comes back online, at which point the lease will be renewed or a new discovery will occur. A new discovery should occur after timing out, instead. It is believed that this problem has always existed in previous stack revisions.
5. DHCP will continually send out DHCP Request packets when the lease expires and the original DHCP server that gave the lease does not include Option 54, the Server Identifier. A new discovery should occur after timing out. It is believed that this problem has always existed in previous stack revisions.
6. TFTPc module has not been tested with this version.
7. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).

\*\*\*\*\*

v3.75 14 August 2006

\*\*\*\*\*

Changes:

---

1. Added beta DNS client module (DNS.c). DHCP was also updated to obtain a DNS server address. Added AppConfig.PrimaryDNSServer IP address. Added STACK\_USE\_DNS configuration macro. To use the DNS client, call DNSResolve (see page 180)() with the server name, ex: DNSResolve ("www.microchip.com"), and then periodically call DNSIsResolved (see page 181)() until it returns TRUE, ex: DNSIsResolved (see page 181)(&IPAddressDestination). Only one DNS resolution can be in progress at a time. Because the DNS client is a beta module, the API or code may change before being finalized. No formal DNS API documentation is available yet.
2. Added beta NetBIOS Name Service responder module (NBNS.c). Added AppConfig.NetBIOSName string. Added STACK\_USE\_NBNS configuration macro. Added MY\_DEFAULT\_HOST\_NAME macro in StackTsk.h. Now, whenever a NetBIOS broadcast attempting to resolve AppConfig.NetBIOSName arrives, a response will be made. This form of name resolution only works on a single subnet. Off the subnet, manual registration in a DNS server or other means will be needed to allow the local Host Name to be recognized and translated to an IP address. The default NetBIOS name for the board is "MCHPBOARD". To test the NetBIOS Name Service module, try entering http://MCHPBOARD/ into your web browser instead of the board's IP address.
3. Added beta HTTP client module (GenericTCPClient.c). This module demonstrates how to make a TCP client application. To test this module, uncomment the STACK\_USE\_GENERIC\_TCP\_EXAMPLE macro in StackTsk.h, recompile, and then press the BUTTON1 button while the stack is running. RemoteURL (see page 94)[] should be downloaded from ServerName (see page 94)[] and written to the UART. For the default values of ServerName (see page 94)[] and RemoteURL (see page 94)[], the HTML search page for "Microchip" will be fetched from "www.google.com" and written to the serial port. No formal documentation is available for this example yet.
4. Added Embedded Ethernet Device Discoverer PC project to aid in embedded product discovery when connected to a network and demonstrate how to write PC applications which can communicate with embedded devices. The source code for this device is included. It can be built using the Microsoft Visual C# 2005 Express Edition compiler. At the time of stack release, this 3rd party PC development tool can be downloaded at no cost from <http://msdn.microsoft.com/vstudio/express/>. If using only the Microchip Device Discoverer executable file without the Visual C# compiler, the .NET Framework 2.0 must be installed on the local PC. The application setup utility should allow dynamic downloading of this component if the target machine does not already have it installed.
5. Updated Announce.c to listen (see page 169) and respond to discovery requests sent to UDP port 30303 starting with the character 'D'. To test this functionality, use the Embedded Ethernet Device Discoverer on a PC connected to the same subnet.
6. Updated UART configuration menu to accommodate the new beta module configuration options (DNS server address, device host name).
7. Increased MPFS reserve block to 64 bytes from 32. Also, because the APP\_CONFIG structure was updated, all current MPFS images and data stored in deployed EEPROMs needs to be updated.
8. Added a means to erase (invalidate) the onboard EEPROM using the BUTTON0 momentary switch (right-most switch on demo boards with multiple switches). To erase the EEPROM, hold down BUTTON0, RESET the board (press and release MCLR switch), and then continue to hold down BUTTON0 for an additional 4 seconds. If you press MCLR again, the EEPROM contents will now be invalid. If you press '0' on the UART, the same configuration that was read prior to invalidating the contents will be written back into the EEPROM. Invalidating the EEPROM allows the MY\_DEFAULT\_\* constants to get loaded into a previously programmed EEPROM chip. Because of change #7, this procedure should be done for all currently programmed EEPROMs to prevent anomalous values from being read.
9. remoteNode in StackTsk.c was changed from private to global scope. Now external modules can reference the address of the last received packet. Announce.c uses this to send a unicast response to a broadcast discovery request.
10. All stack modules that can be disabled (DHCP.c, FTP.c, etc) now will no longer emit a compiler error if you have it in the project without defining the appropriate macro (STACK\_USE\_DHCP, STACK\_USE\_FTP, etc). It will simply generate no machine code when compiled and the stack will not use that module. Make sure the proper macro is defined for each module that you wish to use.
11. Added SetRXHashTableEntry() to ENC28J60.c. This function can be used to set the appropriate bit in the Hash Table registers to join a particular multicast group.
12. Added Realtek RTL8019AS Ethernet controller support to the stack. MAC.c was renamed to RTL8019AS.c. This Ethernet controller is not recommended for new designs. RTL8019AS support was reintroduced to provide ongoing assistance to former Application designs implementing this chip. For new applications, use the Microchip ENC28J60 or PIC18F97J60 family of microcontrollers.
13. Added I2C EEPROM support for MPFS storage. In older 2.xx stack revisions, I2C EEPROM was supported by the XEEPROM.c file. This file has been renamed to I2CEEPROM.c. It is mutually exclusive with SPIEEPROM.c, and only one may be included in the project at a time.
14. Added new hardware definitions to Compiler.h. Pin mappings for the PICDEMNET and PIC18F97J60\_TEST\_BOARD boards have been added. FS\_USB was also defined; however, it is untested and not recommended. See Compiler.h. The PIC18F97J60\_TEST\_BOARD is a non-production board that some Early Adopters of the PIC18F97J60 family parts have.
15. Changed type definitions for BYTE\_VAL, WORD\_VAL, DWORD\_VAL, and moved the generic typedefs to GenericTypeDefs.h from StackTsk.h. This should improve compatibility with some future code

libraries released by Microchip. 16.LCDBlocking.c module was modified to support 4-bit interfaces to LCD modules. The PICDEM.net board has the module wired using a 4-bit bus.

#### Fixes:

1. Fixed a serious MAC TXBuffer leak in TCP.c. Previously TCP.c would allocate a buffer for each socket in use, but under heavy traffic conditions (ex: user holds down F5 on web browser), the buffer handle might have been discarded before releasing the buffer. As a result all TCP connections would have lost the ability to send any application data after the TXBuffer pool ran out.
2. In the TCP\_SYN\_SENT TCP state, ACKs may only be received (as opposed to SYN+ACK packets) if the remote node thinks the connection is already open. A RST is now sent in response to an unexpected ACK, which may improve reconnection time when this (rare) condition occurs.
3. A bug was present in the UDP module where remote MAC addresses would be cached for each socket, even when UDPInit (see page 530) or UDPClose (see page 521) was called, or the microcontroller was reset. As a result, responses to incoming packets could have been sent to the wrong MAC address. UDP Sockets (see page 146) are now properly initialized/closed.
4. Fixed a potential timing bug in LCDBlocking.c. For lower values of CLOCK\_FREQ, insufficient delay time was given to the LCD module, potentially causing improper operation.
5. Changed PIC24F to default to the XT oscillator fuse rather than HS. The PIC24FJ128GA010 data sheet, rev. C reports that 8MHz should be used with XT mode, not HS mode like prior data sheets.
6. Added a couple of wait states to the Realtek RTL8019AS MAC layer module for NICPut() and NICGet(). Previously, the PICmicro could not operate above approximately 25MHz without losing communication with the RTL8019AS chip.
7. Updated PC based MPFS utility. When generating C files to be added to your MPLAB project, the include path to "Compiler.h" is now "..\IncludeCompiler.h". The output file, ex: "MPFSImg.c" should be placed in the "Source" subfolder before compiling. For example, if you are in the main stack folder with the MPLAB projects, type: "mpfs /c WebPages SourceMPFSImg.c"
8. IP Gleaning will now get properly disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled. The stack will still respond to ping requests which have the wrong destination IP address, but a correct MAC address. However, the stack will continue to keep its statically defined IP address when DHCP/IP Gleaning are disabled and the ping arrives.
9. SPIEEPROM.c now saves and reconfigures the EEPROM\_SPICON1 register (SSPCON1) before reading or writing to the SPI. After the read/write, it restores the saved state. This allows the SPI bus to operate at different speeds, depending on what peripheral is being accessed if other devices share the bus and can support different speeds. In particular, this fixes the SPI @ 10.4MHz problem on the PICDEM.net 2 board when using the ENC28J60.

#### Known Problems:

1. DHCP will continually send out DHCP Request packets when the lease expires and the original DHCP server that gave the lease is offline. The board will continue to use the expired IP address until the DHCP server comes back online, at which point the lease will be renewed or a new discovery will occur. A new discovery should occur after timing out, instead. It is believed that this problem has always existed in previous stack revisions.
2. DHCP will continually send out DHCP Request packets when the lease expires and the original DHCP server that gave the lease does not include Option 54, the Server Identifier. A new discovery should occur after timing out. It is believed that this problem has always existed in previous stack revisions.
3. When an MPFS .c image file is added to a C30 project, a linking error reporting insufficient contiguous .const memory may occur when too much data is in the MPFS image (PSV window size limitation). Using the PSV window, 1 out of every 3 program memory bytes is wasted.
4. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.
5. SNMP, TFTPc modules have not been tested with this version.
6. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).
7. The C30 linker may misplace the \_\_CONFIG2 section or disallow usage of MPFS images that are too big (add too much to the .const code section). The consequences of this are that the first configuration word at 0x157FC may not get set through code (must use the Configuration Bits dialog instead), and/or the project will not compile. This problem has been observed with C30 ver. 2.02 on the PIC24FJ128GA010 product. To work around this problem, the p24FJ128GA010.gld linker script has been modified. Specifically, line 68 has been commented out, which causes the linker to place all .text sections after placing all absolute sections. SSR 25966 in the C30 2.02 release notes may be related.
8. It is observed with the Realtek RTL8019AS Ethernet controller and the demo AJAX web page which self refreshes

rapidly, that occasional HTTP GET requests sent by the computer do not get received by the HTTP server. This is believed to be a RTL8019AS MAC layer bug. The TCP protocol handles the packet loss, but application performance suffers while waiting for the TCP retransmission. This problem is not observed with ENC28J60.c or ETH97J60.c MAC layers.

9. The HI-TECH compiler version 9.50PL1 crashes when compiling LCDBlocking.c with 4 bit mode (PICDEMNET) and using a warning level of -3 or higher. To work around the problem, the HI TECH projects were set to use warning level -4.

Guiding Notes:

1. To use the stack on a classic PICDEM.net demo board with the Realtek Ethernet controller, a PIC18F452 processor, and Microchip C18: -Use the C18EEPROM MPLAB project -Change the processor in the MPLAB IDE -Change linker script to "18f452i.lkr" in the MPLAB project. Use the one provided in the Linker subfolder, it has been modified to make more RAM available. -Update the hardware definitions macro. Click on Project -> Build Options... -> Project -> MPLAB C18 -> Add PICDEMNET, remove HPC\_EXPLORER) -Remove ENC28J60.c from the project -Remove SPIEEPROM.c from the project -Add RTL8019AS.c to the project -Add I2CEEPROM.c to the project -Enable all compiler optimizations (Project -> Build Options... -> Project -> MPLAB C18 -> Categories Optimization -> Enable all)

\*\*\*\*\*

v3.60 12 July 2006

\*\*\*\*\*

General Information: This stack version is being publicly released, so the following changes are with respect to the prior public stack release (v3.02). Interim stack changes for version 3.16 and 3.50 are documented below for those using non-public releases, but can be ignored by most people.

Troubleshooting notes:

1. If you have an Ethernet PICtail revision 2.1 and are having reliability issues when viewing the fast-refresh demo web page, you may need to install resistors in series with the ENC28J60 SI, nCS, and SCK pins. The recommended value is 100 to 200 ohms. This will reduce signal undershoot caused by long traces (parasitic inductance), which can violate the absolute maximum electrical specs and cause SPI data corruption. The HPC Explorer Rev 5 has fairly long traces to the PICtail connector.
2. Enabling C30 2.02 compiler optimizations on the dsPIC33FJ256GP710, PIC24HJ256GP610 ES chips may produce unreliable code.
3. When changing a C30 project to a PIC24H or dsPIC33F processor on the Explorer 16 demo board, the JTAG configuration fuse should be disabled to free the I/O pins associated with it. JTAG is enabled by default.
4. This stack release was tested using MPLAB 7.40, C18 version 3.03, C30 version 2.02, and HI TECH PICC18 version 9.50PL1.
5. When using the Ethernet PICtail board and HPC Explorer demo boards, make sure to plug the power into the Ethernet PICtail and not the HPC Explorer. The HPC Explorer's power regulator cannot provide enough current.

Changes:

1. Source files have been split into separate directories. To compile old applications with this new stack, application source files may need to be updated to include the proper path to the stack header files.
2. New MPLAB projects have been created: -C18EEPROM: Equivalent to the previously named "mpnicee" project. Designed for PIC18's using the C18 compiler. Web page content, board's IP address, MAC address, DHCP enabled state, etc. is stored in an external SPI EEPROM (25LC256 on demo boards). FTP Server demo is included. -C30EEPROM: New supporting PIC24 and dsPIC controllers using the C30 compiler. Similar to C18EEPROM. -C18ProgramMem: Equivalent to the previously named "mpnicpg" project. Web page content stored in internal FLASH program memory. Board's IP address, MAC address, DHCP enabled state, etc. is stored only in RAM and defaults are loaded from MY\_DEFAULT\_\* constants in StackTsk.h. FTP Server demo is not included. Web pages cannot be updated remotely. -C30ProgramMem: New supporting PIC24 and dsPIC controllers using the C30 compiler. Similar to C18ProgramMem. -HTC18EEPROM: Equivalent to the previously named "htnicee" project. Designed for PIC18's using the HI TECH PICC18 compiler. Similar to C18EEPROM. -HTC18ProgramMem: Equivalent to the previously named "htnicpg" project. Designed for PIC18's using the HI TECH PICC18 compiler. Similar to C18ProgramMem.
3. Created hardware definitions (pins, interrupt flags, special registers, etc) in Compiler.h for easy changing of hardware. Four demo board combinations are supported out-of-box now: -EXPLORER\_16: Explorer 16 motherboard + Ethernet

PICtail Plus daughter card. Tested with dsPIC33FJ256GP710, PIC24HJ256GP610, and PIC24F128GA010 ES PIMs. -HPC\_EXPLORER: PICDEM HPC Explorer motherboard + Ethernet PICtail daughter card. Tested with PIC18F8722 onboard and PIC18F87J10 PIM. -DSPICDEM11: dsPICDEM 1.1 motherboard + Ethernet PICtail daughter card (manually air wired). See Compiler.h for proper pins to air wire. Tested with dsPIC30F6014A PIM. -PICDEMNET2: PICDEM.net 2 motherboard (PIC18F97J60) Change boards by changing the defined macro (Project -> Build Options... -> Project -> MPLAB Cxx -> Add macro). When moving to custom hardware, add an appropriate profile to Compiler.h. YOUR\_BOARD is present as a placeholder.

4. Added Ethernet PICtail Plus schematic (reference ENC28J60 daughter card design for Explorer 16 demo board). These boards have a Microchip part number of AC164123.
5. Latest ENC28J60 rev. B5 errata workarounds added. The code checks the EREVID register and implements the appropriate workarounds as needed for the silicon revision, so rev. B1, B4, and B5 are all supported in this stack release.
6. Significantly revised demonstration web page content in WebPages folder to use AJAX technology. Using asynchronous JavaScript code executing in the web browser, the status sections of the page are updated rapidly from the web server without doing a full page refresh. As a result, a virtually real time update of the potentiometer and button values can be displayed. Due to the constant use of new TCP sockets, multiple simultaneous users are not recommended. See the Index.cgi file for a simple static method of retrieving dynamic variables from the HTTP server.
7. Changed IP Gleaning procedure. Now, if DHCP is enabled, the DHCP module will continue to look for a new IP address/renew existing IP address if the IP address is configured using IP Gleaning. Previously, the DHCP module would be disabled once a successful ICMP packet was received and used to configure the IP address.
8. MAX\_RETRY\_COUNTS is 3 (previously it was 3, but an interim release changed it to 5).
9. Updated TCP state machine. It now includes the TCP\_FIN\_WAIT\_2 state. Some other changes were made to handle errors more robustly.
10. AN0String and AN1String now return all characters excluding the null terminator when the HTTP server calls HTTPGetVar (except when the string is 0 length). Previously, the null terminator was returned as well.
11. Dynamic pages (ie: .cgi files) are now served with an expired HTTP header to prevent browser caching and allow more dynamic content to be displayed.
12. Support for the HI TECH PICC18 compiler has changed. Special Function Register bits and other definitions have changed substantially from the previous HI TECH PICC18 projects in TCP/IP stack version 3.02 and earlier. The C18/C30 SFR and SFRbits naming conventions are now used and special remapping macros in Compiler.h are used to maintain a consistent syntax. The HI TECH PICC18 projects were tested with compiler version 9.50PL1 on the HPC Explorer board (PIC18F8722).
13. FTP client hash printing has been added to the FTP server. Now, whenever a chunk of data is successfully uploaded to the device, a '#' character will appear on the FTP client screen. The numbers of bytes each '#' represents is variable.
14. To improve maintainability, built in support for the "Compatible" A/D converter present on older PIC18 parts (ex: PIC18F452) has been removed.
15. Removed old LCD code originally provided for the PICDEM.net demo board.
16. Added LCDBlocking.c and LCDBlocking.h, which implement simple routines for writing to the LCD module on the Explorer 16 and PICDEM.net 2 development boards. The LCD on the dsPICDEM 1.1 board is not supported. The stack version and IP address are shown on the LCD on power up.
17. UART functions in MainDemo.c were replaced with C18 and C30 peripheral library functions. However, because the UART peripheral libraries are not being updated for newer silicon devices, the code was copied into UART.c and is compiled with the stack.
18. Multiple TX buffer support has been implemented. Most stack layers have been touched. ENC28J60.c has the most extensive changes. Each socket may use only one TX buffer.
19. Implemented TCP retransmission support regardless of if TCP\_NO\_WAIT\_FOR\_ACK is defined or not.
20. TCP\_NO\_WAIT\_FOR\_ACK in StackTsk.h has been undefined by default. This should increase default TCP connection robustness. Packets sent from the stack to the remote node will now be detected and retransmitted if lost or corrupted.
21. All TCP packets are now retransmitted immediately after being initially transmitted when TCP\_NO\_WAIT\_FOR\_ACK is undefined. This improves throughput greatly when communicating with systems which wait a long time before transmitting ACKs. TCP/IP stacks, such as that used by Microsoft Windows, implement the TCP Delayed Acknowledgement algorithm, which is why this retransmission is necessary for high performance. The double transmission feature can be disabled in the Microchip TCP/IP stack by defining "DEBUG" either in the TCP.c file or the project compiler macros section. Using DEBUG mode can be useful when trying to look for errors using Ethreal [ <http://www.ethereal/> ].
22. Lowered TCP\_START\_TIMEOUT\_VAL (☞ see page 479) from 60 seconds to 3 seconds. 60 seconds is an unreasonably long timeout for modern day network speeds.
23. Native support for the SLIP module has been dropped.

#### Fixes:

1. A new IP address obtained via IP Gleaning will now update the LCD (if present), invoke the Announce (☞ see page 149) module (for MCHPDetect.exe), and output the new address out the RS232 port.

2. DHCP client will now correctly use the first DHCP offer received when connected to a network running multiple DHCP servers. Previously, the board would get no IP address when attached to a network with multiple DHCP servers (unless the DHCP request was transmitted before a second DHCP offer was received -- a relatively rare event). Additionally, DHCPLeaseTime does not get reset to 60 seconds or the value stored in the last DHCP packet received prior to receiving the ACK.
3. UDPProces() will now correctly process received UDP packets that have a 0x0000 checksum field. The UDP protocol specifies that 0x0000 means the checksum is disabled. Packets with a 0x0000 checksum were previously thrown away unless the calculated checksum also happened to be 0x0000.
4. The TCPIsPutReady (see page 450)() function will now honor the remote node's TCP window size. In other words, if the remote application pauses or cannot handle the incoming data rate, the TCP flow control feature will correctly function. Previously, if the remote node ran out of incoming buffer memory, the TCP layer would still allow more data to be transmitted. This would result in the loss or corruption of application data, with a potentially broken connection. The change requires 2 more bytes of RAM per TCP socket (TCB array).

#### Known Problems:

1. On PICDEM.net 2 board ENC28J60 and 25LC256 EEPROM share the same SPI1 module. At 3.3V, the 25LC256 is only rated to 5MHz SPI clock, but the code is setting it to 10.4MHz because the MACInit() function reconfigures the same SPI1 module.
2. DHCP will continually send out DHCP Request packets when the lease expires and the original DHCP server that gave the lease is offline. The board will continue to use the expired IP address until the DHCP server comes back online, at which point the lease will be renewed or a new discovery will occur. A new discovery should occur after timing out, instead. It is believe that this problem has always existed in previous stack revisions.
3. DHCP will continually send out DHCP Request packets when the lease expires and the original DHCP server that gave the lease does not include Option 54, the Server Identifier. A new discovery should occur after timing out. It is believe that this problem has always existed in previous stack revisions.
4. The MPFS utility has not been updated. When creating a .c image file, the include path for the Compiler.h file will be incorrect and need to be manually updated to "..IncludeCompiler.h".
5. When an MPFS .c image file is added to a C30 project, a linking error reporting insufficient contiguous .const memory may occur when too much data is in the MPFS image (PSV window size limitation). Using the PSV window, 1 out of every 3 program memory bytes is wasted.
6. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.
7. SNMP, TFTPc modules have not been tested with this version.
8. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).
9. IP Gleaning may not get disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled.
10. The C30 linker may misplace the \_\_CONFIG2 section or disallow usage of MPFS images that are too big (add too much to the .const code section). The consequences of this are that the first configuration word at 0x157FC may not get set through code (must use the Configuration Bits dialog instead), and/or the project will not compile. This problem has been observed with C30 ver. 2.02 on the PIC24FJ128GA010 product. To work around this problem, the p24FJ128GA010.gld linker script has been modified. Specifically, line 68 has been commented out, which causes the linker to place all .text sections after placing all absolute sections. SSR 25966 in the C30 2.02 release notes may be related.

#### Guiding Notes:

1. To change processors using a C18\* project: -Change the processor in the MPLAB IDE -Change linker script (ex: 18f87j10i.lkr) in the MPLAB project. Use \*.i.lkr if the ICD2 is going to be used to debug with. -Update the hardware definitions in Compiler.h or change your demo board selection macro. (Project -> Build Options... -> Project -> MPLAB Cxx -> PICDEMNET2, etc)
2. To change processors using a HTC18\* project: -Change the processor in the MPLAB IDE -Update the hardware definitions in Compiler.h or change your demo board selection macro. (Project -> Build Options... -> Project -> MPLAB Cxx -> PICDEMNET2, etc)
3. To change processors using a C30\* project: -Change the processor in the MPLAB IDE -Change linker script (ex: p33FJ256GP710.gld) in the MPLAB project. -Update the hardware definitions in Compiler.h or change your demo board selection macro. (Project -> Build Options... -> Project -> MPLAB Cxx -> DSPICDEM11, etc) -Disable JTAG configuration fuse, if enabled
4. When using the PICDEM.net 2 board, to write code targeting the PIC18F97J60 family Ethernet module: -Remove

ENC28J60.c from the project -Add ETH97J60.c to the project -Plug the Ethernet cable into the left-most RJ45 jack (next to LCD)

5. When using the PICDEM.net 2 board, to write code targeting the ENC28J60 Ethernet device: -Make sure ENC28J60.c is in the project -Make sure that ETH97J60.c is not in the project -Plug the Ethernet cable into the right-most RJ45 jack (next to board edge)
6. When using the PICDEM.net 2 board, to write code targeting an Ethernet PICtail module (ENC28J60): -Make sure ENC28J60.c is in the project -Make sure that ETH97J60.c is not in the project -Make sure that the Ethernet PICtail J9 jumper is in the 2-3 position (default). -Properly update the hardware profile in Compiler.h. ENC\_CS\_TRIS and ENC\_CS\_IO need to be changed from D3 to B3. -Plug the Ethernet cable into the PICtail -Plug power into the PICDEM.net 2 board
7. When using the Explorer 16 and Ethernet PICtail Plus demo boards, make sure to mate the PICtail to the motherboard using the topmost socket position, leaving the cable hanging over prototyping area. If SPI2 is desired, the PICtail should have the same orientation but be installed in the middle slot. Using SPI2, the hardware profile will need to be updated in Compiler.h.

\*\*\*\*\*

v3.50 13 April 2006

\*\*\*\*\*

#### Changes:

1. Improved dsPIC33F and PIC24H support. UART functions are included now instead of precompiled object files for the PIC24F. The 12-bit A/D converter is now shown in use on the demo web content. When changing a C30 project to a PIC24H or dsPIC33F processor on the Explorer 16 demo board, the JTAG configuration fuse should be disabled to free the I/O pins associated with it. JTAG is enabled by default.
2. Added LCDBlocking.c and LCDBlocking.h, which implement simple routines for writing to the LCD module on the Explorer 16 development board. The stack version and IP address are shown on the LCD on power up.
3. Added "C18ProgramMem" and "C30ProgramMem" MPLAB projects for MPFS storage (web page content) on on-chip program memory. These projects are equivalent to the previously named "mpnicpg" project in prior stack releases.
4. Multiple TX buffer support has been implemented. Most stack layers have been touched. ENC28J60.c has the most extensive changes. Each socket may use only one TX buffer.
5. Implemented TCP retransmission support when TCP\_NO\_WAIT\_FOR\_ACK is undefined.
6. TCP\_NO\_WAIT\_FOR\_ACK in StackTsk.h has been undefined by default. This should increase default TCP connection robustness.
7. All TCP packets are now retransmitted immediately after being initially transmitted when TCP\_NO\_WAIT\_FOR\_ACK is undefined. This improves throughput greatly when communicating with systems which wait a long time before transmitting ACKs.
8. Lowered TCP\_START\_TIMEOUT\_VAL (☒ see page 479) from 60 seconds to 3 seconds.
9. Increased MAX\_RETRY\_COUNTS from 3 to 5 times.
10. The example HTTP server now returns a content expiration date which has already past. This prevents web browser caching and allows more dynamic content to be displayed.
11. Added WebPages\_JScript folder, with new web pages that support dynamic page updates without a full page reload. A tiny page of dynamic variables is returned by the web server and Javascript executing on the target web browser changes DOM elements as needed. Button S5 (RA7) on the Explorer 16 demo board and S1 (RB0) on the HPC Explorer demo board changes the page color scheme. The rapid dynamic updates do not work on some web browsers (Internet Explorer works, Firefox does not).

#### Known Problems:

1. MPFS utility has not been updated. When creating a .c image file, the include path for the compiler.h file will be incorrect and need to be manually updated.
2. When an MPFS .c image file is added to a C30 project, a linking error reporting insufficient contiguous .const memory may occur (PSV window size limitation).

3. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.
4. SNMP, TFTPc, SLIP modules have not been tested with this version.
5. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).
6. IP Gleaning may not get disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled.
7. The IP address being outputted out the RS232 port and through the Announce (see page 149) module does not happen when the IP address is configured using IP Gleaning.
8. On the PIC24F with C30 compiler optimizations enabled (such as Option 3, maximum speed), the project may not work. The PIC24F headers that come with C30 ver. 2.01 declare several SFRs without using the volatile keyword.
9. dsPIC30 support is incomplete. Currently PIC18, PIC24F, PIC24H, and dsPIC33F processors are supported.

\*\*\*\*\*

v3.16.00: 06 March 2006

\*\*\*\*\*

#### Changes:

1. Added unified support for both the Microchip C18 and C30 compilers. The intention is to allow one code base to be compiled for any PIC18, PIC24F/H, dsPIC30, or dsPIC33 product (with adequate memory). See the "Tested Using" section for what is known to work.
2. To improve maintainability, support for the HI-TECH PICC18 compiler has been dropped.
3. New project workspaces have been created, "C30EEPROM.mcw" and "C18EEPROM.mcw". C18EEPROM.mcw is equivalent to the previously named "mpnicee.mcw." C30EEPROM is intended to be used for PIC24 and dsPIC 16-bit controllers.
4. Source files have been split into separate directories.
5. Latest ENC28J60 rev. B5 errata workarounds added. The code checks the EREVID register and implements the appropriate workarounds as needed for the silicon revision, so rev. B1, B4, and B5 are all supported in this stack release.
6. Removed old LCD code originally provided for the PICDEM.net demo board.
7. To improve maintainability, built in support for the "Compatible" A/D converter present on older PIC18 parts (ex: PIC18F452) has been removed.
8. UART functions in MainDemo.c were replaced with C18 and C30 peripheral library functions.

#### Tested Using:

1. Software: -MPLAB version 7.31.01 -C18 version 3.02 -C30 version 2.01
2. Hardware: -PICDEM HPC Explorer rev. 4 (PIC18F8722) + Ethernet PICtail Daughter Board (ENC28J60 B1) -Explorer 16 rev. 4 (PIC24FJ128GA010 ES and dsPIC33FJ256GP710 ES) + Ethernet PICtail+ Daughter card (ENC28J60 B1).
3. Notes: -MPLAB 7.31.01 is a development build. The publicly available version 7.31 should work fine, with the exception of being unable to program dsPIC33 and PIC24H parts with the ICD 2. -No dsPIC30 or PIC24H parts have been tested yet.

#### Known Problems:

1. MPFS utility has not been updated. When creating a .c image file, the include path for the compiler.h file will be incorrect and need to be manually updated.
2. When an MPFS .c image file is added to a C30 project, a linking error reporting insufficient contiguous .const memory may occur.
3. On the PIC24FJ128GA010, it is observed that some inbound packets are lost from time to time with no anticipated reason.
4. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.
5. SNMP, TFTPc, SLIP modules have not been tested with this version.

6. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).
7. IP Gleaning may not get disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled.
8. The IP address being outputted out the RS232 port and through the Announce (see page 149) module does not happen when the IP address is configured using IP Gleaning.
9. Multiple TX buffer support is not fully implemented in the MAC layer, ENC28J60.c. Stack behavior when TCP\_NO\_WAIT\_FOR\_ACK is undefined may be unexpected.

\*\*\*\*\*

v3.02.00: 20 Feb 2006

\*\*\*\*\*

Fixes:

1. Changed TXSTART in ENC28J60.c to stop wasting a byte.
2. Changed RXSTOP in ENC28J60.c to always be an odd value to properly implement an ENC28J60 silicon errata workaround.
3. Changed initialization of ERXRDP in MACInit() to agree with the current errata.

Changes:

1. Licence agreement
2. Schematics and other board files to the Ethernet PICtail Daughter Board have been updated to revision 5. Of significant note, the nRESET pin has been freed and 200 ohm resistors were added to the ENC28J60 SI, nCS, and SCK pins. The added resistors reduce undershoot caused by stray trace inductance and strong host output drivers.

Known Problems:

1. Testing on the PICDEM.net demo board with the Realtek RTL8019AS Ethernet controller has not been done. Moving to the HPC Explorer demo board has resulted in pinout and other hardware changes.
2. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.
3. SNMP, TFTPc, LCD, SLIP modules have not been tested with this version.
4. The stack may behave incorrectly if compiled using the Hitech compiler with a high optimizations setting.
5. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).
6. IP Gleaning may not get disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled.
7. The IP address being outputted out the RS232 port and through the Announce (see page 149) module does not happen when the IP address is configured using IP Gleaning.
8. Multiple TX buffer support is not fully implemented in the MAC layer, ENC28J60.c. Stack behavior when TCP\_NO\_WAIT\_FOR\_ACK is undefined may be unexpected.

\*\*\*\*\*

v3.01.00: 18 Jan 2006

\*\*\*\*\*

Fixes:

1. Implemented latest ENC28J60 silicon errata workarounds.
  2. Fixed a bug in TCP.c and UDP.c which would incorrectly write the packet checksum into the RX buffer incorrectly when
-

the checksum field was exactly spanning the RX wraparound boundary in the ENC28J60. This problem would have caused packets to be discarded in rare circumstances

#### Known Problems:

1. Testing on the PICDEM.net demo board with the Realtek RTL8019AS Ethernet controller has not been done. Moving to the HPC Explorer demo board has resulted in pinout and other hardware changes.
2. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.
3. SNMP, TFTPc, LCD, SLIP modules have not been tested with this version.
4. The stack may behave incorrectly if compiled using the Hitech compiler with a high optimizations setting.
5. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).
6. IP Gleaning may not get disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled.
7. The IP address being outputted out the RS232 port and through the Announce (see page 149) module does not happen when the IP address is configured using IP Gleaning.
8. Multiple TX buffer support is not fully implemented in the MAC layer, ENC28J60.c. Stack behavior when TCP\_NO\_WAIT\_FOR\_ACK is defined may be unexpected.

\*\*\*\*\*

v3.00.00: 16 Jan 2006

\*\*\*\*\*

#### Changes:

1. The stack now targets the PICDEM HPC Explorer demo board (PIC18F8722 MCU) with an attached Ethernet PICtail Daughter Board (with the Microchip ENC28J60 Ethernet controller).
2. IP Gleaning is no longer enabled (STACK\_USE\_IP\_GLEANING is not defined) by any of the default project files.
3. The IP address, whenever it changes, is outputted out the RS232 serial port in human readable form. Any terminal program, such as HyperTerminal can be used to read it. This allows the IP address to be easily determined when DHCP is used. The serial port defaults to 19200 baud when CLOCK\_FREQ in Compiler.h is properly defined.

#### Additions:

1. Microchip ENC28J60 Ethernet controller support. Support is included through the ENC28J60.c and ENC28J60.h files. Various other files were modified to take advantage of ENC28J60 specific features, like the hardware DMA/IP checksum engine. This new MAC driver incorporates several new functions which can be called from any layer above the MAC. The functions are: MACSetDuplex() MACPowerDown() MACPowerUp() MACSetPMFilter() MACDisablePMFilter() CalcIPBufferChecksum (see page 210)() MACCalcRxChecksum() MACCalcTxChecksum() MACCopyRxToTx() See the ENC28J60.c file comments for function descriptions. The ENC28J60.c file also incorporates TestMemory() which can do a power on self test of various hardware functions. TestMemory() is included and used when MAC\_POWER\_ON\_TEST is defined in StackTsk.h. It is undefined by default. Defining it will require some program memory.
2. Announce (see page 149) module. Announce.c and announce.h have been added. When included in the project, STACK\_USE\_ANNOUNCE must be defined. This module will broadcast a UDP message to port 30303 containing the local MAC address whenever the local IP address changes. This addition is intended to facilitate device discovery on DHCP enabled networks and eliminate the need for an RS232 connection if board reconfiguration is not needed. To retrieve the UDP message on your computer, use the new MCHPDetect.exe program included in the MCHPDetect subfolder.
3. The spieeprom.c file was added to support SPI EEPROM chips for MPFS storage. ENC28J60.c and spieeprom.c may both be included and they will share the same SPI module.

#### Improvements:

1. Renamed files/edited files so that the HI-TECH compiler won't raise messages stating that include files were spelled wrong.
2. Moved MAX\_ICMP\_DATA\_LEN from StackTsk.c to ICMP.h file for easier maintenance.
3. Corrected STACK\_USE\_SIIP typo in dhcp.c file - Thanks to Gisle J.B.

4. Implemented UDP checksum logic in UDPPProcess (see page 530)() in UDP.c file.
5. Renamed CalcTCPChecksum() in tcp.c file to CalcIPBufferChecksum (see page 210)().
6. Moved CalcIPBufferChecksum (see page 210)() to helpers.c to reuse it for UDP checksum calculation.
7. Modified UDPPProcess (see page 530)() in UDP.c and TCPProcess (see page 464)() in TCP.c to include localIP as third new parameter. This makes pseudo header checksum calculation correct in both functions. StackTsk.h, UDP.h and TCP.h files were also modified to reflect these changes.
8. Modified TCP.C file to include compile-time check of STACK\_USE\_TCP define. If it is not defined, an error will be displayed.
9. Removed an unnecessary call to MACDiscardRx() when an IP packet is received but fails version, options length, or header checksum tests.
10. Changed LCD code to be compile time removable by undefining USE\_LCD.

Fixes:

1. IPHeaderLen in IP.c is initialized properly now when IPGetHeader() is called.
2. Under some circumstances, HandleTCPSeq (see page 469)() would acknowledge, but throw valid received TCP packets away, resulting in loss of application data. An invalid comparison in HandleTCPSeq (see page 469)() has been fixed to prevent this situation from occurring. \*\*\* Thanks go to Richard Shelquist for identifying this problem.
3. Fixed StackTsk.c file so that if a static IP address is used and the LINK is removed, the node IP address is not cleared.
4. Invalid ICMP echo replies are no longer generated for echo requests with a data length of 33 (one more than the configured maximum).
5. Changed MAX\_OPTIONS\_LEN from 20 to 40. The maximum IP options length is now in agreement with the IP RFC.
6. Changed IPSetRxBuffer() from a macro to a function. The function takes into account any options which may be present in the header of received IP packets. Previously, possible options were not taken into account when calculating the offset.

Known Problems:

1. Testing on the PICDEM.net demo board with the Realtek RTL8019AS Ethernet controller has not been done. Moving to the HPC Explorer demo board has resulted in pinout and other hardware changes.
2. Sometimes when the FTP sever is used, an attempt to put a file is unsuccessful. The problem may be caused when an HTTP request to GET a file is made at the wrong time.
3. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.
4. SNMP, TFTPc, LCD, SLIP modules have not been tested with this version.
5. The stack may behave incorrectly if compiled using the Hitech compiler with a high optimizations setting.
6. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).
7. IP Gleaning may not get disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled.
8. The IP address being outputted out the RS232 port and through the Announce (see page 149) module does not happen when the IP address is configured using IP Gleaning.
9. Multiple TX buffer support is not fully implemented in the MAC layer, ENC28J60.c. Stack behavior when TCP\_NO\_WAIT\_FOR\_ACK is defined may be unexpected.

\*\*\*\*\*

v2.20.04.01: 9/24/03

\*\*\*\*\*

1. Recreated MPLAB projects to avoid problems when source is not at MCHPStack location.

\*\*\*\*\*

v2.20.04: 9/5/03

\*\*\*\*\*

Fixes:

1. Modified DHCPReset() in DHCP.c to not reset DHCP state machine if it was previously disabled using DHCPDisable(). This would make sure that if DHCP module was enabled and application had run-time disabled DHCP and network cable is disconnected, stack will not clear its IP address.
2. Rebuilt mib2bib.exe file with static library options. This fixes problem where one tries to execute this exe, an error occurs about missing DLLs.

\*\*\*\*\*

v2.20.03:

\*\*\*\*\*

Improvements:

1. When DHCP is enabled, LINK is monitored and IP address is reset on disconnect. New IP configuration is obtained on LINK reconnect. - For RealTek only. Modified DHCP.c to add DHCPReset() Modified MAC.c to add MACIsLinked() Modified StackTsk.h to add BYTE\_VAL def.

Changes:

1. Modified SMSC91c111.c to add empty MACIsLinked() - will be populated in next rev.

Bug Fixes:

1. Corrected DHCP logic to accept (see page 163) first DHCP offer instead of second response.
2. Corrected DHCP logic to check for chaddr in DHCP offer and accept (see page 163) one that matches with local MAC address. This will fix problem where if multiple nodes were on bus and all requested DHCP address, all would accept (see page 163) response from one server instead of verifying who was intended node.
3. Fixed UDPClose (see page 521)() in UDP.c to use INVALID\_UDP\_PORT (see page 518) instead of INVALID\_UDP\_SOCKET (see page 518) because of which a closed socket would not be scanned correctly.
4. Modified UDP.h to use long constant designators for INVALID\_UDP\_OPRT to explicitly state that it is a long.

\*\*\*\*\*

v2.20.02:

\*\*\*\*\*

Beta version containing TFTP client module.

Addition:

1. TFTP Client module - See TFTPc.\* and TFTPcDemo.c for more information. See MpTFTPcDemo and HtTFTPcDemo projects for build information.

Bug Fix:

1. UDPIsGetReady (see page 523)() was modified to overcome compiler rule where only 8-bit value was used to evaluate non-zero condition.
2. ARPResolve (see page 152)() in ARPTsk was fixed to clear Cache.IPAddr value.

\*\*\*\*\*

v2.20.01:

\*\*\*\*\*

Bug fix:

1. Fixed SMSC91C111.c where MACInit() would hang if ethernet link is not detected.

\*\*\*\*\*

v2.20:

\*\*\*\*\*

Bug Fixes:

1. General - Removed most of harmless warnings.
2. C18Cfg.asm - Fixed "include" instead of "define".
3. DHCP.c - Increased DHCP\_TIMEOUT\_VAL to 2 seconds. Fixed problem where UDP active socket was not set before calling UDP

functions in SM\_DHCP\_BROADCAST state.

4. MAC.c - Fixed MACIsTxReady() where under heavy traffic it would always return FALSE. This fixes bug where all high level applications would stop transmitting.
5. TCP.c - Enabled portion of code that performs timeout logic even if TCP\_NO\_WAIT\_ACK is defined. This fixes bug where occasionally, tcp applications such as HTTP server would stop working after few hours.
6. UDP.c - Fixed UDPGet (see page 522)() where it would return FALSE on last good byte. Fixed UDPProcess (see page 530)() where it was calculating incorrect length.

Added bFirstRead flag with UDP sockets similar to TCP sockets so that whenever first UDP byte is read, MAC read pointer will be reset to beginning of correct packet. This change fixes problem where if one transmits a packet while UDP packet is pending in a socket, next get to pending UDP socket would return wrong data. (This is apparent only when there is heavy network traffic)

Known Issues:

1. HiTech v8.20 PL4 with all optimization enabled may not work properly.
2. C18 "Static" and "Auto" mode may not be used - there are too many local variables to fit in standard stack of 256 bytes. One may modify linker script file to avoid this limitation.

Improvements:

1. Modified TICK def. in Tick.h to unsigned long to support 32-bit wide SNMP tick.
  2. Added SNMP Module (SNMP.c)
  3. Added Two new demo projects - DemoSNMPApp and HtDemoSNMPApp.
  4. Created MPLAB 6.X projects for different demo configurations.
  5. MAC.c - Added MACGetTxOffset().
  6. MPFS.c - Added MPFSSeek (see page 277)(), MPFSTell (see page 282)().
  7. MPFSImg.\*- Rebuilt to reflect v2.20, footprint changes etc.
  8. StackTsk.h- Enhanced WORD\_VAL, DWORD\_VAL defs. Added STACK\_USE\_SNMP and related compile-time checks.
-

9. UDP.h - Added UDPSetTx and UDPSetRx macros. Moved UDP\_SOCKET\_INFO (see page 534) structure to header file.
10. WebSrvr.c- Modified MCHPStack version message and added DATE info to BoardSetup menu.
11. Added support for SMSC LAN91C111 10/100 Non-PCI ethernet controller Use "SMSC91C111.C" instead of MAC.c. "mpnicee\_smsc" is a sample project that uses PIC18F8720 and SMSC NIC. "MasterDemo.c" is a main source file for above project that includes all modules - must use device with more than 32KB of memory.

\*\*\*\*\*

v2.11:

\*\*\*\*\*

Bug Fixes:

1. Fixed dhcp.c to make it work with new C18 startup code.

Improvements:

1. Modified webservr.c DownloadMPFS() to make use of compiler allocated XMODEM data block rather than use fixed address block starting at 0x400.

\*\*\*\*\*

v2.10: 7/9/02

\*\*\*\*\*

Bug Fixes:

1. Fixed HTTP Server bug where a form submission with empty parameter value would not parse correctly.

\*\*\*\*\*

v2.0: 5/22/02

\*\*\*\*\*

\*\*\*\*\*

New Modules:

\*\*\*\*\*

1. Added UDP, DHCP, FTP and IP Gleaning
2. Added PICDEM.net LCD support
3. Added board setup through RS-232.

\*\*\*\*\*

Improvements:

---

\*\*\*\*\*

- 1. Optimized serial EEPROM access routines in terms of speed and size (Replaced ee256.\* files with eeprom\*.h)
- 2. Improved board setup through RS-232.

\*\*\*\*\*

Known Issues:

\*\*\*\*\*

- 1. LCD may not display properly on MCLR only. Workaround:
  - 1. Debug XLCDInit() routine in "xlcdlh"
  - 2. Always do POR reset.
  
- 2. SLIP connection is not very robust. Workaround: None at this time.
  
- 3. Hi-Tech Compiler:
  - 1. Aggressive optimization breaks the functionality. Workaround: Apply optimization listed in each source file comment header.
  - 2. In order to use V8.12, you will need to remove "FTP Server" from Ht\*.pjt. You will also need to disable all optimizations.
  
- 4. C18 Compiler:
  - 1. Static model does not compile. Workaround: None at this time.
  - 2. Overlay model breaks the functionality. Workaround: None at this time.
  - 3. All modules does not fit in 32KB memory. Workaround:
    - 1. None at this time.
    - 2. Sample project disables some modules.

\*\*\*\*\*

New Files:

\*\*\*\*\*

File	Purpose
------	---------

- 1. delay.\* Provides CLOCK\_FREQ depenent delay routines.
- 2. dhcp.\* DHCP client support
- 3. ftp.\* FTP server
- 4. udp.\* UDP socket support
- 5. xeprom.\* Improved ee256.\* and renamed.
- 6. xlcd.\* External LCD support.
- 7. version.log To track changes and history.

\*\*\*\*\*

Changes:

\*\*\*\*\*

```

=====
===== File Change To-do for v1.0 stack applications
=====
=====

```

- 1. arptsk.c 1. Fixed STACK\_CLIENT\_MODE compile errors.  
None
- 2. Modified ARPisResolved (see page 153)() to support IP Gleaning  
None
  
- 2. c18cfg.asm 1. Added PIC18F452 configuration  
None
- 2. Fixed "include" errors.  
None
  
- 3. compiler.h 1. Included "stdlib.h" in both C18 and Hi-Tech compilers.  
None
- 2. Moved CLOCK\_FREQ from "stacktsk.h" to this file.  
None
- 3. Added PORTA defs.  
None
  
- 4. htnicee.pjt 1. Removed "ee256.c".  
None
- 2. Added "udp.c", "dhcp.c", "ftp.c", "xlcd.c", "xeprom.c" files  
Add these files if needed.
  
- 5. htnicpg.pjt None
  
- 6. htslee.pjt 1. Removed "ee256.c".  
None
- 2. Added "ftp.c", "xlcd.c", "xeprom.c" files  
None
  
- 7. http.c 1. Included compile-time verification that HTTP module is included.  
None
- 2. Put HTTP message strings into one array "HTTPMessages".

None

3. Modified to return "Service Unavailable" message if MPFS is being  
None remotely programmed.

4. Modified SendFile() to make use of sequential EEPROM read.

None

8. ip.c 1. Added one more paramter to IPGetHeader() to support IP Gleaning  
Custom apps using IP needs to be  
modified.

9. mac.c 1. Replaced fixed delay routines with CLOCK\_FREQ dependent  
None routines

10. mpfs.c 1. Replaced ee256.h with xeprom.h.

None

2. Added MPFSFormat (see page 268), MPFSPut() etc. routines

None

3. Added sequential read and page write operations

Custom apps using MPFS directly

needs to be modified.

4. Defined MPFS\_WRITE\_PAGE\_SIZE (see page 281) for MPFSPut operations.

Apps using different EEPROM page size

needs to be modified.

11. mpnicee.pjt 1. Removed "ee256.c"

None

2. Added "xcl.d.c", "xeprom.c" files

None

12. stacksk.c 1. Replaced ee256.h with xeprom.h

None

2. Added IP Gleaning and DHCP support.

None

13. stacksk.h 1. Moved CLOCK\_FREQ to compiler.h

None

2. Added STACK\_USE\_DHCP, STACK\_USE\_FTP\_SERVER etc. options

None

3. Added compile-time enable/disable of modules based on selection of higher level modules.

None

4. Modified MY\_DEFAULT\_MAC\_BYTE? to use Microchip OUI id.

None

5. Added compiler-time check to confirm available TCP sockets

None

6. Added MSB and LSB macros.

None

7. Added SerialNumber etc. to AppConfig structure

None

8. Commented module selection defines: They are defined by compiler

None command-line options. Real application should define them here in this file.

14. tcp.c 1. Moved TCP\_STATE (see page 462) and TCP\_INFO to .h file.

None

2. Fixed TCPIsConnected (see page 449)()

None

3. Fixed TCPDisconnect (see page 442)()

None

4. Modified TransmitTCP() to set receive window of one segment

None

5. Modified TransmitTCP() to use max segment size equal to predefined value.

None

6. Improved TCP State machine

None

15. tick.c 1. Modified TICK type to 16-bit.

None

2. Made use of TICK\_PRESCALE\_VALUE

None

3. Added code to blink PICDEM.net "System LED"

Remove if not required.

16. webservr.c 1. Added LCD support

N/A

2. Made TickUpdate (see page 514)() on Timer0 interrupt

N/A

3. Added code to save/restore board configuration

N/A

4. Added board setup via RS-232.

N/A

5. Added call to FTP modules

If needed, add this.

## 3.1 Stack Performance

Note that this table will not appear in the PDF version of the help file; see the "TCPIP Stack Performance.htm" file in the TCPIP documentation folder in the Microchip Application Library help folder.

## 3.2 Memory Usage

These tables contain the PIC program and data memory requirements for the TCP/IP stack. The first two rows list the program memory consumption of the stack's required files (see page 131), and each additional row contains the additional memory required to implement specific modules. These values are approximations; the program memory size may increase depending on application code, or decrease based on optimizations of modules with overlapping code. Modules that require user-implemented API functions (SNMP, HTTP) are tested without additional code. The global data memory column includes only the RAM needed for the required structures in the stack; it does not include the memory used for socket allocation (see page 146).

The C18 code uses the PIC18F97J60 family Ethernet controller as the MAC/PHY chip; the C30 and C32 measurements are made using the ENC28J60 Ethernet controller (ENCX24J600 sizes are similar). All compilers include a separate Required Stack Code line for Wi-Fi applications using the MRF24WB0M as the network controller. These two Required Stack Code lines are mutually exclusive -- do not add them together. Instead, chose the line representing your network controller.

These values are approximations obtained from TCP/IP Stack version 5.31. Note that these tables will not appear in the PDF version of the help file; see the "TCPIP Cxx Memory Usage.htm" files in the TCPIP documentation folder in the Microchip Application Library help folder.

**C18**

**C30**

**C32**

## 3.3 Peripheral Usage

Several microcontroller peripherals can/must be used to implement a TCP/IP stack application.

Type	Specific/Configurable	Polled/Interrupt	Purpose
Timer	Timer 0 for PIC18, Timer 1 otherwise	Interrupt.	Used to implement a tick timer
SPI or PMP	Select via #define in HardwareProfile.h. See Hardware Configuration (see page 136).	Polled.	The SPI module is used to drive the ENC28J60 or MRF24WB0M. An ENCX24J600 can be driven by the SPI module or a PMP module.
SPI	Select via #define in HardwareProfile.h. See External Storage (see page 136).	Polled.	Used to interface to an EEPROM or Serial Flash chip, as an option to store web pages for MPFS/MPFS2 (see page 265) or the AppConfig (see page 132) structure.

SPI	Select via #define in HardwareProfile.h. See External Storage ( <a href="#">see page 136</a> ).	Polled.	Used to interface to a serial RAM as a optional socket ( <a href="#">see page 146</a> ) allocation method.
-----	---	---------	--

## 4 Silicon Solutions

One of the first choices to make when designing your application is which hardware layer to use. Microchip supports a number of hardware TCP/IP solutions, each with an integrated MAC and/or PHY. The ENC28J60 and ENCX24J600 are stand-alone Ethernet controller chips, developed by Microchip Technology. The MRF24WB0M is a stand-alone 802.11b wireless transceiver. The PIC18F97J60 is a PIC18 microcontroller with an integrated Ethernet peripheral. The PIC32MX7XX/6XX series of 32-bit microcontrollers are high performance devices with integrated Ethernet MAC peripheral (MII/RMII interface to external PHY).

For information about demonstration boards that use these devices, see the Demo Kits (see page 62) section.

Feature	ENC28J60	ENCX24J600	PIC18F97J60	MRF24WB0M	PIC32MX7XX/6XX
Technology	Wired Ethernet	Wired Ethernet	Wired Ethernet	802.11 Wireless	Wired Ethernet
MAC	Internal	Internal	Internal	Internal	Internal
PHY	Internal (10-Base-T)	Internal (10/100-Base-T)	Internal (10-Base-T)	Internal	External PHY (MII/RMII Interface)
RAM Buffer (bytes)	8,192	24,576	3,808	14,170	Configurable descriptors in Internal RAM (128k of Internal RAM)
Interface	SPI	SPI, 8 or 16 bit multiplexed or demultiplexed parallel interface	None (built-in Ethernet MAC/PHY)	SPI	None (built-in Ethernet MAC)
Pins	28	44, 64	64/80/100	36	64/100/121
Package	SOIC, SPDIP, SSOP, QFN (6x6 mm)	TQFP, QFN	TQFP	Surface Mount WiFi I/O module	TQFP, QFN (9x9 mm), BGA
Cryptographic Engines	No	Yes	No	No	No
Pre-programmed MAC address	No <sup>(1)</sup>	Yes	No <sup>(1)</sup>	Yes	Yes

1: For devices without a pre-programmed MAC address, you may consider using an EEPROM with a built-in MAC address, such as the device family described here (see page 141).

# 5 Software

This section will discuss the computer software applications included with Microchip's TCP/IP Stack.

These tools are implemented using the C# or Java programming languages, or both. The C# tools (\*.exe) will require the Microsoft® .NET Framework v2.0 to be installed on the local PC. The Java tools (\*.jar) require Java Runtime Environment (JRE) 1.6 or later to be installed on the target computer.

## 5.1 TCP/IP Configuration Wizard

The TCP/IP Configuration Wizard is the easiest, safest way to set up firmware (and some hardware) configuration options. It will read and parse configuration settings from a copy of `TCPIPConfig.h` and then provide a graphical user interface that will easily allow you to view and modify those settings. In addition, if a feature that you enable will require another resource or feature to operate, the additional features will be enabled automatically. The TCP/IP Configuration Wizard will be installed to the Start menu when the TCP/IP Stack is installed.

When you launch the configuration wizard, you will be prompted to enter the path to a copy of `TCPIPConfig.h` and given the opportunity to modify advanced configuration settings. The advanced setting option will give more precise control over stack features, but will also require a greater working knowledge of Microchip's TCP/IP Stack.



## 5.2 MPFS2 Utility

The MPFS2 Utility packages web pages into a format for efficient storage in an embedded system. It is a graphical application for PCs that can generate MPFS2 and the older MPFS Classic images for storage in external storage or internal Flash program memory.

When used to build MPFS2 images, the MPFS2 Utility also indexes the dynamic variables found. It uses this information to generate `HTTPPrint.h`, which ensures that the proper callback functions are invoked as necessary. It also stores this index information along with the file in the MPFS2 image, which alleviates the task of searching from the embedded device.

Finally, when developing an application that uses external storage, the MPFS2 Utility can upload images to the external storage device using the upload functionality built into the HTTP2 web server (for MPFS2) or FTP server (for MPFS Classic).

## 5.2.1 Building MPFS2 Images

The MPFS2 Utility has four steps, which are denoted on the left hand side of the dialog. To build an MPFS image, select **Start With: Webpage Directory** in step 1 and choose the directory in which the web pages are stored.

Step 2 selects the output format. If storing the web pages in external EEPROM or serial Flash, choose the **BIN Image** output format. If internal program memory will be used, select **C18/C32 Image** for use with 8-bit and 32-bit parts, or **ASM30 Array** for 16-bit targets. To store the web pages on a device formatted with the FAT file system without compressing them into an MPFS image, select **MDD** (see the **Demo App MDD Getting Started** guide for more information).

Step 3 asks for the MPLAB IDE project directory. The MPFS tool will write the image file to the project directory, and will also update the `HTTPPrint.h` file there if needed. Select the correct directory so that the right files are modified.

Step 4 controls the upload settings. When external EEPROM or serial flash is used for storage, the option to upload the newly created image to the board is available. Check the box next to **Upload Image To** to enable this feature. The **Settings** button on the right can be used to configure the correct host name.

If internal program memory is being used, the image will be compiled in with the project and so direct uploads are not available. Make sure to include the output source file indicated in step 3 as part of the project.

Once all the correct settings have been chosen, click the **Generate** button to create the image. If uploads are enabled, this will also attempt to upload the file to the device.

## 5.2.2 Uploading Pre-built MPFS2 Images

There are two ways to upload a pre-built image to external storage. The first is described in the Getting Started (see page 72) section, and involves uploading from the browser directly. The second is to use the MPFS2 Utility to upload the image. You can select HTTP or FTP uploading to match the protocol that your application uses.

To use the MPFS2 Utility to upload an image, begin by selecting **Start With: Pre-Build MPFS Image** in step 1 at the top. Choose the image file to upload.

Steps 2 and 3 are not required for pre-built images. Proceed directly to step 4 and verify that the upload settings are correct. The host name may need to be changed to the one chosen when the board was first configured. Use the **Settings** button to edit these values.

Once all the settings are correct, click the **Upload** button. The image will be uploaded to the board.

## 5.2.3 Advanced MPFS2 Settings

The **Advanced Settings** dialog found in step 2 provides greater control over how files are processed.

The **Dynamic Files** list indicates which file types to parse for dynamic variables. By default, all files with the extensions htm, html, cgi, or xml are parsed. If an application has dynamic variables in other file types, these types must be added to the list. This field must be a comma-separated list of extensions and file names.

The **Do Not Compress** field indicates which file types should never be compressed. Compressing files with GZIP saves both storage space and transmission time. However, this is only suitable for static content such as CSS or JavaScript. Any files with dynamic variables will automatically be excluded. In addition, any file that the PIC may need to process internally should be excluded. Files included via ~inc:filename~ should not be compressed, nor should any BIB file used for the SNMP module (if present). Additional file types can be added to this list if a custom application will be accessing the MPFS.

The GZIP compressor will attempt to shrink all files. In some cases, especially with images, little or no compression is achieved. When this occurs the file is stored as-is in the MPFS image.

## 5.2.4 MPFS2 Command Line Options

To facilitate batch files and automation, the MPFS2 Utility also supports execution from the command line. The syntax is as follows:

```
MPFS2.jar [options] <SourceDir> <ProjectDir> <OutputFile>
```

The **SourceDir**, **ProjectDir**, and **OutputFile** options are required and should be enclosed in quotation marks. The **OutputFile** option will be relative to **ProjectDir**, and **cannot** be a full path name.

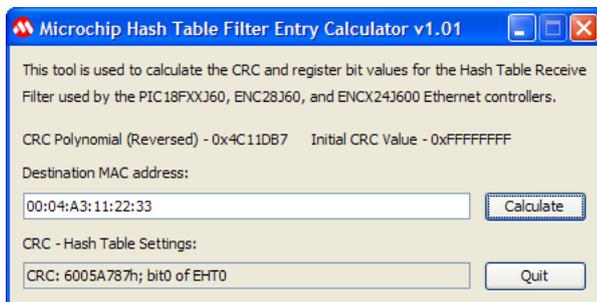
The various option switches are described in the table below:

Switch	Short	Description
/BIN	/b	Output a BIN image (Default)
/C18_C32	/c	Output a C18 or C32 image
/ASM30	/s	Output an ASM30 image
/mpfs2	/2	Use the MPFS2 format (Default)
/html "..."	/h "..."	File types to be parsed for dynamic variables (Default: "*.htm, *.html, *.cgi, *.xml")
/xgzip "..."	/z "..."	File types to be excluded from GZIP compression (Default: "*.bib, *.inc")

The command-line interface does not support image uploads. For batch or production uploads, use a tool such as `wget` to upload the generated BIN image.

## 5.3 Hash Table Filter Entry Calculator

This Hash Table receive filter on the ENC28J60, ENCX24J600, and PIC18F97J60 microcontroller family performs a CRC calculation over the six destination address bytes in a received packet, then uses that value as a pointer into the EHT0-EHT7 registers. If the bit that the pointer points to is set, the packet will be received. The Microchip Hash Table Filter Entry Calculator will determine the bit that must be set in this register bank for a given destination address. If you have a fixed MAC address, known at design time, you can set up your Hash Table receive filter in your code using the value obtained from this tool; otherwise, you must use the `SetRXHashTableEntry()` function to set it during runtime. To use this tool, specify the address of your device, click calculate, and the CRC value and the corresponding bit will be displayed in the output box.

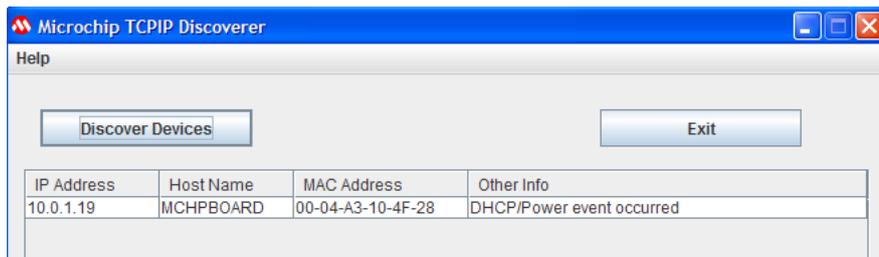


## 5.4 Microchip TCP/IP Discoverer

The Microchip TCP/IP Discoverer PC project (formerly known as the Embedded Ethernet Device Discoverer) will aid in embedded product device discovery (with the Announce (see page 149) protocol) and will demonstrate how to write PC applications to communicate to embedded devices.

When the "Discover Devices" button is clicked, this application will transmit a broadcast UDP packet containing the message, "Discovery: Who is out there?" on the local network to port 30303. If any embedded devices with the Announce (see page 149) protocol enabled are connected to the network, they will respond with a UDP packet containing their host name (NBNS (see page 284)) and MAC address.

The Java source code for this application is also included. This source code should provide a rough idea of how to write a PC-based application to communicate with your embedded devices.



# 6 Getting Started

This section describes the steps necessary to begin using Microchip's TCP/IP Demo Applications. This section contains specific information for setting up and using the generic TCPIP Demo App ([see page 81](#)). Most of this setup information can be applied to get started with other demo applications as well.

## 6.1 Hardware Setup

The first step to use the stack is to make sure an appropriate development board is configured. To get started, select a platform from the topics presented below.

### 6.1.1 Daughter Boards

Microchip offers four daughter boards that provide different Ethernet functionality to available demo boards. Each board is designed with:

- A PICtail™ connector, which enables an interface to the PICDEM.net 2 ([see page 63](#)) or the PIC18 Explorer ([see page 65](#)) board (populated with a PIC18 processor)

and/or

- A PICtail Plus connector, which will allow it to interface to an Explorer 16 ([see page 66](#)) development board (populated with a PIC24, dsPIC33, or PIC32 processor) or a PIC32 Starter Kit ([see page 66](#)).

Note that the PICDEM.net 2 is populated by default with an ENC28J60 and a PIC18F97J60.

#### Ethernet PICtail Daughter Board



The Ethernet PICtail Daughter board is populated with an ENC28J60, an RJ-45 connector (with integrated magnetics), and the few other components required for Ethernet operation. It provides a 10-Base-T Ethernet connection for any demo board with a PICtail connector. This daughter board has been largely superseded by the PICDEM.net 2 ([see page 63](#)) for debugging Ethernet applications using the PIC18. Visit the Microchip web site to view the Ethernet PICtail [Product Page](#).

#### Ethernet PICtail Plus Daughter Board



The Ethernet PICtail Plus Daughter Board is the PICtail Plus version of the Ethernet PICtail Daughter Board. It allows the

interface of an ENC28J60 to any demo board with a PICtail Plus connector. Visit the Microchip web site to view the Ethernet PICtail Plus Daughter Board [Product Page](#).

#### Fast 100Mbps Ethernet PICtail Plus Daughter Board



The Fast 100Mbps Ethernet PICtail Plus Daughter Board provides a method for testing and demonstrating the ENC624J600 Ethernet Controller. The board is designed for flexibility and can be connected to a PICtail or a PICtail plus connector. In addition, it is designed to allow the use of any of the parallel or SPI connection modes featured on the ENC624J600 on the PICtail Plus connector. This daughter board provides 10/100-Base-T functionality. Visit the Microchip web site to view the Fast 100Mbps Ethernet PICtail™ Plus Daughter Board [Product Page](#).

#### Microchip 802.11b WiFi PICtail Plus Daughter Board



The Microchip 802.11b WiFi PICtail Plus Daughter Board is a demonstration board for evaluating Wi-Fi connectivity on boards with a PICtail or a PICtail Plus connector. The board features the Microchip MRF24WB0MA module, which includes a Wi-Fi transceiver and associated circuit elements. Visit the Microchip Web Site to view more information on [Wireless Solutions](#) and the 802.11b WiFi PICtail [Product Page](#).

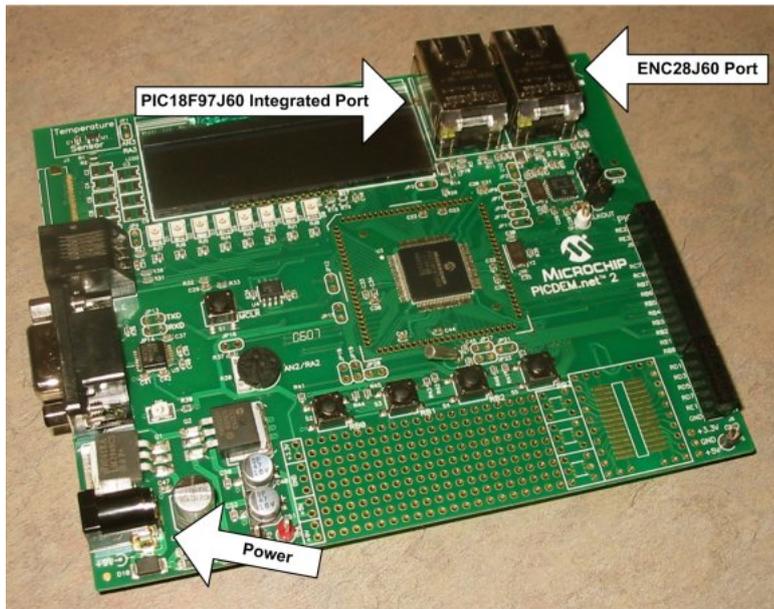
---

## 6.1.2 PICDEM.net 2

Visit the Microchip web site to view the PICDEM.net 2 [Product Page](#).

The PICDEM.net 2 development board comes populated with a PIC18F97J60 with an integrated Ethernet controller, as well as a standalone ENC28J60 Ethernet controller. The integrated controller is connected to the left Ethernet jack (closest to the LCD), and the standalone part is connected to the right one. By default the stack is configured to use the integrated controller, so the **left port** should be connected to the network cable. No other configuration of the board is necessary.

The User's Guide that shipped with this development board may refer to an older version of the TCP/IP Stack. This document updates much of that documentation for version 5.36.



### Using the Fast Ethernet PICtail

By default, this board will use the ENC28J60 or the PIC18F97J60 for Ethernet communication. However, by connecting the Fast Ethernet PICtail to the PICtail connector on the board, you can use it to test the ENC624J600. To use the Fast Ethernet PICtail, insert it as shown in the picture, with header J4 on the PICtail inserted into connector J5 on the demo board.



The Fast Ethernet PICtail is designed to use the SPI communication bus when connected through a PICtail header, so the jumper settings are unused in this configuration, with one exception: the JP2 jumper on the PICtail, labeled ISENSE, should be shorted. The pre-compiled and pre-configured versions of the demo that correspond to this setup are already written to enable ENC624J600 functionality; for manual configuration information, see the ENC624J600 (see page 138) configuration page.

### Using the Microchip 802.11b WiFi PICtail

The PICDEM.net 2 can be used to debug wireless functionality by connecting the PICtail as show in the picture, with header J1 on the PICtail inserted into connector J5 on the demo board.



Note if jumper JP3 exists, it must be shorted between pins 2 and 3 when used on this development platform.

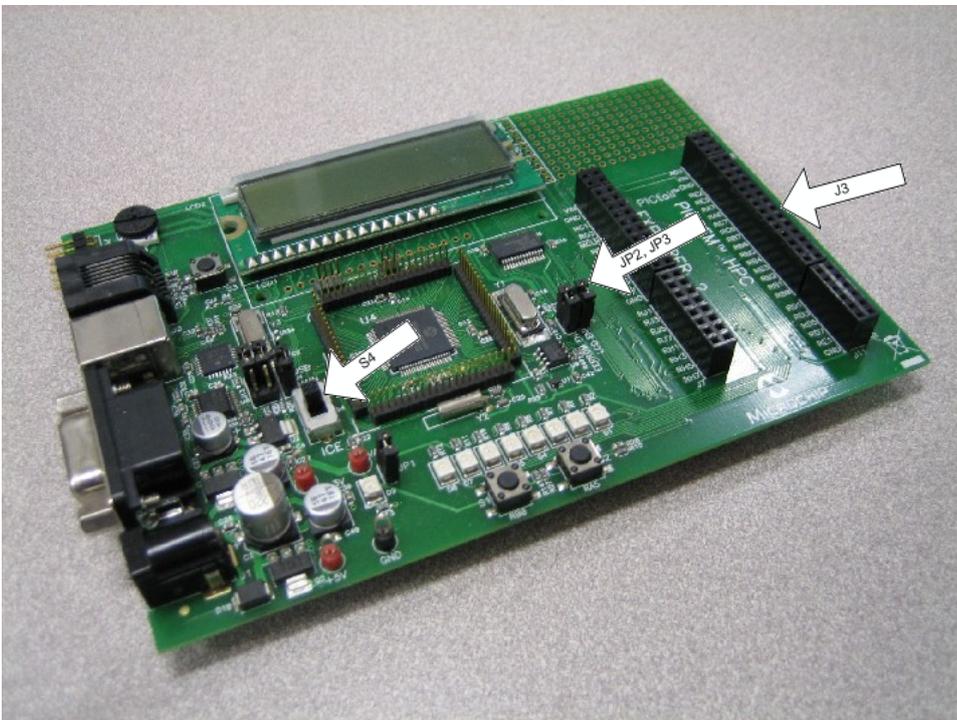
Once your hardware is configured, you can program your board with your preferred demo project. The next few topics (see page 69) in the Getting Started section of this help file provide a tutorial for setting up the generic TCPIP demo application.

## 6.1.3 PIC18 Explorer

Visit the Microchip web site to view the PIC18 Explorer [Product Page](#).

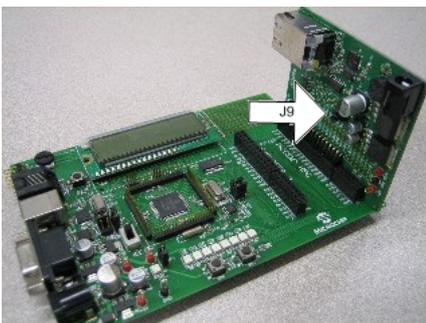
The PIC18 Explorer is for evaluation of high pin-count PIC18 microcontrollers. By connecting a TCP/IP daughter board to it, you can test and debug Ethernet functionality with a variety of PIC18s. The PIC18F97J60 family includes a built-in Ethernet peripheral, making it the default low-cost, PIC18 Ethernet development platform; the PICDEM.net 2 (see page 63) is the recommended development board for this part.

When using the PIC18 Explorer, ensure that jumpers JP2 and JP3 are shorted to enable the LCD and EEPROM, and switch S4 is configured to properly select the on-board PIC or the ICE setting, as your application requires.



### Using the Ethernet PICtail

Unlike the PICDEM.net 2, the PIC18 Explorer does not include an ENC28J60 on the board. To enable testing and debugging using the ENC28J60, you must connect (see page 165) an Ethernet PICtail, as shown in the picture (insert header J2 into connector J3 on the demo board).



When using this configuration, short pins 2 and 3 on jumper J9, to indicate that the PIC18 Explorer is providing a 5V power

supply. The pre-compiled and pre-configured versions of the demo that correspond to this setup are already written to enable ENC28J60 functionality; for manual configuration information, see the ENC28J60 (see page 137) configuration page.

### Using the Fast Ethernet PICtail

By connecting the Fast Ethernet PICtail to the PICtail connector on the board, you can use it to test the ENC624J600. To use the Fast Ethernet PICtail, insert it as shown in the picture, with header J4 on the PICtail inserted into connector J3 on the demo board.



The Fast Ethernet PICtail is designed to use the SPI communication bus when connected through a PICtail header, so the jumper settings are unused in this configuration, with one exception: the JP2 jumper on the PICtail, labeled ISENSE, should be shorted. The pre-compiled and pre-configured versions of the demo that correspond to this setup are already written to enable ENC624J600 functionality; for manual configuration information, see the ENC624J600 (see page 138) configuration page.

### Using the Microchip 802.11b WiFi PICtail

The PIC18 Explorer can be used to debug wireless functionality by connecting the PICtail as show in the picture, with header J1 on the PICtail inserted into connector J3 on the demo board.



Note if jumper JP3 exists, it must be shorted between pins 2 and 3 when used on this development platform.

Once your hardware is configured, you can program your board with your preferred demo project. The next few topics (see page 69) in the Getting Started section of this help file provide a tutorial for setting up the generic TCPIP demo application.

---

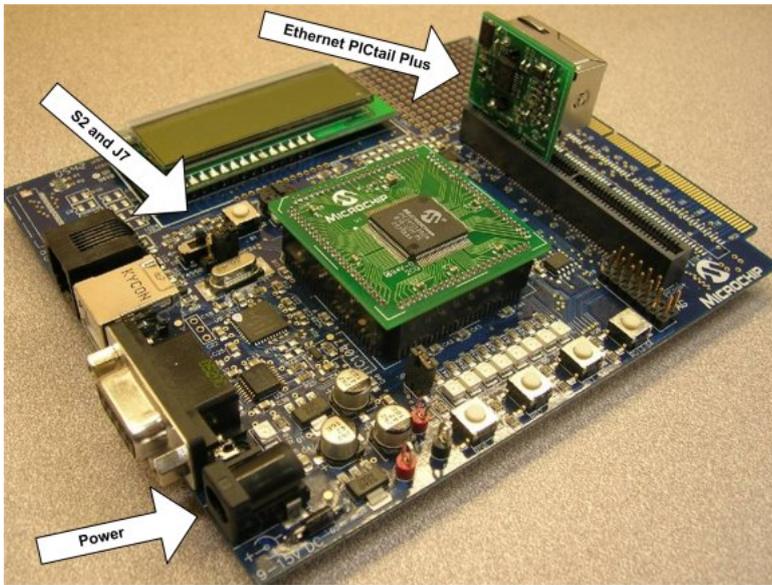
## 6.1.4 Explorer 16 and PIC32 Starter Kit

Visit the Microchip web site to view the Explorer 16 [Product Page](#) and the PIC32 Starter Kit [Product Page](#).

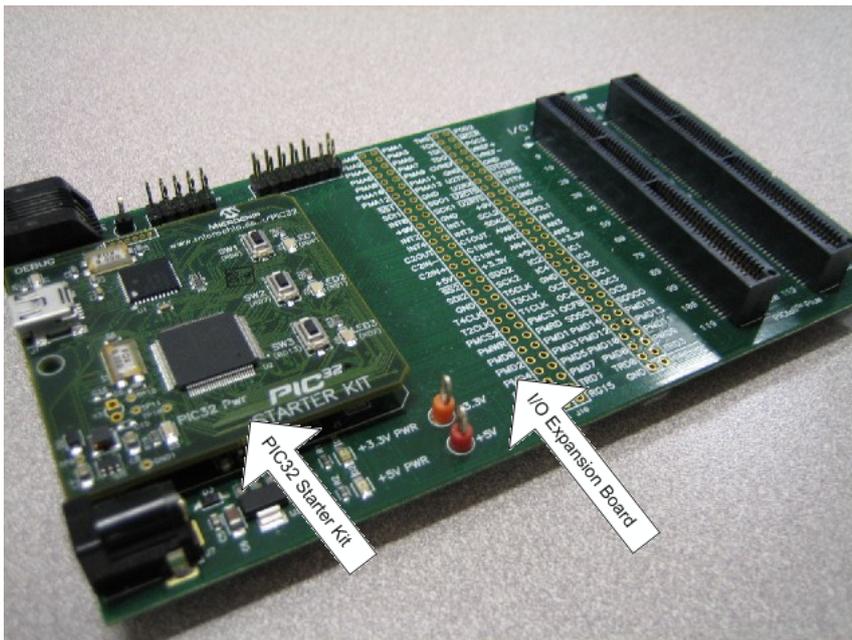
The Explorer 16 board is an all-purpose demonstration and development board for 16-bit and 32-bit parts. It can be expanded for TCP/IP support using the Ethernet PICtail Plus, Fast 100Mbps Ethernet PICtail Plus, or 802.11b WiFi PICtail Plus daughter board.

Before using the Explorer 16, check that:

1. Switch S2 selects PIM
2. Jumper J7 selects PIC24 (even though the label reads PIC24, this jumper setting selects the programming signals to any PIC on the Explorer 16).

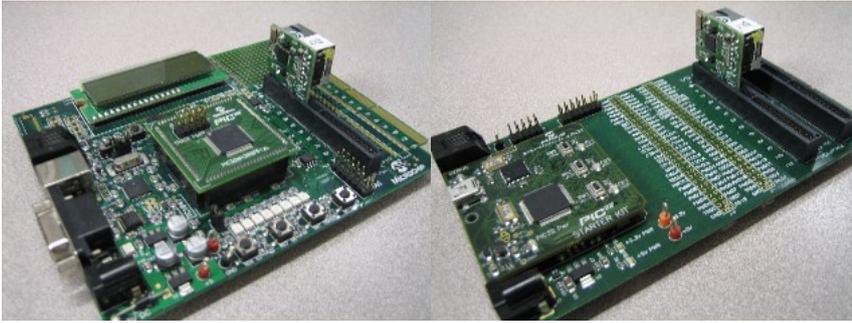


The PIC32 Starter Kit performs a similar function for 32-bit PIC32 parts. By using the [PIC32 I/O Expansion Board](#) you can connect (see page 165) the same PICtail Plus board that connect (see page 165) to the Explorer 16.



### Using the Ethernet PICtail Plus

To enable testing and debugging of the ENC28J60 on these boards, you must connect (see page 165) an Ethernet PICtail Plus, as shown in the picture (insert header J2 into the upper card-edge connector J5 (Explorer 16) or J4 (I/O Expansion Board)). Note that for some demos, the Ethernet PICtail Plus will need to be inserted into the center card-edge connector of the PICtail Plus connector to use the SPI2 module. See the Demo Compatibility Table (see page 77) for more information.



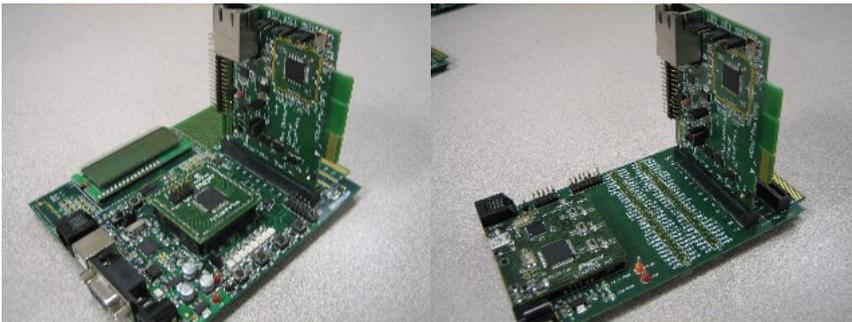
The pre-compiled and pre-configured versions of the demo that correspond to this setup are already written to enable ENC28J60 functionality; for manual configuration information, see the ENC28J60 (see page 137) configuration page.

### Using the Fast Ethernet PICtail Plus

By connecting the Fast 10/100 Ethernet PICtail Plus to the PICtail Plus connector on your board, you can use it to test the ENC624J600. The Fast Ethernet PICtail Plus can be used with these boards in either serial (SPI) or parallel communication mode. For serial mode, connect (see page 165) header J2 of the daughter board to connector J5 (Explorer 16) or J4 (I/O Expansion Board), as seen in the pictures. When operating in serial mode, the jumpers on the Fast Ethernet PICtail are unused, with one exception: the JP2 jumper on the PICtail, labeled ISENSE, should be shorted.



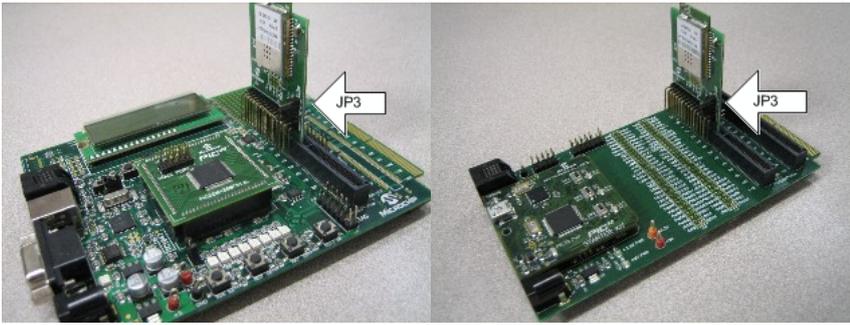
To use the Fast Ethernet PICtail Plus board in parallel mode, insert header J1 into connector J5 of the Explorer 16 or J4 of the I/O Expansion Board, as seen in the pictures. In this configuration, the jumpers must be shorted or opened corresponding to the parallel communication mode being used. A matrix outlining which jumper connections must be made for the jumpers labeled P<sub>SPCFG3</sub>, P<sub>SPCFG2</sub>, P<sub>SPCFG1&4</sub>, P<sub>MA to AD</sub>, and P<sub>MA to A</sub> is printed on the back side of the daughter board.



The pre-compiled and pre-configured versions of the demo that correspond to this setup are already written to enable ENC624J600 functionality; for manual configuration information, see the ENC624J600 (see page 138) configuration page.

### Using the Microchip 802.11b WiFi PICtail

The Explorer 16 and PIC32 Starter Kit can be used to debug wireless functionality by connecting the PICtail as show in the pictures, with header J2 on the PICtail inserted into the top slot of connector J5 (Explorer 16) or J4 (I/O Expansion Board) on the demo boards.



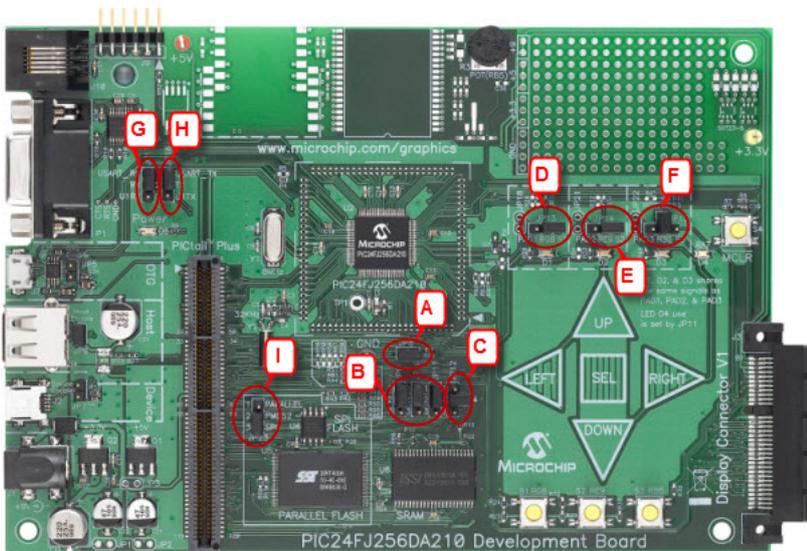
Note if jumper JP3 exists, it must be shorted between pins 1 and 2 when used on this development platform.

Once your hardware is configured, you can program your board with your preferred demo project. The next few topics (see page 69) in the Getting Started section of this help file provide a tutorial for setting up the generic TCP/IP demo application.

## 6.1.5 PIC24FJ256DA210 Dev Board

Visit the Microchip web site to view the PIC24FJ256DA210 Development Kit [Product Page](#).

The PIC24FJ256DA210 Development Kit is a low cost and efficient development kit to evaluate the features and performance of the PIC24FJ256DA210 with integrated graphics, mTouch™ and USB.



You can add network connectivity to this demo board by inserting an Ethernet PICtail Plus, Fast Ethernet PICtail Plus, or Microchip 802.11b WiFi PICtail into the PICtail Plus connector on the demo board. The method for doing this is functionally identical to the method used for the Explorer 16 and PIC32 Starter Kit (see page 66).

## 6.2 Programming and First Run

Once the hardware is configured (see page 62), you are ready to program the device for the first time.

### Project Setup

Open a session of the MPLAB IDE.

1. From the "File" menu, select "Import." Browse to the **Precompiled Hex** subdirectory in your demo project directory and select the \*.hex file that matches your hardware setup. The hex file names describe the hardware that the file has been compiled for. For example, the file "Microchip Solutions v2011-06-02\TCPIP\Demo App\Precompiled Hex\C18-PICDN2\_ETH97\_18F97J60.hex" corresponds to the generic TCP/IP Demo application for the PIC18F97J60 on the PICDEM.net 2, using the PIC's internal Ethernet module. A document enumerating the abbreviations used in the hex file and project file names is available in the **Microchip Solutions v20xx-xx-xx/Help** directory.
2. Verify that the MPLAB IDE processor target selection and linker script (if one is present) match the part on your demonstration board (ex: PIC18F97J60).

Note that the projects and source code used to build each hex file are present in the project directory. The hardware and firmware configuration files used to build each project are included in the **Configs** subdirectory.

### Programming

Select your device programmer from the Programmer menu in MPLAB, and then use the Program shortcut button or the Program menu option to program the code you imported to your board.

### Clearing the EEPROM

The TCP/IP Stack stores network configuration settings (such as the host name, MAC address, default static IP addresses, SNMP strings, WiFi network name (SSID), etc) in external EEPROM on the board. The demo project will detect if the default values have been changed in the EEPROM, and if so, use the new values. If not, the demo will use the default values configured in TCPIPConfig.h and WF\_Config.h. Checksums stored in the EEPROM are used to determine if the structures stored in EEPROM are valid. Manually clearing the EEPROM will allow the demo to resume using the default settings.

Use the following procedure to clear the EEPROM:

1. Make sure the development board is programmed and not in debug mode
2. Disconnect the MPLAB® ICD 2/3 or MPLAB REAL ICE™ from the board
3. Press and hold BUTTON0 (RD13/S4 on Explorer 16 or RB3/S5 on PICDEM.net™ 2)
4. Press and release the MCLR button
5. Continue holding BUTTON0 until several LEDs flash indicating that EEPROM has been cleared. This takes about 4 seconds.
6. Release BUTTON0
7. Press and release MCLR again to reset the software

Once you see LED0 (right-most LED) blinking, the software is running and ready for use.

If you are using the MRF24WB0M WiFi PICtail, you'll need to configure your wireless access point (☞ see page 70) first. For all Ethernet devices, Connect your Development Board (☞ see page 72) to your network.

---

## 6.3 Configure your WiFi Access Point

To run the Wi-Fi demos with the MRF24WB0M PICtail, you'll also need to setup a wireless access point. As an example, this guide will walk through the setup of a Linksys WRT54G2 access point.

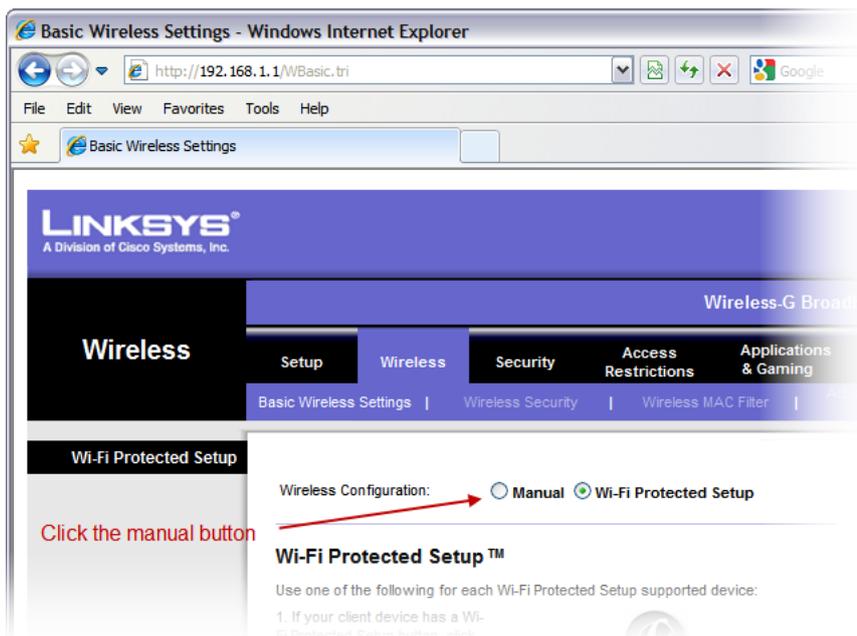
### Access Point Browser GUI

The Linksys, along with many other popular router brands, uses a built-in webserver on the router to administer the network (both wired and wireless). Please consult the documentation that came with your router for more information on configuration and setup. For a list of known compatible routers refer to section "Access Point Compatibility" (☞ see page 597). To gain access to this web page, you'll need to point your browser to <http://192.168.1.1>. By default, the username field is left blank, and the password is admin.



## Wireless Setup

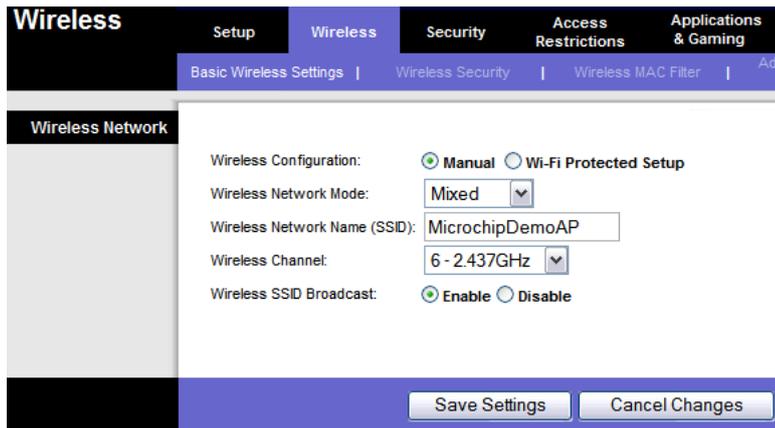
Along the top of the webpage, there should be many tabs for all the different features of the access point. One of the tabs should read "Wireless". After clicking the tab, you will be presented with the Wi-Fi protected setup page. You'll need to click the manual tab to be able to enter your own wireless settings to match the demo.



The out of box demo is looking for an AP with the following parameters (note that the SSID is case sensitive):

SSID	MicrochipDemoAP
Security	None
Channel	Either 1, 6, or 11

You should have settings similar to the following:



Once the network is setup, you can connect your device to the network (see page 72).

## 6.4 Connecting to the Network

All devices on a TCP/IP network must be assigned an IP address (see page 142). Whereas the MAC address (see page 141) is the hardware address of the device, the IP address is a software address. The DHCP (Dynamic Host Configuration Protocol) allows this assignment to take place automatically (for more address information and configuration options, see the Addresses (see page 141) topic).

The demo application comes with both a DHCP server and DHCP client configured. This allows the board to connect (see page 165) to most networks without configuration. If a free Ethernet port is available on a nearby router, switch, or wall plate, the board can be connected directly using any standard straight-through Ethernet cable. Under this configuration, the board will attempt to obtain an IP address from your network's DHCP server.

If this method is not possible, a crossover Ethernet cable can be used to connect (see page 165) the board directly to a PC's Ethernet port. Using this configuration, the board will act as its own DHCP server and will assign a single IP address to the computer. (The Fast 100Mbps Ethernet PICTail Plus and some newer PCs do not require a special crossover cable, so any Ethernet cable can be used.)

Connect the development board to the network and wait for the link LED on the Ethernet jack to light up. The board is now on the network and capable of communicating with other devices.

If the link LED on the Ethernet jack does not light, your board cannot link to the network. Ensure that you have selected the proper cable, and try switching from a straight-through to a crossover cable, or vice versa.

Now that the board is online, you can Upload the Demo Web Pages (see page 72).

## 6.5 Uploading Web Pages

Web pages are stored as an MPFS2 (see page 265) image. This image can be placed in either external non-volatile storage (see page 136) (EEPROM or SPI Flash), or in the microcontroller's internal Flash program memory. For this example, the EEPROM chip (25LC256) on your demo board will be programmed with a pre-built MPFS2 BIN image. This location can be changed via a compile-time option in TCPIPConfig.h.

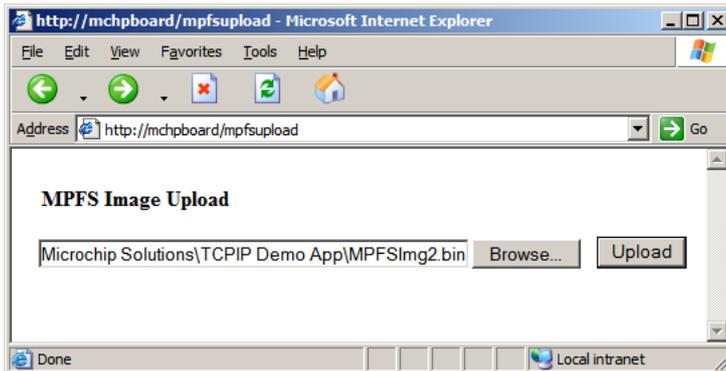
The target application on the development board must be running for this procedure to work. Make sure the right most status LED is blinking.

Each hex file is configured to provide a **Host Name** for your development board. This will be the name by which your board is accessed. In the default hex files, the host name is `mchpboard`, so you board can be accessed at `http://mchpboard`. This

host name uses the NetBIOS Name Service (see page 284). It is only available on your local subnet, and will not be accessible from the Internet. Note that this service is not supported by all operating systems. If you have difficulty accessing your board, try using the IP address shown on the LCD screen instead (e.g. access the board at <http://192.168.1.101>). You can also determine the IP address by using the Microchip TCP/IP Discoverer (see page 60).

Open a web browser and access the board at <http://mchpboard/mpfsupload>. This form will allow web pages stored on the device to be updated. If you mistype this URL, the board will provide a default HTTP 404 error page with a link to the MPFS Upload page. This default 404 page will not appear if you've configured your browser to override custom error pages (e.g. by checking "Show friendly HTTP error messages" in Internet Explorer 7's internet options menu). Select the file `MPFSImg2.bin` from the `TCPIP\Demo App` folder as shown below.

This update method is only available when using external storage.



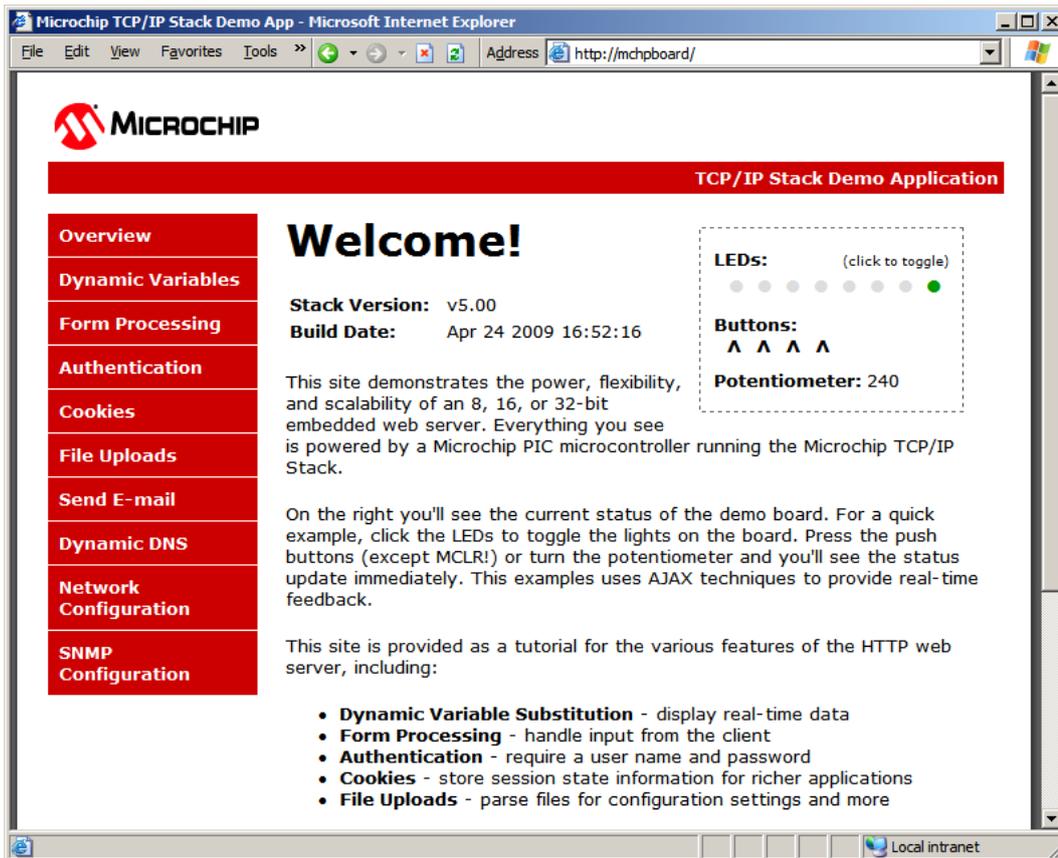
When the **Upload** button is clicked, the MPFS image is sent to the board and programmed into the EEPROM. As this happens, the activity LED on the Ethernet jack will blink. Once the browser reports that the upload has completed, click the link provided within the status message to access the board's web pages.

You can now Access the Demo Application (see page 73).

---

## 6.6 Accessing the Demo Application

The board is now accessible at the `mchpboard` host name or at the board's IP address. When accessed in a web browser, a real-time update of the board's controls is displayed. The demo application will show off several features, and will explain how to modify the web pages and application to suit various needs.



If you attempt to access the Network Configuration or SNMP Configuration web pages from the red menu on the left, you will be prompted for a username and password. The default username is "admin" and the default password is "microchip". More information is available on the Authentication (see page 84) web page, or in the HTTP2 server authentication (see page 230) help topic.

Some features of the default demo application may not be available on certain hardware platforms. For more information, see the TCPIP Demo App Features by Hardware Platform (see page 81) topic. For information about how to use each feature of the TCP/IP Demo Application, consult the subtopics in the TCP/IP Demo Application Demo Modules (see page 82) topic.

Once you have finished exploring the demo application, you can proceed to the Stack API (see page 149) section to learn more about the stack and start developing your own application.

If you are exploring the Wi-Fi demo applications and want to set up security, you can get more information on the WLAN security page (see page 74).

## 6.7 Configuring WiFi Security

The MRF24WB0M can be configured to connect (see page 165) to wireless networks with encryption enabled. The MRF24WB0M supports WEP (40-bit and 104-bit), as well as WPA (TKIP) and WPA2 (TKIP/AES).

### Device Security Modes

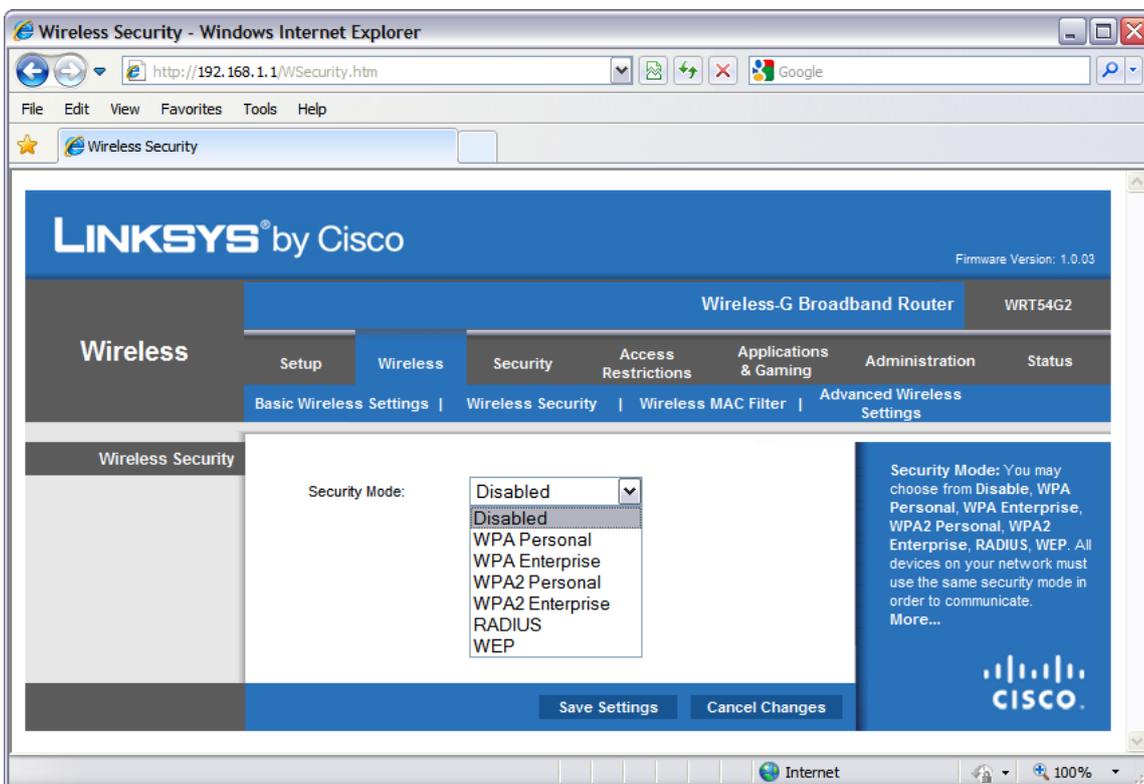
Security settings for the MRF24WB0M are located in the file `WF_Config.h`. To enable security features the `#define` preprocessor definition for `MY_DEFAULT_WIFI_SECURITY_MODE` must be defined as one of the following options:

WF_SECURITY_WEP_40	40-bit WEP security. This equates to 5 ASCII characters or 10 hex digits. MY_DEFAULT_WEP_KEYS_40 contains up to four keys that can be programmed (default is key 0).
WF_SECURITY_WEP_104	104-bit WEP security. This equates to 13 ASCII characters or 26 hex digits. MY_DEFAULT_WEP_KEYS_104 contains up to four keys that can be programmed (default is key 0).
WF_SECURITY_WPA_WITH_KEY WF_SECURITY_WPA2_WITH_KEY WF_SECURITY_WPA_AUTO_WITH_KEY	Uses the 32 bytes in MY_DEFAULT_PSK as the key to join the network. These values are generated from a hash of the SSID name and WPA passphrase. For the purpose of the demo, the 32-bytes in MY_DEFAULT_PSK in WF_Config.h correspond to an SSID of "MicrochipDemoAP" and passphrase "Microchip 802.11 Secret PSK Password".
WF_SECURITY_WPA_WITH_PASS_PHRASE WF_SECURITY_WPA2_WITH_PASS_PHRASE WF_SECURITY_WPA_AUTO_WITH_PASS_PHRASE	Instructs the MRF24WB0M to generate the 32 byte PSK using the SSID and passphrase. The default in WF_Config.h corresponds to an SSID of "MicrochipDemoAP" and passphrase "Microchip 802.11 Secret PSK Password". Note that it takes approximately 30 seconds for the MRF24WB0M to calculate this value <sup>[1]</sup> .

**Note:** Some routers try to increase the random nature of the WEP key by adding an additional layer that will convert an ASCII passphrase into a hexadecimal key. The MRF24WB0M PICtail will require a hexadecimal key, no matter which way it is generated.

**Access Point Security Settings**

The access point will also need to be changed to match the same security settings. Wireless security settings can be found in the "Wireless Security" tab under the main "Wireless" tab (example shows a Linksys WRT5G2). The drop-down box for security has all the different security options. Note that for WPA/WPA2, the MRF24WB0M only supports personal security levels (as opposed to enterprise, which is not supported).



[1]: Once the 32-byte PSK is calculated, it can be retrieved by the host from the MRF24WB0M. The host can then save this key to external non-volatile memory. On future connection attempts, the host can program the MRF24WB0M with the

WF\_SECURITY\_\*\_WITH\_KEY options, provide the saved key, and not have to wait 30 seconds to reconnect to the network.

**Pre-generated PSK**

You also have the option to pre-generate the PSK and use the 32-byte PSK directly in the source code. One handy tool to generate the PSK can be found online at the Wireshark Foundation <http://www.wireshark.org/tools/wpa-psk.html>. The Wireshark website can generate the expected 32-byte PSK key with the SSID name and the passphrase. You can then use these values in the variable MY\_DEFAULT\_PSK in TCPIPConfig.h.

# 7 Demo Information

This section describes Microchip's TCP/IP Demo projects, including information about demo-hardware compatibility. For information about how to load and configure the demos, please consult the Getting Started section.

## 7.1 Demo Compatibility Table

Each stack demonstration project comes with several predefined, tested configurations. Pre-built hex files for each demo are available in the **Precompiled Hex** subdirectory in that demo's project folder (i.e. the files for Demo App are located in <install directory>\Microchip Solutions v20xx-xx-xx\TCPIP\Demo App\Precompiled Hex). This section will specify the combinations of demo boards, processors, MAC/PHY layers, and communication buses that are set up to work by default.

### TCPIP Demo App (see page 81)

Demo Board	Processor	MAC/PHY Layer	Comm. Bus	Notes
PIC18 Explorer	18F87J11	ENC28J60	SPI	Requires silicon revision A4 or later.
PIC18 Explorer	18F87J11	ENCX24J600	SPI	
PIC18 Explorer	18F87J11	MRF24WB0M	SPI	
PIC18 Explorer	18F87J50	ENC28J60	SPI	
PIC18 Explorer	18F87J50	ENCX24J600	SPI	
PIC18 Explorer	18F87J50	MRF24WB0M	SPI	
PIC18 Explorer	18F8722	ENC28J60	SPI	
PIC18 Explorer	18F8722	ENCX24J600	SPI	
PIC18 Explorer	18F8722	MRF24WB0M	SPI	
PICDEM.net 2	18F97J60	ETH97J60	-	
PICDEM.net 2	18F97J60	ENC28J60	SPI	
PICDEM.net 2	18F97J60	ENCX24J600	SPI	
PICDEM.net 2	18F97J60	MRF24WB0M	SPI	
Explorer 16	24FJ128GA010	ENC28J60	SPI	
Explorer 16	24FJ128GA010	ENCX24J600	SPI	
Explorer 16	24FJ128GA010	MRF24WB0M	SPI	
Explorer 16	24FJ128GA010	ENCX24J600	PSP Indirect	5
Explorer 16	24FJ256GA110	ENC28J60	SPI2	
Explorer 16	24FJ256GA110	ENCX24J600	SPI2	
Explorer 16	24FJ256GA110	ENCX24J600	PSP Indirect	5
Explorer 16	24FJ256GA110	MRF24WB0M	SPI2	

Explorer 16		24FJ256GB110	ENC28J60	SPI	
Explorer 16		24FJ256GB110	ENCX24J600	SPI	
Explorer 16		24FJ256GB110	ENCX24J600	PSP Indirect	5
Explorer 16		24FJ256GB110	MRF24WB0M	SPI	
Explorer 16		24FJ256GB210	ENC28J60	SPI	
Explorer 16		24FJ256GB210	ENCX24J600	SPI	
Explorer 16		24FJ256GB210	ENCX24J600	PSP Indirect Bitbang	5
Explorer 16		24FJ256GB210	MRF24WB0M	SPI	
Explorer 16		33FJ256GP710	ENC28J60	SPI	
Explorer 16		33FJ256GP710	ENCX24J600	SPI	
Explorer 16		33FJ256GP710	ENCX24J600	PSP Indirect	5
Explorer 16		33FJ256GP710	MRF24WB0M	SPI	
Explorer 16		32MX360F512L	ENC28J60	SPI	
Explorer 16		33EP512MU810	ENC28J60	SPI2	
Explorer 16		24EP512GU810	ENC28J60	SPI2	
Explorer 16		32MX360F512L	ENCX24J600	SPI	
Explorer 16		32MX360F512L	ENCX24J600	PSP Indirect	5
Explorer 16		32MX360F512L	ENCX24J600	PSP 9	
Explorer 16		32MX360F512L	MRF24WB0M	SPI	
Explorer 16		32MX460F512L	ENC28J60	SPI	
Explorer 16		32MX460F512L	ENCX24J600	SPI	
Explorer 16		32MX460F512L	MRF24WB0M	SPI	
Explorer 16		32MX795F512L	ENC28J60	SPI	
Explorer 16		32MX795F512L	ENCX24J600	SPI	
Explorer 16		32MX795F512L	MRF24WB0M	SPI	
PIC24FJ256DA210 Board	Development	24FJ256DA210	ENC28J60	SPI	
PIC24FJ256DA210 Board	Development	24FJ256DA210	ENCX24J600	SPI	
PIC24FJ256DA210 Board	Development	24FJ256DA210	ENCX24J600	PSP Indirect Bitbang	5
PIC24FJ256DA210 Board	Development	24FJ256DA210	MRF24WB0M	SPI	
PIC32 General Purpose Starter Kit (DM320001)		32MX360F512L	ENC28J60	SPI	
PIC32 General Purpose Starter Kit (DM320001)		32MX360F512L	ENCX24J600	SPI	
PIC32 General Purpose Starter Kit (DM320001)		32MX360F512L	ENCX24J600	PSP Indirect	5
PIC32 General Purpose Starter Kit (DM320001)		32MX360F512L	MRF24WB0M	SPI	

PIC32 USB Starter Kit (DM320003_2)	32MX795F512L	ENC28J60	SPI2	
PIC32 USB (DM320003_2)	32MX795F512L	ENCX24J600	SPI2	
PIC32 USB (DM320003_2)	32MX795F512L	ENCX24J600	PSP Indirect	5
PIC32 USB (DM320003_2)	32MX795F512L	ENCX24J600	PSP 9	
PIC32 USB (DM320003_2)	32MX795F512L	MRF24WB0M	SPI2	
PIC32 Ethernet Starter Kit	32MX795F512L	Internal MAC, National DP83848C PHY	-	
dsPIC33E USB Starter Kit	33EP512MU810	ENCX24J600	SPI2	
dsPIC33E USB Starter Kit	33EP512MU810	ENCX24J600	PSP 5	
dsPIC33E USB Starter Kit	33EP512MU810	ENCX24J600	PSP Indirect	5
dsPIC33E USB Starter Kit	33EP512MU810	MRF24WB0M	SPI2	
PIC24E USB Starter Kit	24EP512GU810	ENCX24J600	SPI2	
PIC24E USB Starter Kit	24EP512GU810	ENCX24J600	PSP5	
PIC24E USB Starter Kit	24EP512GU810	ENCX24J600	PSP Indirect	5
PIC24E USB Starter Kit	24EP512GU810	MRF24WB0M	SPI2	

**TCPIP WebVend ( see page 120)**

Demo Board	Processor	MAC/PHY Layer	Comm. Bus	Notes
PICDEM.net 2	18F97J60	ENC28J60	SPI	
PICDEM.net 2	18F97J60	ETH97J60	-	
Explorer 16	24FJ128GA010	ENC28J60	SPI	
Explorer 16	24FJ128GA010	ENCX24J600	SPI	
Explorer 16	24FJ128GA010	MRF24WB0M	SPI	
Explorer 16	33FJ256GP710	ENC28J60	SPI	
Explorer 16	33FJ256GP710	ENCX24J600	SPI	
Explorer 16	33FJ256GP710	MRF24WB0M	SPI	
Explorer 16	32MX360F512L	ENC28J60	SPI	
Explorer 16	32MX360F512L	ENCX24J600	SPI	
Explorer 16	32MX360F512L	MRF24WB0M	SPI	
Explorer 16	32MX460F512L	ENC28J60	SPI	
Explorer 16	32MX460F512L	ENCX24J600	SPI	
Explorer 16	32MX460F512L	MRF24WB0M	SPI	
Explorer 16	32MX795F512L	ENC28J60	SPI	
Explorer 16	32MX795F512L	ENCX24J600	SPI	
Explorer 16	32MX795F512L	MRF24WB0M	SPI	

**TCPIP WiFi EasyConfig Demo App ( see page 127)**

Demo Board	Processor	MAC/PHY Layer	Comm. Bus	Notes
Explorer 16	24FJ128GA010	MRF24WB0M	SPI	

Explorer 16		33FJ256GP710	MRF24WB0M	SPI	
Explorer 16		32MX360F512L	MRF24WB0M	SPI	
Explorer 16		32MX460F512L	MRF24WB0M	SPI	
Explorer 16		32MX795F512L	MRF24WB0M	SPI	
PIC24FJ256DA210 Board	Development	PIC24FJ256DA210	MRF24WB0M	SPI	

**TCPIP WiFi Console Demo App** (see page 121)

Demo Board		Processor	MAC/PHY Layer	Comm. Bus	Notes
PICDEM.net 2		18F97J60	MRF24WB0M	SPI	
Explorer 16		24FJ128GA010	MRF24WB0M	SPI	
Explorer 16		33FJ256GP710	MRF24WB0M	SPI	
Explorer 16		32MX360F512L	MRF24WB0M	SPI	
Explorer 16		32MX460F512L	MRF24WB0M	SPI	
Explorer 16		32MX795F512L	MRF24WB0M	SPI	
PIC24FJ256DA210 Board	Development	PIC24FJ256DA210	MRF24WB0M	SPI	

**TCPIP Internet Radio App** (see page 120)

Demo Board		Processor	MAC/PHY Layer	Comm. Bus	Notes
Internet Radio Board		18F67J60	ENC28J60	SPI	

**TCPIP Internet Bootloader** (see page 114)

Demo Board	Processor	MAC/PHY Layer	Comm. Bus	Notes
N/A	18F66J60	ETH97J60	-	
N/A	18F66J60	ETH97J60	-	Extended Instruction Mode
N/A	18F66J65	ETH97J60	-	
N/A	18F66J65	ETH97J60	-	Extended Instruction Mode
N/A	18F67J60	ETH97J60	-	
N/A	18F67J60	ETH97J60	-	Extended Instruction Mode
N/A	18F86J60	ETH97J60	-	
N/A	18F86J60	ETH97J60	-	Extended Instruction Mode
N/A	18F86J65	ETH97J60	-	
N/A	18F86J65	ETH97J60	-	Extended Instruction Mode
N/A	18F87J60	ETH97J60	-	
N/A	18F87J60	ETH97J60	-	Extended Instruction Mode
N/A	18F96J60	ETH97J60	-	
N/A	18F96J60	ETH97J60	-	Extended Instruction Mode
N/A	18F96J65	ETH97J60	-	
N/A	18F96J65	ETH97J60	-	Extended Instruction Mode
N/A	18F97J60	ETH97J60	-	

N/A	18F97J60	ETH97J60	-	Extended Instruction Mode
-----	----------	----------	---	---------------------------

### TCPIP MDD Demo App (see page 129)

Demo Board	Processor	MAC/PHY Layer	Comm. Bus	Notes
Explorer 16	PIC24FJ256GB110	ENC28J60	SPI	Uses USB Thumb Drive as a storage medium for web pages.
Explorer 16	PIC24FJ128GA010	ENC28J60	SPI	Uses SD Card as a storage medium for web pages.

### Google Map

For information on the Google Map demo compatibility, see the file "Getting Started - Running the Graphics Google Map Demo" in the `Combo Demos/Google Map` directory in your Microchip Applications Library installation directory.

## 7.2 Available Demos

The TCP/IP Stack comes with several example applications. These applications are described in the following sections.

### 7.2.1 Demo App

The **TCPIP Demo App** project folder contains the main demo application for the Microchip TCP/IP Stack. Besides showing example applications using the web server, e-mail client, SNMP server, and more, this application also includes examples for implementing custom application layers. Details about these applications are provided here.

For a list of pre-tested demo hardware configurations, please consult the Demo Compatibility Table (see page 77). Unspecified hardware configurations may also be useable with the Demo App, but some additional configuration may be necessary.

Some demo features are disabled in certain Demo App projects to support the associated hardware platform and TCP/IP controller. Please consult the following table to determine which features are available on which configurations:

#### 7.2.1.1 TCPIP Demo App Features by Hardware Platform

Some hardware platforms cannot support all of the features implemented in the TCP/IP Demo Application. The following table outlines which features are available for each combination of demo board and MAC/PHY layer supported natively by the TCP/IP Demo App. Note that this table will not appear in the PDF version of the help file; see the "TCPIP Demo App Features.htm" file in the TCPIP documentation folder in the Microchip Application Library help folder.

#### NVM Storage

A board with Non-Volatile Memory can modify and save its configuration variables at runtime. In the TCP/IP Demo App, this allows you to change the board name, IP Address (see page 141), wireless SSID, wireless security, or other configuration parameters via a web page interface. The data will be written to SPI Flash or EEPROM and then used to reinitialize the board if it is reset. A board without this feature will always use the default settings after power-up.

#### Buttons and LEDs

The TCP/IP Stack-compatible demo boards have a variable number of buttons and LEDs. By default, the TCP/IP Demo App is configured to display and accept (see page 163) input from 8 LEDs and 4 buttons on the demo's index page; the buttons and LEDs used depend on what is available on the board.

## 7.2.1.2 Demo Modules

### Modules

Name	Description
Web Page Demos ( <a href="#">see page 82</a> )	Provides an example for building a custom HTTP application using the HTTP2 server and allows several other demo features to be accessed and controlled via web interface.
E-mail (SMTP) Demo ( <a href="#">see page 91</a> )	Demonstrates how to use an e-mail client to send messages when events occur. This is a standalone demo; for the web page "Send Email" demo, see the Forms using POST ( <a href="#">see page 85</a> ) topic.
Generic TCP Client ( <a href="#">see page 92</a> )	Demonstrates how to build a TCP Client application through an HTTP client example.
Generic TCP Server ( <a href="#">see page 95</a> )	Demonstrates how to build a TCP server application
Ping (ICMP) Demo ( <a href="#">see page 96</a> )	Demonstrates how to build a Ping client.
Network Management (SNMP) Server ( <a href="#">see page 98</a> )	Describes the Simple Network Management Protocol Demo.

### Description

Several custom modules are used in this demo. This section will describe the components and functionality of these modules.

### 7.2.1.2.1 Web Page Demos

#### Functions

	Name	Description
⇒	HTTPPostSNMPCommunity ( <a href="#">see page 87</a> )	This is function HTTPPostSNMPCommunity.
⇒	HTTPPostConfig ( <a href="#">see page 88</a> )	Processes the configuration form on config/index.htm
⇒	HTTPPostDDNSConfig ( <a href="#">see page 88</a> )	Parsing and collecting http data received from http form.
⇒	HTTPPostEmail ( <a href="#">see page 89</a> )	Processes the e-mail form on email/index.htm
⇒	HTTPPostLCD ( <a href="#">see page 89</a> )	Processes the LCD form on forms.htm
⇒	HTTPPostMD5 ( <a href="#">see page 90</a> )	Processes the file upload form on upload.htm

#### Variables

	Name	Description
◆	DDNSData ( <a href="#">see page 90</a> )	RAM allocated for DDNS parameters
◆	lastFailure ( <a href="#">see page 91</a> )	Stick status message variable. See lastSuccess ( <a href="#">see page 91</a> ) for details.
◆	lastSuccess ( <a href="#">see page 91</a> )	Sticky status message variable. This is used to indicated whether or not the previous POST operation was successful. The application uses these to store status messages when a POST operation redirects. This lets the application provide status messages after a redirect, when connection instance data has already been lost.

### Description

The CustomHTTPApp.c file demonstrates how to build a custom HTTP application on top of the HTTP2 server. All the features of the TCPIP Demo App web pages are implemented here. Examples can be found for handling Authentication ([see page 84](#)), processing web forms (using HTTP GET and POST), and providing status information through the output of dynamic variables.

### 7.2.1.2.1.1 Dynamic Variables

#### Module

Web Page Demos (🔗 see page 82)

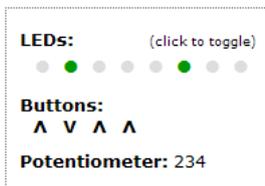
#### Description

##### Overview

This section describes how to view dynamic variables in the TCP/IP Demo App HTTP2 demo. For information about how to implement dynamic variables in your own page, see the HTTP2 dynamic variables topic (🔗 see page 225).

##### Instructions

1. Program your board with the demo code and upload the demo app web page. Open your web browser and navigate to the board's web page (<http://mchpboard> by default).
2. Observe the LED output, button state, and potentiometer reading in the box in the upper right of the web page. The button and potentiometer values will be updated dynamically based on the status of the buttons on your board. In addition, if you click the LEDs to toggle them, their status will be dynamically updated on the page. Note that some LEDs or buttons may not be implemented, depending on your hardware setup. Consult the TCPIP Demo App Features by Hardware Platform (🔗 see page 81) topic for more information.



3. Observe the current Stack Version and Build Date in the top center of the Overview Page.
4. Navigate to the Dynamic Variables page using the navigation panel on the left of the page.
5. Observe the Build Date and Time, LED state, stack version, and current IP address- these variables are output to this page dynamically when it's downloaded by the browser.

##### Exercises

You can optionally complete the exercises described on the Dynamic Variables page. You may want to read the HTTP2 dynamic variables topic (🔗 see page 225) first. The first exercise is to implement the display of LED0 on the dynamic variable demo page.

1. Start by opening `dynvars.htm` in your "TCPIP Demo App\WebPages2" folder.
2. Locate the dynamic variables in the page and replace the question mark with a dynamic variable to display the value of LED 0. You can use the other LED variables as a template, but specify 0 as the LED to open.

```
<div class="examplebox code" style="letter-spacing: 10px">
~led(7)~ ~led(6)~ ~led(5)~ ~led(4)~ ~led(3)~ ~led(2)~ ~led(1)~ ?
</div>
```

3. In your MPLAB project, open `CustomHTTPApp.c` and ensure that the `HTTPPrint_led` function (if you used `~led(0)~` as your dynamic variable) if written to output data when 0 is passed in as a parameter.
4. Rebuild your web page with the MPFS2 Utility.
5. Rebuilt your project, and reprogram your board. Navigate to the dynamic variable page and verify that the LED0 field reflects the status of the LED on your board. Since the LED on your board is blinking, you may need to refresh the web page to view its current status.

The second exercise on this page simply demonstrates how to dynamically insert a file into a web page.

1. Start by opening `dynvars.htm` in your "TCPIP Demo App\WebPages2" folder.
2. Locate the dynamic variables that include `header.inc` and `footer.inc`. Observe the difference between the declaration of these variables and the other variables on the page.

### 7.2.1.2.1.2 Authentication

#### Module

Web Page Demos (🔗 see page 82)

#### Description

##### Overview

This section describes how to use the authentication demo in the TCP/IP Demo App HTTP2 demo. For information about how to implement authentication in your own page, see the HTTP2 Authentication topic (🔗 see page 230).

##### Instructions

1. Program your board with the demo code and upload the demo app web page. Open your web browser and navigate to the board's web page (<http://mchpboard> by default).
2. Navigate to the Authentication page using the navigation panel on the left of the page.
3. Note the authentication user name ("admin") and password ("microchip").
4. Click on the "Access Restricted Page" link on the Authentication page.



User Name: admin Password: microchip  
Access Restricted Page

5. Enter an incorrect combination of usernames and passwords. The browser will not advance to the Access Restricted Page. After 3 incorrect username/password combinations, the browser will be redirected to an "Unauthorized" screen.
6. Click the back button in your browser. Click on the "Access Restricted Page" link and enter the correct username and password.
7. You will advance to the "Login Successful" page. Your browser will store this username/password combination until it is closed and reopened.

##### Exercise

You can optionally complete the exercise described on the "Login Successful" page. In this exercise, you will change the username and password that you use to log in to this page.

1. Open `CustomHTTPApp.c` in your TCP/IP Demo App MPLAB project.
2. Locate the `HTTPCheckAuth` (🔗 see page 235) function.
3. Change the values being compared to the function inputs to a username and password of your choosing.
4. Rebuild your project and program your board.

### 7.2.1.2.1.3 Forms using GET

#### Module

Web Page Demos (🔗 see page 82)

#### Description

##### Overview

This section describes how to use web forms in the TCP/IP Demo App HTTP2 demo. For information about how to implement forms in your own page, see the HTTP2 form processing topic (🔗 see page 227).

##### Instructions

1. Program your board with the demo code and upload the demo app web page. Open your web browser and navigate to the board's web page (<http://mchpboard> by default).
2. Observe the LED state on the board. Click on an LED indicator in the box on the top right of the Overview page. Verify that the LED state changes on the board. Note that some LEDs or buttons may not be implemented, depending on your hardware setup. Consult the TCP/IP Demo App Features by Hardware Platform (🔗 see page 81) topic for more information.
3. Navigate to the Form Processing page using the navigation panel on the left of the page.

- Select new LED states in the pull-down boxes. Click "Save" and observe that the LED states of your board changed to match the settings you selected.

### Exercise

You can optionally complete the exercise described on the "Form Processing" page. In this exercise, you will change the example to support LED5. You may want to read the HTTP2 form processing topic (see page 227) first.

- Start by opening `forms.htm` in your "TCPIP Demo App\WebPages2" folder.
- Locate the GET method implementation that will display the LEDs. You should see `select` forms for the four LEDs that are already implemented. Each of these has two options: the On option will send a '1' to the server when submitted and the Off option will send a '0' when submitted. Each of the declarations of these options also use the `ledSelected` dynamic variable to determine which option will be selected by default, based on the current status of the corresponding LED on the board. This dynamic variable accepts two arguments: the first defines which LED is being checked, and the second describes the state being checked for. So, for example, the `~ledSelected(4, TRUE)~` variable will be replaced by the word "SELECTED" if LED4 is on when this variable callback function is called. In this case, `~ledSelected(4, FALSE)~` would be replaced by nothing. This would result in the 'On' option being selected by default in the page.
- Create a new `select` input for LED5.
- Open `CustomHTTPApp.c` in the TCP/IP Demo App MPLAB project.
- Verify that the `HTTPPrint_ledSelected` dynamic variable callback function has been implemented for LED5.
- Find the `HTTPExecuteGet` (see page 236) function. Locate the section of code that processes GET arguments for the `forms.htm` file.
- Add implementation to search for the "led5" argument string in the GET data buffer and then set `LED5_IO` based on the associated value.

## 7.2.1.2.1.4 Forms using POST

### Module

Web Page Demos (see page 82)

### Description

#### Overview

This section describes how to use web forms in the TCP/IP Demo App HTTP2 demo. For information about how to implement forms in your own page, see the HTTP2 form processing topic (see page 227).

#### Instructions

- Program your board with the demo code and upload the demo app web page. Open your web browser and navigate to the board's web page (`http://mchpboard` by default).
- Navigate to the Form Processing page using the navigation panel on the left of the page.
- Enter a text string into the "LCD" text box and click on "Save." Verify that this string was written to the LCD display on your demo board.

- Navigate to the File Uploads page using the navigation panel on the left of the page.
- Browser for a file on your computer and click "Get MD5." The application will read your file using a series of POST transfers and calculate and display and MD5 hash of the contents.

- Navigate to the Send E-mail page using the navigation panel on the left of the page.

7. Fill in the form fields with the appropriate information.

1. No SSL - You will need a local SMTP server that does not require a secure connection. Enter the address in the SMTP Server field, set the port to 25, and enter your user name and password for the server. Set the "To:" field to the email recipient and press "Send Message."
2. SSL - Enter the address of a public SMTP server (e.g. smtp.gmail.com). Set the port number to 465 or 587. Enter your email account information (e.g. username@gmail.com and your Gmail password). Set the "To:" field to the email recipient and press "Send Message." Note that some corporate subnets may block outgoing secure traffic on the SMTP port. If this is the case, you'll have to establish a VPN tunnel outside this network or connect (see page 165) your board to a network that's not blocked by this type of firewall. You must have installed the Microchip Data Encryption Libraries to use SSL, and SSL Client support must be enabled. See the SSL API (see page 373) topic for more information.

**SMTP Server:**  **Port:**

Use SSL (usually port 465)

**User Name:**

**Password:**

**To:**

**Message:**

8. Verify that the e-mail was received on the recipient e-mail address.

### 7.2.1.2.1.5 Cookies

#### Module

Web Page Demos (see page 82)

#### Description

##### Overview

This section describes how to use the cookie demo in the TCP/IP Demo App HTTP2 demo. For information about how to implement cookies in your own page, see the HTTP2 Cookies topic (see page 232).

##### Instructions

1. Program your board with the demo code and upload the demo app web page. Open your web browser and navigate to the board's web page (http://mchpboard by default).
2. Navigate to the Cookies page using the navigation panel on the left of the page.
3. Type your first name into the "First Name" text box and click "Set Cookies." Verify that the name was read successfully and displayed in the "Name" output field.

**First Name:**

**Favorite:**

---

**Name:** not set

**Favorite:** not implemented

#### Exercise

You can optionally complete the exercise described on the "Cookies" page. In this exercise, you will create a cookie called "fav" with the value in the favorite field in the example box. You may want to read the HTTP2 dynamic variable (see page 225), GET (see page 227), and cookie (see page 232) topics first.

1. Start by opening `cookies.htm` in your "TCPIP Demo App\WebPages2" folder.
2. Locate the code for the example box that displays the name and favorite PIC architecture. Replace (see page 216) the "not implemented" string with a dynamic variable to output the data from the cookie.
3. Locate the code for the select box form input for the favorite architecture. Note the value of the name field of the select form.
4. Open `CustomHTTPApp.c` in the TCP/IP Demo App MPLAB project. Locate the `HTTPExecuteGet` (see page 236) function and find the code that handles GET method inputs from `cookies.htm`.
5. Set the value of `curHTTP.hasArgs` to indicate that two form arguments are present in the data buffer.
6. In `CustomHTTPApp.c`, create a function to output data for the dynamic variable you created in step 2. The name of the function will depend on the name of the variable. For a variable named `~cookiefavorite~` you would implement a function called `HTTPPrint_cookiefavorite`. This function should search through the `curHTTP.data` data buffer to try and find a name/value pair with the name equal to the name of your select form from step 3. If it finds it, it should write the value for that pair to the TCP buffer; otherwise, it should write "not set." See the implementation of `HTTPPrint_cookiename` for an example.

```
void HTTPPrint_cookiename(void)
{
    BYTE *ptr;

    ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE*)"name");

    if(ptr)
        TCPPutString(sktHTTP, ptr);
    else
        TCPPutROMString(sktHTTP, (ROM BYTE*)"not set");

    return;
}
```

7. Compile your web page using the MPFS2 Utility and upload it to your board. You may receive a warning that your dynamic variables have changed in your page.
8. Rebuild your project and program your board.
9. Verify that both cookies can be set.

### 7.2.1.2.1.6 Functions

#### Functions

	Name	Description
☞	<code>HTTPPostSNMPCommunity</code> (see page 87)	This is function <code>HTTPPostSNMPCommunity</code> .
☞	<code>HTTPPostConfig</code> (see page 88)	Processes the configuration form on <code>config/index.htm</code>
☞	<code>HTTPPostDDNSConfig</code> (see page 88)	Parsing and collecting http data received from http form.
☞	<code>HTTPPostEmail</code> (see page 89)	Processes the e-mail form on <code>email/index.htm</code>
☞	<code>HTTPPostLCD</code> (see page 89)	Processes the LCD form on <code>forms.htm</code>
☞	<code>HTTPPostMD5</code> (see page 90)	Processes the file upload form on <code>upload.htm</code>

#### Module

Web Page Demos (see page 82)

#### 7.2.1.2.1.6.1 HTTPPostSNMPCommunity Function

#### File

`CustomHTTPApp.c`

**C**

```
static HTTP_IO_RESULT HTTPPostSNMPCommunity();
```

**Description**

This is function HTTPPostSNMPCommunity.

**7.2.1.2.1.6.2 HTTPPostConfig Function****File**

CustomHTTPApp.c

**C**

```
static HTTP_IO_RESULT HTTPPostConfig();
```

**Description**

Accepts configuration parameters from the form, saves them to a temporary location in RAM, then eventually saves the data to EEPROM or external Flash.

When complete, this function redirects to config/reboot.htm, which will display information on reconnecting to the board.

This function creates a shadow copy of the AppConfig structure in RAM and then overwrites incoming data there as it arrives. For each name/value pair, the name is first read to curHTTP.data[0:5]. Next, the value is read to newAppConfig. Once all data has been read, the new AppConfig is saved back to EEPROM and the browser is redirected to reboot.htm. That file includes an AJAX call to reboot.cgi, which performs the actual reboot of the machine.

If an IP address cannot be parsed, too much data is POSTed, or any other parsing error occurs, the browser reloads config.htm and displays an error message at the top.

**Preconditions**

None

**Return Values**

Return Values	Description
HTTP_IO_DONE	all parameters have been processed
HTTP_IO_NEED_DATA	data needed by this function has not yet arrived

**7.2.1.2.1.6.3 HTTPPostDDNSConfig Function****File**

CustomHTTPApp.c

**C**

```
static HTTP_IO_RESULT HTTPPostDDNSConfig();
```

**Description**

This routine will be executed every time the Dynamic DNS Client configuration form is submitted. The http data is received as a string of the variables separated by '&' characters in the TCP RX buffer. This data is parsed to read the required configuration values, and those values are populated to the global array (DDNSData (see page 90)) reserved for this purpose. As the data is read, DDNSPointers is also populated so that the dynamic DNS client can execute with the new parameters.

**Preconditions**

curHTTP (see page 234) is loaded.

**Return Values**

Return Values	Description
HTTP_IO_DONE	Finished with procedure
HTTP_IO_NEED_DATA	More data needed to continue, call again later

HTTP_IO_WAITING	Waiting for asynchronous process to complete, call again later
-----------------	--

#### 7.2.1.2.1.6.4 HTTPPostEmail Function

##### File

CustomHTTPApp.c

##### C

```
static HTTP_IO_RESULT HTTPPostEmail();
```

##### Description

This function sends an e-mail message using the SMTP client and optionally encrypts the connection to the SMTP server using SSL. It demonstrates the use of the SMTP client, waiting for asynchronous processes in an HTTP callback, and how to send e-mail attachments using the stack.

Messages with attachments are sent using multipart/mixed MIME encoding, which has three sections. The first has no headers, and is only to be displayed by old clients that cannot interpret the MIME format. (The overwhelming majority of these clients have been obsoleted, but the so-called "ignored" section is still used.) The second has a few headers to indicate that it is the main body of the message in plain-text encoding. The third section has headers indicating an attached file, along with its name and type. All sections are separated by a boundary string, which cannot appear anywhere else in the message.

##### Preconditions

None

##### Return Values

Return Values	Description
HTTP_IO_DONE	the message has been sent
HTTP_IO_WAITING	the function is waiting for the SMTP process to complete
HTTP_IO_NEED_DATA	data needed by this function has not yet arrived

#### 7.2.1.2.1.6.5 HTTPPostLCD Function

##### File

CustomHTTPApp.c

##### C

```
static HTTP_IO_RESULT HTTPPostLCD();
```

##### Description

Locates the 'lcd' parameter and uses it to update the text displayed on the board's LCD display.

This function has four states. The first reads a name from the data string returned as part of the POST request. If a name cannot be found, it returns, asking for more data. Otherwise, if the name is expected, it reads the associated value and writes it to the LCD. If the name is not expected, the value is discarded and the next name parameter is read.

In the case where the expected string is never found, this function will eventually return HTTP\_IO\_NEED\_DATA when no data is left. In that case, the HTTP2 server will automatically trap the error and issue an Internal Server Error to the browser.

##### Preconditions

None

##### Return Values

Return Values	Description
HTTP_IO_DONE	the parameter has been found and saved
HTTP_IO_WAITING	the function is pausing to continue later
HTTP_IO_NEED_DATA	data needed by this function has not yet arrived

**Section**

Function Prototypes and Memory Globalizers

**7.2.1.2.1.6 HTTPPostMD5 Function****File**

CustomHTTPApp.c

**C**

```
static HTTP_IO_RESULT HTTPPostMD5();
```

**Description**

This function demonstrates the processing of file uploads. First, the function locates the file data, skipping over any headers that arrive. Second, it reads the file 64 bytes at a time and hashes that data. Once all data has been received, the function calculates the MD5 sum and stores it in curHTTP.data.

After the headers, the first line from the form will be the MIME separator. Following that is more headers about the file, which we discard. After another CRLFCRLF, the file data begins, and we read it 16 bytes at a time and add that to the MD5 calculation. The reading terminates when the separator string is encountered again on its own line. Notice that the actual file data is trashed in this process, allowing us to accept (see page 163) files of arbitrary size, not limited by RAM. Also notice that the data buffer is used as an arbitrary storage array for the result. The ~uploadedmd5~ callback reads this data later to send back to the client.

**Preconditions**

None

**Return Values**

Return Values	Description
HTTP_IO_DONE	all parameters have been processed
HTTP_IO_WAITING	the function is pausing to continue later
HTTP_IO_NEED_DATA	data needed by this function has not yet arrived

**7.2.1.2.1.7 Variables****Module**

Web Page Demos (see page 82)

**Variables**

	Name	Description
	DDNSData (see page 90)	RAM allocated for DDNS parameters
	lastFailure (see page 91)	Stick status message variable. See lastSuccess (see page 91) for details.
	lastSuccess (see page 91)	Sticky status message variable. This is used to indicated whether or not the previous POST operation was successful. The application uses these to store status messages when a POST operation redirects. This lets the application provide status messages after a redirect, when connection instance data has already been lost.

**7.2.1.2.1.7.1 DDNSData Variable****File**

CustomHTTPApp.c

**C**

```
BYTE DDNSData[100];
```

**Description**

RAM allocated for DDNS parameters

**7.2.1.2.1.7.2 lastFailure Variable****File**

CustomHTTPApp.c

**C**

```
BOOL lastFailure = FALSE;
```

**Description**

Stick status message variable. See lastSuccess (see page 91) for details.

**7.2.1.2.1.7.3 lastSuccess Variable****File**

CustomHTTPApp.c

**C**

```
BOOL lastSuccess = FALSE;
```

**Description**

Sticky status message variable. This is used to indicate whether or not the previous POST operation was successful. The application uses these to store status messages when a POST operation redirects. This lets the application provide status messages after a redirect, when connection instance data has already been lost.

**7.2.1.2.2 E-mail (SMTP) Demo****Functions**

	Name	Description
	SMTPDemo (see page 92)	Demonstrates use of the e-mail (SMTP) client.

**Description****Overview**

This file provides two examples for using the SMTP client module to send e-mail messages. The first transmits short alert messages whose entire bodies can be stored in RAM at once. The second example demonstrates how to generate messages on-the-fly when the entire body cannot be allocated in RAM. (This second example is commented. You must comment the first example and uncomment this one to use it.)

A third example of using the SMTP client is provided in HTTPPostEmail (see page 89). This example shows how to send messages with attachments, as well as how to dynamically configure the recipient and e-mail server at run-time.

**Instructions (Short Message Demo)**

1. Open your project in MPLAB and open SMTPDemo.c. Scroll down to the MAIL\_BEGIN case in the switch statement in the SMTPDemo (see page 92)() function.
  1. Replace (see page 216) the initializer of the RAMStringTo[] array with the target email address.
  2. Replace (see page 216) the initializer of the SMTPClient.Server.szROM structure element with the address of your mail server. Note that this demo does not include security features, so you will need a mail server that does not require SSL. To test this functionality with a mail server that does support SSL (including most public mail servers), please use the HTTPPostEmail (see page 89) SMTP demo.
2. Compile the code, program your board, and run the demo.
3. Press buttons 2 and 3 on your board to transmit an email message. LED1 on your board will indicate that the message is being transmitted; LED2 will indicate that it was sent successfully. Check the BUTTON2\_IO, BUTTON3\_IO, LED1\_IO,

and LED2\_IO macros in the copy of HardwareProfile.h that corresponds to your project to determine which buttons and LEDs are used for your hardware setup.

- Verify that the message was received by the email account you specified in the RAMStringTo[] array.

#### Description

The short-message SMTPDemo (see page 92) task function implements a four-state state machine. When the board is powered on, the state machine is initialized to the SM\_HOME state, in which it waits for buttons 2 and 3 to be pressed. Once they are pressed, the task will enter the MAIL\_BEGIN state.

In the MAIL\_BEGIN state, the task will attempt to requisition the SMTP module. Once it's able to do this, it will populate the SMTPClient (see page 298) structure with message parameters and transmit the message. It will then enter the MAIL\_SMTP\_FINISHING state.

In the MAIL\_SMTP\_FINISHING state, the task will check a callback function (SMTPIsBusy (see page 299)) to determine when the module is finished. It will then give up control of the SMTP module and toggle LEDs based on the successful operation of the SMTP module. The state machine will then enter the MAIL\_DONE state, which will wait at least 1 second before transitioning back to MAIL\_HOME, allowing another email to be sent.

### 7.2.1.2.2.1 SMTPDemo Function

#### File

MainDemo.h

#### C

```
void SMTPDemo( );
```

#### Module

E-mail (SMTP) Demo (see page 91)

#### Returns

None

#### Description

This function demonstrates the use of the SMTP client. The function is called periodically by the stack, and checks if BUTTON2 and BUTTON3 are pressed simultaneously. If they are, it attempts to send an e-mail message using parameters hard coded in the function below.

While the client is executing, LED1 will be used as a busy indicator. LED2 will light when the transmission has been completed successfully. If both LEDs extinguish, an error occurred.

For an example of sending a longer message (one that does not exist in RAM all at once), see the commented secondary implementation of this function in this file (SMTPDemo.c) below. For an example of sending a message using parameters gathered at run time, and/or a message with attachments, see the implementation of HTTPPostEmail (see page 89) in CustomHTTPApp.c.

#### Preconditions

The SMTP client is initialized.

### 7.2.1.2.3 Generic TCP Client

#### Functions

	Name	Description
	GenericTCPClient (see page 93)	Implements a simple HTTP client (over TCP).

**Variables**

	Name	Description
	RemoteURL ( <a href="#">see page 94</a> )	Defines the URL to be requested by this HTTP client
	ServerName ( <a href="#">see page 94</a> )	Defines the server to be accessed for this application
	ServerPort ( <a href="#">see page 94</a> )	Defines the port to be accessed for this application

**Description****Overview**

The Generic TCP Client provides an example of how to build an HTTP client (or any other TCP client) using the Microchip TCP/IP Stack. It will print out the results from a search engine query to the PIC's UART module. The result data can be viewed on a PC terminal.

**Instructions**

1. Connect the programmed demo board to a router that is connected to the Internet.
2. Connect your PC to your demo board with an RS-232 cable. Open a terminal program like HyperTerminal, and configure it to the following settings: 19200 bps, 8 data bits, No parity, 1 stop bit, No flow control.
3. Press Button 1 on your demo board (check the BUTTON1\_IO macro in the copy of HardwareProfile.h that corresponds to your project to determine which button is Button 1).
4. Observe the search results for "Microchip" at [www.microchip.com](http://www.microchip.com) on your terminal.

**Description**

The Generic TCP Client demo implements a task function with five states. When the board is powered on, the initial state will be set to SM\_DONE. This state will wait for the user to press Button 1; when a button-press event occurs, the state will switch to SM\_HOME. In the SM\_HOME state, the task will attempt to open a TCP client socket ([see page 146](#)). This socket will use a TCP\_PURPOSE\_GENERIC\_TCP\_CLIENT socket type ([see page 146](#)) from the TCP socket structure ([see page 147](#)) that was initialized in your configuration files. The targeted server will be the Google search engine, and the server port will be 80, the port used for HTTP connections. The task will switch the state machine to the SM\_SOCKET\_OBTAINED state.

The task will wait in the SM\_SOCKET\_OBTAINED state until a connection is established with Google or a 5-second timeout elapses. If a timeout occurs, the state will close the socket and change the state back to SM\_HOME. Otherwise, it will wait until the TCP buffer can accept ([see page 163](#)) 125 bytes of data and then use an HTTP GET ([see page 227](#)) to search for the word "Microchip" at the site "microchip.com." Once the GET has been sent, the state will switch to SM\_PROCESS\_RESPONSE.

In the SM\_PROCESS\_RESPONSE state, the task will wait until a response is received or the socket was disconnected. If a response is received, it will print it to the UART. In either case, the task will transition to the SM\_DISCONNECT state, where it will close the client socket and return to the SM\_DONE state.

**7.2.1.2.3.1 GenericTCPClient Function****File**

MainDemo.h

**C**

```
void GenericTCPClient();
```

**Module**

Generic TCP Client ([see page 92](#))

**Returns**

None

**Description**

This function implements a simple HTTP client, which operates over TCP. The function is called periodically by the stack,

and waits for BUTTON1 to be pressed. When the button is pressed, the application opens a TCP connection to an Internet search engine, performs a search for the word "Microchip" on "microchip.com", and prints the resulting HTML page to the UART.

This example can be used as a model for many TCP and HTTP client applications.

### Preconditions

TCP is initialized.

## 7.2.1.2.3.2 Variables

### Module

Generic TCP Client ([see page 92](#))

### Variables

	Name	Description
	RemoteURL ( <a href="#">see page 94</a> )	Defines the URL to be requested by this HTTP client
	ServerName ( <a href="#">see page 94</a> )	Defines the server to be accessed for this application
	ServerPort ( <a href="#">see page 94</a> )	Defines the port to be accessed for this application

#### 7.2.1.2.3.2.1 RemoteURL Variable

### File

GenericTCPClient.c

### C

```
ROM BYTE RemoteURL[] = "/search?as_q=Microchip&as_sitesearch=microchip.com";
```

### Description

Defines the URL to be requested by this HTTP client

#### 7.2.1.2.3.2.2 ServerName Variable

### File

GenericTCPClient.c

### C

```
BYTE ServerName[] = "www.google.com";
```

### Description

Defines the server to be accessed for this application

#### 7.2.1.2.3.2.3 ServerPort Variable

### File

GenericTCPClient.c

### C

```
WORD ServerPort = 80;
```

### Description

Defines the port to be accessed for this application

## 7.2.1.2.4 Generic TCP Server

### Functions

	Name	Description
	GenericTCPServer ( <a href="#">see page 96</a> )	Implements a simple ToUpper TCP Server.

### Macros

	Name	Description
	SERVER_PORT ( <a href="#">see page 96</a> )	Defines which port the server will listen ( <a href="#">see page 169</a> ) on

### Description

#### Overview

The Generic TCP Server example demonstrates how to build a TCP server application. Once you connect ([see page 165](#)) to the demo server, it will echo your keystrokes back to you after converting the characters to UPPER CASE.

#### Instructions

1. Connect the programmed demo board to a computer either directly or through a router. For Ethernet, a direct connection may require a crossover cable; for WiFi, the board may need to be in AdHoc mode to establish a direct connection.
2. Determine the IP address of the demo board. This can be done several different ways.
  1. If you are using a demo setup with an LCD display (e.g. Explorer 16 or PICDEM.net 2), the IP address should be displayed on the second line of the display.
  2. Open the Microchip TCP/IP Discoverer from the start menu. Press the "Discover Devices" button to see the addresses and host names of all devices with the Announce ([see page 149](#)) Protocol enabled on your network. You may have to configure your computer's firewall to prevent it from blocking UDP port 30303 for this solution.
  3. If your board is connected directly with your computer with a crossover cable:
    1. Open a command/DOS prompt and type 'ipconfig'. Find the network adaptor that is connected to the board. The IP address of the board is located in the 'Default Gateway' field
    2. Open up the network status for the network adaptor that connects the two devices. This can be done by right clicking on the network connection icon in the network settings folder and select 'status' from the menu. Find the 'Default Gateway' field.
3. Open a command/DOS prompt. Type "telnet ip\_address 9760" where ip\_address is the IP address that you got from step 2 and 9760 is the TCP port chosen for the Generic TCP Server implementation.
4. As you type characters, they will be echoed back in your command prompt window in UPPER CASE.
5. Press Escape to end the demo.

#### Description

The GenericTCPServer ([see page 96](#)) demo implements a task function with 3 states. In the first state, SM\_HOME, the task will attempt to open a TCP server socket ([see page 146](#)). This socket will use a TCP\_PURPOSE\_GENERIC\_TCP\_SERVER socket type ([see page 146](#)) from the TCP socket structure ([see page 147](#)) that was initialized in your configuration files. It will also listen ([see page 169](#)) on TCP port 9760 (defined by the macro SERVER\_PORT ([see page 96](#))).

Once the socket has been successfully opened, the task function will enter the SM\_LISTENING state. In this state, the task will always return unless a client has connected to it (by establishing a telnet connection on port 9760). Once a client has connected to the server, the server will read received data from the TCP socket's RX buffer, convert it to upper case, and write it to the TCP socket's TX buffer.

If an Escape character is received, the server will enter the SM\_CLOSING state. In this state, it will close the server socket to break the current connection. The server will then re-enter the SM\_HOME state, where it will reopen the TCP\_PURPOSE\_GENERIC\_TCP\_SERVER socket to listen ([see page 169](#)) for new connections.

### 7.2.1.2.4.1 GenericTCPServer Function

#### File

MainDemo.h

#### C

```
void GenericTCPServer();
```

#### Module

Generic TCP Server ([see page 95](#))

#### Returns

None

#### Description

This function implements a simple TCP server. The function is invoked periodically by the stack to listen ([see page 169](#)) for incoming connections. When a connection is made, the server reads all incoming data, transforms it to uppercase, and echos it back.

This example can be used as a model for many TCP server applications.

#### Preconditions

TCP is initialized.

### 7.2.1.2.4.2 Macros

#### Macros

	Name	Description
	SERVER_PORT ( <a href="#">see page 96</a> )	Defines which port the server will listen ( <a href="#">see page 169</a> ) on

#### Module

Generic TCP Server ([see page 95](#))

#### 7.2.1.2.4.2.1 SERVER\_PORT Macro

#### File

GenericTCPServer.c

#### C

```
#define SERVER_PORT 9760
```

#### Description

Defines which port the server will listen ([see page 169](#)) on

### 7.2.1.2.5 Ping (ICMP) Demo

#### Functions

	Name	Description
	PingDemo ( <a href="#">see page 97</a> )	Demonstrates use of the ICMP (Ping) client.

#### Macros

	Name	Description
	HOST_TO_PING ( <a href="#">see page 98</a> )	Address ( <a href="#">see page 141</a> ) that ICMP client will ping. If the DNS client module is not available in the stack, then this hostname is ignored and the local gateway IP address will be pinged instead.

## Description

### Overview

The Ping Demo explains how to use the ICMP client to check if a remote node is reachable. If the project with this demo includes the DNS module, the PIC will ping "ww1.microchip.com." Otherwise, it will ping the local gateway. This demo is only available on hardware setups with LCD displays (e.g. Explorer 16 or PICDEM.net 2).

### Instructions

1. Press Button 0 on your demo board. Button 0 is usually the rightmost or topmost button on the board (check the `BUTTON0_IO` macro in the copy of `HardwareProfile.h` that corresponds to your project to determine exactly which button is Button 0).
2. When the device receives an echo response from the remote node or when the ping times out, the LCD will be updated with the appropriate information.

### Description

The `PingDemo` (see page 97) task function implements a two-state state machine. The task will wait in the `SM_HOME` state until the user presses button 0. Once the button is pressed, the task will attempt to obtain ownership of the ICMP module with the `ICMPBeginUsage` (see page 258) function. If it does, it will send a ping to the specified address and transition to the `SM_GET_ICMP_RESPONSE` state.

In the `SM_GET_ICMP_RESPONSE` state, the task will call the `ICMPGetReply` (see page 260) callback function and take action depending on the return value:

Value	Action
-2	Remain in this state and keep waiting for a response.
-1	Write a message to the LCD indicating that the ping timed out. Change state to <code>SM_HOME</code> .
-3	Write a message to the LCD indicating that the DNS module couldn't resolve the target address. Change state to <code>SM_HOME</code> .
Other	Convert the response time to a text string and print it to the LCD. Change state to <code>SM_HOME</code> .

### 7.2.1.2.5.1 PingDemo Function

#### File

MainDemo.h

#### C

```
void PingDemo();
```

#### Module

Ping (ICMP) Demo (see page 96)

#### Returns

None

#### Description

This function implements a simple ICMP client. The function is called periodically by the stack, and it checks if `BUTTON0` has been pressed. If the button is pressed, the function sends an ICMP Echo Request (Ping) to a Microchip web server. The round trip time is displayed on the UART when the response is received.

This function can be used as a model for applications requiring Ping capabilities to check if a host is reachable.

#### Preconditions

TCP is initialized.

### 7.2.1.2.5.2 Macros

#### Macros

	Name	Description
	HOST_TO_PING ( <a href="#">see page 98</a> )	Address ( <a href="#">see page 141</a> ) that ICMP client will ping. If the DNS client module is not available in the stack, then this hostname is ignored and the local gateway IP address will be pinged instead.

#### Module

Ping (ICMP) Demo ([see page 96](#))

#### 7.2.1.2.5.2.1 HOST\_TO\_PING Macro

#### File

PingDemo.c

#### C

```
#define HOST_TO_PING "ww1.microchip.com" // Address that ICMP client will ping. If the
DNS client module is not available in the stack, then this hostname is ignored and the
local gateway IP address will be pinged instead.
```

#### Description

Address ([see page 141](#)) that ICMP client will ping. If the DNS client module is not available in the stack, then this hostname is ignored and the local gateway IP address will be pinged instead.

### 7.2.1.2.6 Network Management (SNMP) Server

#### Functions

	Name	Description
	SendNotification ( <a href="#">see page 110</a> )	Prepare, validate remote node which will receive trap and send trap pdu.
	SNMPGetTimeStamp ( <a href="#">see page 111</a> )	Obtains the current Tick value for the SNMP time stamp.

#### Macros

	Name	Description
	MAX_TRY_TO_SEND_TRAP ( <a href="#">see page 112</a> )	

#### Variables

	Name	Description
	gSendTrapSMstate ( <a href="#">see page 111</a> )	This is variable gSendTrapSMstate.
	gSnmpv3UserSecurityName ( <a href="#">see page 111</a> )	This is variable gSnmpv3UserSecurityName.

#### Description

The Microchip SNMP server is a multilingual implementation which supports SNMPv1, V2c and V3 server features simultaneously. SNMP server is implemented to address the requirements of embedded applications. The SNMPv3 support is added with TCPIP Stack Version 5.31. SNMPv1 and V2c are enabled with single macro. The SNMPv3 server could be selectively enabled with an independent macro.

This series of topics will address the application- and demo-specific implementation of an SNMP server included with the TCP/IP Demo applications. For information describing the SNMP module in general, please see the SNMP API topic.

V2c is implemented with support for the configuration of multiple community names, which are stored in selected non-volatile

memory (SPI EEPROM or SPI Flash). The community names can be configured through the TCP/IP Configuration Wizard or through the HTTP/MPFS2 web interface. An access-restricted web page is provided with the demo application to allow dynamic configuration of SNMP communities.

SNMPv3 RFC specifies different types of access mechanism, user security model (USM), authentication and privacy protocols. Microchip SNMPv3 server is implemented with support for USM, AES 128 CFB 128 privacy protocol, MD5 and SHA1 message authentication protocols. The demo implementation of the server is configured with 3 different types of user names with respective authentication and privacy credentials and authentication types. These credentials and other user information are stored in the global array. The user of the SNMPv3 stack can decide on the number of user names in the User's data base to be stored with the Server. According to the SNMPv3 recommendation, SNMPv3 server should not be configured with the authentication and privacy passwords. Instead could be configured with the respective localized keys of the password. Microchip SNMPv3 agent is provided with the password information in the database for the "Getting Started" and for understanding purpose only. It is recommended that the SNMPv3 stack should be modified to restrict access to the password OIDs declared in the user data base.

You will require two firmware packages to run the SNMPv3 server demo. Each of these must be installed in the order listed, as some files will be overwritten. The software encryption library is not required if the SNMPv3 services are not intended.

1. The Microchip Application Libraries, including TCP/IP Stack v5.31 or later. This code is available at [www.microchip.com/mal](http://www.microchip.com/mal).
2. Microchip's Data Encryption Libraries. This demo uses the SSL security layer to communicate with Google. To use SSL with the TCP/IP stack, you will require these libraries. They are available in a CD-based or downloadable format from MicrochipDirect for a small fee (required to comply with U.S. cryptographic export restriction screening). You must execute the "Microchip TCPIP Stack vX.XX Encryption Add-on.exe" installer to install the files required by the demo. The ARCFOUR.c/h and RSA.c/h cryptographic files in this installer will replace the dummy versions found with the default TCP/IP Stack installation.

To enable SNMPv3 functionality, you must add the PIC32\_AES.a library from the Data Encryption Libraries to your demo project.

**Note:** For existing Microchip SNMP V1 and V2c users.

- SNMP V1/V2c users wanting to upgrade the Microchip TCP/IP Stack from older versions to the latest version and continue to use SNMP V1/V2c can get the SNMP V1/V2c services from this agent, provided they do not modify the default settings of the SNMP module in v5.25 onward.
- The implementation framework for V1 and V2c remains the same, except for a few new features and functions. The names and parameters of some of the functions have been changed. V1/V2c users may have to make changes to their application-specific code. There should not be any change in the SNMP stack code unless users have incorporated application code in the SNMP stack.
- Users should build a new MPFS image using the MPFS File System Generator utility and upload it to the selected EEPROM or Flash memory, as the AppConfig structure is updated to accommodate community names in V2c and SNMP engine ID for SNMPv3.

### 7.2.1.2.6.1 MIB Files

#### Module

Network Management (SNMP) Server (see page 98)

#### Description

SNMP describes the hierarchal storage of management objects (referred to with object IDs or OIDs) with Management Information Base (MIB) files. The Microchip SNMP server demo includes two MIB files:

- mchip.mib - This is an Abstract Syntax Notation One (ASN.1) formatted MIB file containing information about the variables used in the demo.

- `snmp.mib` - This is a custom-formatted file that can be parsed to create webpage and header resources that can be accessed with a PIC microcontroller.

The TCP/IP stack includes the `mib2bib` utility, which will compile the custom Microchip MIB script (`snmp.mib`) to generate two files called `snmp.bib` and `mib.h`. The `snmp.bib` file is a compressed record of management objects that will be stored with web pages and the `mib.h` file contains C defines for each OID. These files are included in the appropriate directories for the TCP/IP Demo Apps, but for a custom application you must copy `snmp.bib` to your web page directory, copy `mib.h` to your application directory and include it in your project, rebuild your project, and then rebuild and re-upload your web page. This will bundle the BIB file into your web page image, which will allow the SNMP agent to search for the required variable information with the MPFS file system.

### 7.2.1.2.6.2 MIB Browsers

#### Module

Network Management (SNMP) Server (see page 98)

#### Description

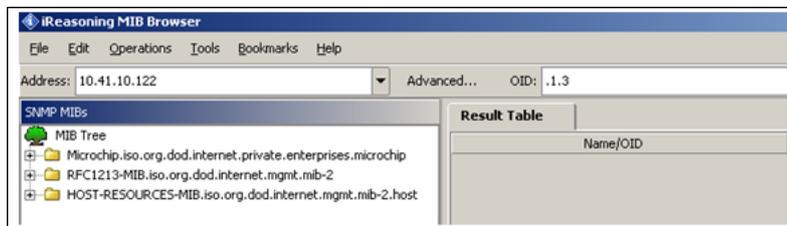
Several SNMP MIB browsers are available. Users can also install a customized MIB browser specific to their application. This help file describes using the **iReasoning** MIB browser to run the demo app. The iReasoning MIB browser can be obtained from: <http://www.ireasoning.com/downloadmibbrowserlicense.shtml>. The MIB script upload, the MIB tree structure display, and the SNMP query mechanism procedures vary from browser to browser.

Note that the use of a MIB browser or other third-party tools may require that users review and agree to the terms of a license. Microchip's reference to the **iReasoning** MIB browser is for the users' convenience. It is the user's responsibility to obtain information about, and comply with the terms of, any applicable licenses.

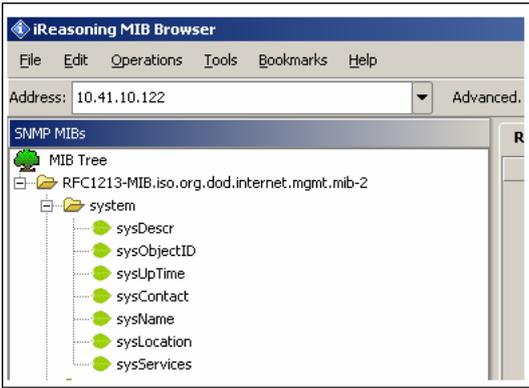
Once your browser installation has been completed, perform the following steps:

1. Copy the `mchip.mib` file to the MIB file directory of your browser (e.g. "C:\Program Files\ireasoning\mibbrowser\mibs").
2. Open the iReasoning browser, select File->Load MIBs, and select the `mchip.mib`, `RFC1213.mib` and `SNMP-FRAMEWORK-MIB.mib` (If SNMPv3 server is enabled) file.

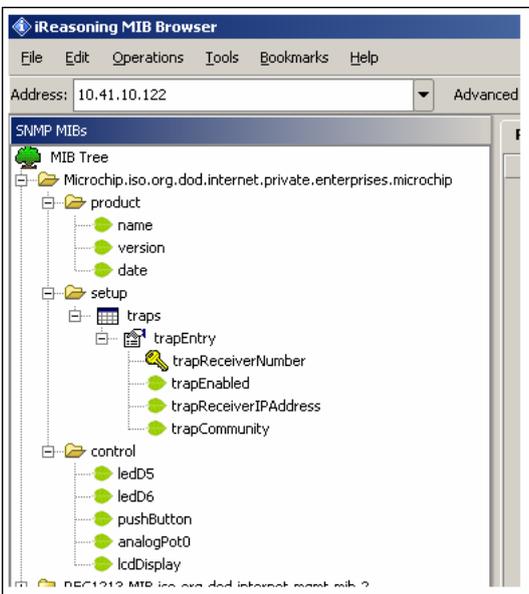
The Microchip MIB directory will be displayed in the SNMP MIB pane.



The minimum set of RFC 1213 MIB2 variables that are required to identify the Microchip node as an SNMP node to the network are implemented. These variables can be accessed by any SNMP browser with a "public" type community name. Refer to AN870 - "SNMP V2c Agent for Microchip TCP/IP Stack" for more details on the MIB scripts, community names, and demo SNMP MIB variable tree structure. The following figure shows the variables implemented in the Microchip SNMP Agent.



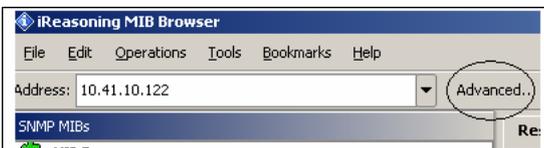
The ASN.1 format `mchip.mib` file is defined with a private variable tree structure for the MIB variables. Also the `mchip.mib` is added with number of OIDs which could be accessed only with SNMPv3 request. The browser can access every variable in the MIB database provided the community name matches. The access to the MIB variables is restricted to the type of the request. The RFC1213 mib variables could be accessed with SNMPv2c/v3 request. But the `SNMP-FRAMEWORK-MIB.mib` variables could only be accessed with SNMPv3 request if the credentials are matched and the message is authenticated. To modify these MIB variables, corresponding changes must be made to both MIB scripts (`snmp.mib` and `mchip.mib`). The following figure shows the Microchip private MIB variable tree structure in the browser.



### Configuring the Browser

To configure the iReasoning MIB browser:

1. Select the "Advanced" tab in the browser.



The following configuration window will be displayed:

2. If V2C services are required, select SNMP version V2c, configure the Read and Write community to the browser.

- The V2c agent will respond only to the queries from SNMP MIB browsers using the same community. That is, the V2c agent and the browser should be members of the same community.
- If the community fields are left blank, the manager sends the SNMP request with the community name as "public."
- The V2c agent is configured by default with 3 Read communities ("public", "read", "") and 3 Write communities ("private", "write", "public").
- The default maximum community length is 8 characters.
- As the default communities also contain the "public" community name, the agent will respond to all of the browsers requesting the "public" community.
- The TCP/IP Configuration Wizard (see page 57) can be used to configure the default SNMP community names. At run time, the community names can be dynamically configured using the HTTP interface for SNMP community name configuration.

If the V2c agent receives an SNMP request with an unknown community name, the agent will generate an Authentication (see page 84) trap.

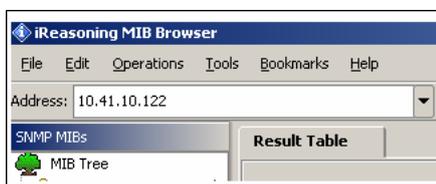
The V2c agent's multiple community support feature enables the user application to provide limited access to the requesting browser based on the community name used by the browser to access the MIB database variables of the agent.

3. If SNMPv3 services are required, select the SNMP Version as 'V3' in the 'Advanced' tab of the SNMP MIB Browser. The following configuration window will be displayed:

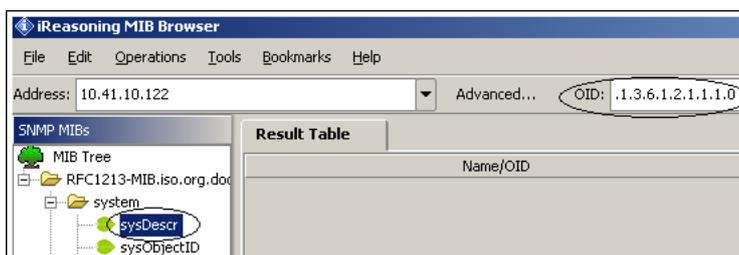
4. If SNMPv3 services are required, SNMPv3 browser is required to be configured with the user name, authentication and privacy password, message authentication hash type, privacy protocol type. The SNMP server would respond only if one of the user credentials and user security parameters in the below table is configured at the manager. The below table is stored in the global structure with the SNMPv3 server stack. The SNMPv3 server would only respond if the request credentials of the MIB browser matches to that of the stored user data base of the SNMP server.

	USER 1	USER 2	USER 3
USM User	microchip	SnmpAdmin	root
Security Level	auth, priv	auth, no priv	no auth, no priv
Auth Algorithm	MD5	SHA1	
Auth password	auth12345	ChandlerUS	
Privacy Algorithm	AES		
Privacy password	priv12345		

- The Microchip SNMPv3 stack does support only one Context Engine ID with the server. Leave the "Context Name" option in the "Advanced" tab empty. It is ignored on the server.
- According to the user and the auth and privacy protocols configured with the SNMP browser, the UDP authenticated and encrypted message would be exchanged between server and the client.
  - If the USER 1 values, as in above table, are configured in the MIB browser, the data exchange between client and server is encrypted and authenticated. The PDU could be captured in the Ethernet packet sniffer like WireShark and examined. As the data is encrypted and authenticated, the data integrity and the privacy is achieved.
  - If USER 2 values, as in above table, are configured in the MIB browser, the data exchange between client and server is authenticated. The data integrity would be checked once the data is received at either end. The message authentication mechanism protects from the possible data sniffing and modification threat, and also guarantees that the data is received from the authenticated and guaranteed source.
  - If USER 3 values, as in above table, are configured in the MIB browser, the data exchange between client and server is neither authenticated nor encrypted.
  - Considering the above three USER configurations, if the SNMP server is to be accessed over WAN, in the internet cloud, the data should be encrypted and authenticated to have the highest level of data privacy and integrity.
- Configure the IP address of the SNMP agent to the "Address" (see page 141) field.



- Select the variable to be accessed from the agent MIB database from the SNMP MIBs pane. The selected variable's OID can be seen in the OID tab in the following figure.



- Select the SNMP Get operation from the operations tab.



- The SNMPv3 server demo MIB is included with RFC1213 SNMPv2 MIB variables, private mib variables and the SNMP-FRAMEWORK-MIB variables. If the SNMPv2C request with validated community name is generated from the MIB Browser, only set of few variables is accessed. The access to the MIB variables is restricted to the type of SNMP version request received. If the SNMPv3 request with correct credentials is generated from the MIB Browser, the complete MIB access is provided.
- The user would require to decide on which part of the MIB should be required to be restricted depending upon the

SNMP version type. The MIB design is the one of the important step in deciding the MIB tree structure and the variable to be placed accordingly.

- The SNMP server demo MIB is added with a static variable OID named as "snmpv3PvtObject" with OID value as 43.6.1.4.1.17095.6.1. This variable is placed in the private branch of the MIB by creating an independent branch. All the other variables in the private branch are accessible by SNMPv2c request. The access to this static variable is restricted by the SNMP version type. Only the SNMPv3 request with correct credentials could access this variable.

### Exploring the Demo

After the MIB script is uploaded to the SNMP browser, the MIB tree structure will be displayed in the browser. Any of the variables in this tree can be accessed (using SNMP operations) from the agent if the agent supports these variables. The browser and agent should be members of the same community. To learn more about SNMP operations, PDU types, and terminology, refer to AN870 - "SNMP V2C Agent for Microchip TCP/IP Stack."

## 7.2.1.2.6.3 SNMP Operations

### Module

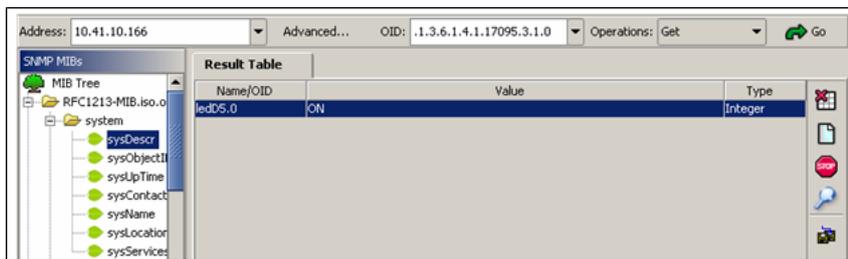
Network Management (SNMP) Server (see page 98)

### Description

#### Get

- Select the "Advanced" tab and configure the SNMP version to '1' and the Read community to "public".
- Select "Get" from the operations menu.
- Select the `sysDescr` variable from the MIB Tree.

The Result Table displays the `sysDescr` variable information. Repeat this procedure for any MIB variable. For SNMP V2c, repeat the same procedure, substituting '2' in place of '1' in the version configuration.



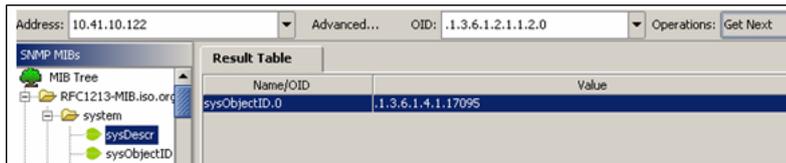
As explained earlier, the V2c agent is configured with three Read and Write community defaults. Configure the browser to use any of these communities and try accessing the MIB variables. You should be able to access some of the MIB variables even with the Read Community configured as any of the 'write' community defaults. For GET operations, if the Read or Write community matches, the agent processes the request. For SET operations, the received community names must match any of the 'write' community names.

For SNMP V3, substitute '3' in place of '1' in the version configuration in the "Advanced" tab. Configure the other user based auth and priv credentials as explained in the "MIB Browsers" section.

With appropriate credentials, all the MIB variables are accessible. Select any of the MIB variables in the MIB tree and do a GET operation.

#### Get\_Next

- Repeat the process for GET. Select the `sysDescr` variable from the MIB tree. Select "Get Next" from the operations menu. The result table will display the `sysObjectID` variable information.
- Repeat for additional MIB variables to get the information for the corresponding next variable.



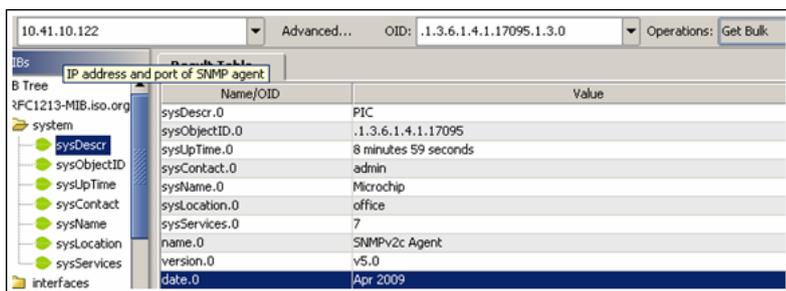
3. Set the SNMP MIB Browser version to v1/v2c. Try to access the private MIB variable "snmpv3PvtObject" with OID value as 43.6.1.4.1.17095.6.1. The access should be restricted. Set the version to V3, configure the credentials, again try a Get\_Next operation for the same variable. The access should be granted.

### Get\_Bulk

This operation is supported in SNMP V2c and SNMP V3. Get\_Bulk enables the collection of bulk information from the agent with a single request from the manager.

1. Configure the SNMP version to '2' or '3' in the SNMP browser.
2. If version is configured to '2', set the Read Community to 'public' or 'read.'
3. If version is configured to '3', configure the appropriate V3 credentials.
4. Select the `sysDescr` variable from the MIB tree.
5. Select the Get Bulk operation from the Operations menu.

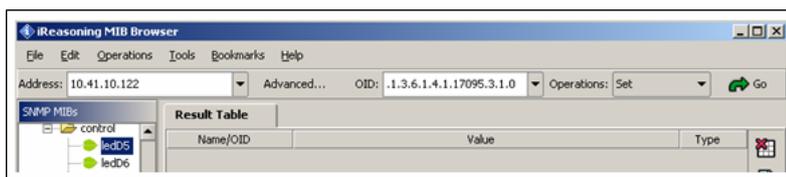
The result table will display information for 10 MIB variables in a single request (if the Max-Repetitions=10 and Non-Repeaters=0 is configured). These variables are the lexicographical successors of the `sysDescr` variable. The number of variables that the agent will respond with can be configured in the browser through the menus: "Tools->Options->Non-Repeaters" and "Tools->Options->Max-Repetitions." The Non-Repeaters and Max-Repetitions numbers are extracted by the SNMP agent from the received Get\_Bulk request and the number of variables that will be included in the response PDU is calculated. For more information on calculating the number of variables, Non-Repeaters, and Max-Repetitions, refer to RFC 3416.



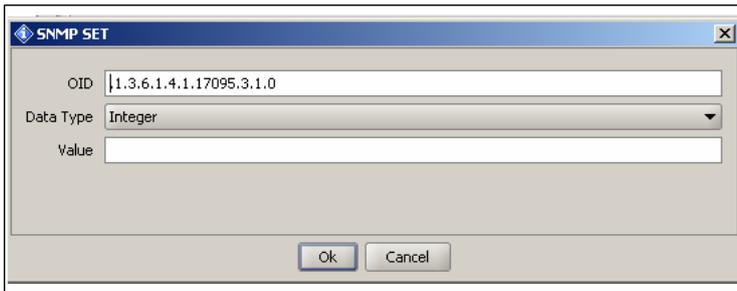
### Set

The Set command updates the variable information of the MIB database in the agent. The Set command can be performed only on those variables which are declared as 'READWRITE' in the MIB scripts, and only if the community name matches any one of the 'write' community names configured with the agent.

1. Select the `ledD5` variable from the MIB tree.
2. Configure the SNMP version to '1' or '2'. Configure the Write Community to 'public', 'write', or 'private'.
3. If version is configured to '3', configure the appropriate V3 credentials.
4. Select 'Set' from the Operations menu.



The SNMP SET window will pop up. Enter the value for the browser in the OID field.



A success message will appear.



A 'Get' operation for the same variable should now return the new 'Set' value for this variable. LED5 on the demo board should now be ON. Repeat the procedure to set LED5 to OFF. LED6 can also be set ON or OFF.

### 7.2.1.2.6.4 SNMP Traps

#### Module

Network Management (SNMP) Server (see page 98)

#### Description

The SNMP agent in version 5.25 and later of Microchip's TCP/IP Stack supports SNMP V1 and V2c formatted traps. Traps are notifications from the agent to the manager that are used when a predefined event occurs at the agent.

By default, traps are enabled in the agent. The following preprocessor macro in the `TCPIPConfig.h` header file can be used to disable traps in the agent:

```
#define SNMP_TRAP_DISABLED
```

The user must configure the expected trap format at the SNMP Manager. SNMPv2 entities acting as an agent should be able to generate and transmit SNMP V2 trap PDUs when the manager is configured to receive and process SNMP V2 trap PDUs. To configure the trap format, comment or uncomment the following macro in the `TCPIPConfig.h` header file:

```
#define SNMP_STACK_USE_V2_TRAP
```

If the macro has been commented out, the SNMP agent will send V1 formatted trap PDUs; otherwise, it will send V2 formatted trap PDUs. By default, the SNMP agent is configured to send V2 formatted traps. Note that the SNMP V2c agent should only send V2 formatted traps.

#### Demos

Two trap demos are included with the TCP/IP Stack. The task functions for these demos are called in the main application function:

- `SNMPTrapDemo()` - This API demonstrates V1 or V2 trap formats (depending on the status of the `SNMP_STACK_USE_V2_TRAP` macro). The trap PDU will only have one demo variable binding on the varbind list.
- `SNMPV2TrapDemo()` - This API provides V2 format notifications with multiple (3) variable bindings. The user should modify or use this routine as a reference for sending V2 trap format notifications with multiple bindings on the varbind list.

The user should only enable one SNMP demo API at a time. By default, the `SNMPTrapDemo()` API is enabled and `SNMPV2TrapDemo()` is commented out.

#### V1/V2 Formatted Traps with a Single Variable Binding

In the `TCPIPConfig.h` header file:

- Uncomment `#define SNMP_TRAP_DISABLED`

- Comment `//#define SNMP_STACK_USE_V2_TRAP`

For the Trap demonstration, two events are defined within the V2c agent:

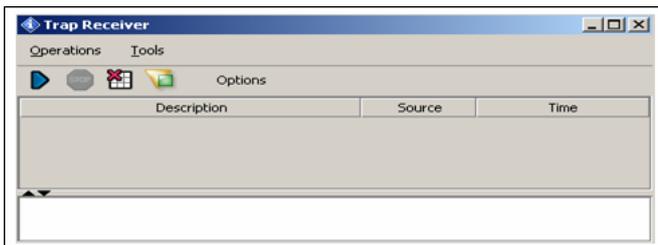
- If the Analog Potentiometer value is greater than 512, the agent will send a Trap every 5 seconds to the configured 'trapReceiverIPAddress.'
- If Button 3 on the demo board is pressed, an organization-specific PUSH\_BUTTON trap will be sent.

The current implementation of the V2c agent also generates a standard "Authentication (see page 84) Failure Trap":

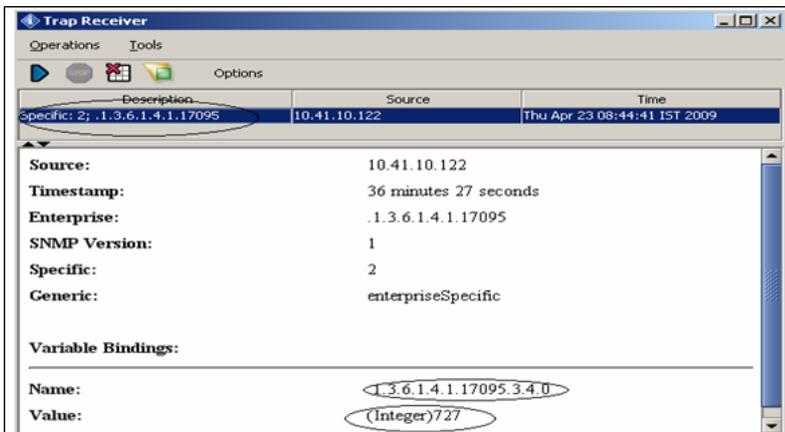
- If a request is received to modify (Set) a private MIB variable, or
- If the value of the variable is requested (get) by a browser with the wrong community name.

Procedure:

1. Open the "Advanced" configuration menu, configure the SNMP version to '2,' and configure the Write Community to "public', 'write', or 'private'.
2. Select the 'trapEnabled.0' variable from the MIB tree.
3. Select 'Set' from the Operations menu.
4. Enter '1' in the value field of the SNMP SET window.
5. Select 'trapReceiverIPAddress.0' from the MIB tree.
6. Set the value to the IP address of the PC on which the SNMP browser is installed and running.
7. Select 'trapCommunity.0' from the MIB tree.
8. Set the community name of the SNMP browser (the default community, if not set, is 'public'). The 'trapCommunity' name will work as a filter for the SNMP browsers on a trap-monitoring server.
9. Open the "Trap Receiver" utility that was installed with the iReasoning MIB browser (Start->Programs->iReasoning->MIB Browser->Trap Receiver).



To test the analog potentiometer trap, adjust the potentiometer on the demo board so the value is greater than 512 (turn it clockwise). This is an enterprise-specific trap. The SNMP Manager will receive the source IP address, the OID (as the name of the variable), the value, the timestamp, etc. for each event. The browser will interpret the data as AnalogPot variable information based on the OID name.

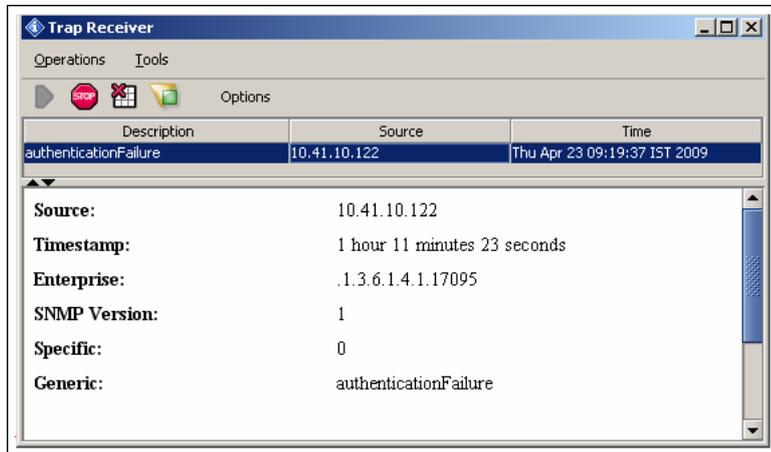


To test the push button trap, press the appropriate button on the development board (RB0 on the PICDEM.net 2 or S3 on the Explorer 16 board).

To test the Authentication (see page 84) Failure trap, configure the Read Community in your browser to a community name that is not supported by the agent (the default supported names are 'public' and 'read'). For example:

1. Configure 'mchp' as the Read Community name in the browser.
2. Select the private MIB variable LED5 from the MIB tree and issue a 'Get' operation from the browser.

The result table of the browser won't display any result, but the Trap Receiver will receive and Authentication (see page 84) Failure trap.



This is an intimation from the agent to the SNMP Manager that there was an unauthorized attempt to access the private MIB variable in the database.

### V2 Formatted Traps with a Multiple Variable Bindings

In the TCPIPConfig.h header file:

- Uncomment `#define SNMP_TRAP_DISABLED`
- Uncomment `#define SNMP_STACK_USE_V2_TRAP`

The SNMP V2 Trap PDU structure is:

Version (2) | community | SNMP-PDU pdu-type (TRAP=0xA7) | request-id | error-status | err-index | varbind List

The first two variable varbinds in the variable binding list of an SNMPv2-TRAP-PDU are sysUpTime.0 and snmpTrapOID.0, respectively. If any additional variables are to be included, then each of these varbind structures must be copied to the variable binding list.

For the SNMPv2 multiple TRAP variable varbind demonstration, ANALOG\_POT0 is used to generate an event and transmit an SNMP v2 Trap PDU. Adjust the analog potentiometer to a value greater than 512 (turn it clockwise) on the demo board. In addition to the sysUpTime.0 and snmpTrapOID.0 varbinds, the additional varbinds that are included with the trap PDU are:

- PUSH-BUTTON
- LED0\_IO
- ANALOG\_POT0

The following figure shows a V2 formatted trap with ANALOG\_POT0 as the variable binding to be notified.

Description	Source	Time
trapOID: .1.3.6.1.4.1.17095	10.41.10.78	Mon Apr 26 13:52:29 IST 2010
trapOID: .1.3.6.1.4.1.17095	10.41.10.78	Mon Apr 26 13:52:24 IST 2010
trapOID: .1.3.6.1.4.1.17095	10.41.10.78	Mon Apr 26 13:52:19 IST 2010
trapOID: .1.3.6.1.4.1.17095	10.41.10.78	Mon Apr 26 13:52:14 IST 2010

<b>Source:</b>	10.41.10.78
<b>Timestamp:</b>	1 minute 20 seconds
<b>Trap OID:</b>	.1.3.6.1.4.1.17095
<b>SNMP Version:</b>	2
<b>Variable Bindings:</b>	
<b>Name:</b>	.1.3.6.1.2.1.1.3.0
<b>Value:</b>	(TimeTicks)1 minute 20 seconds
<b>Name:</b>	snmpTrapOID
<b>Value:</b>	(OID).1.3.6.1.4.1.17095
<b>Name:</b>	.1.3.6.1.4.1.17095.3.4.0
<b>Value:</b>	(Integer)1023
<b>Description:</b>	

The next figure shows a multiple-variable varbind for an SNMP V2 Trap PDU, with the three additional variable bindings:

Description	Source	Time
trapOID: .1.3.6.1.4.1.17095	10.41.10.78	Mon Apr 26 14:35:33 IST 2010
<b>Source:</b>	10.41.10.78	
<b>Timestamp:</b>	2 minutes 15 seconds	
<b>Trap OID:</b>	.1.3.6.1.4.1.17095	
<b>SNMP Version:</b>	2	
<b>Variable Bindings:</b>		
<b>Name:</b>	.1.3.6.1.2.1.1.3.0	
<b>Value:</b>	(TimeTicks)2 minutes 15 seconds	
<b>Name:</b>	snmpTrapOID	
<b>Value:</b>	(OID).1.3.6.1.4.1.17095	
<b>Name:</b>	.1.3.6.1.4.1.17095.3.4.0	
<b>Value:</b>	(Integer)820	
<b>Name:</b>	.1.3.6.1.4.1.17095.3.3.0	
<b>Value:</b>	(Integer)1	
<b>Name:</b>	.1.3.6.1.4.1.17095.3.1.0	
<b>Value:</b>	(Integer)1	

## 7.2.1.2.6.5 HTTP Configuration

### Module

Network Management (SNMP) Server (see page 98)

### Description

If an HTTP2 server is used with the Microchip TCP/IP Stack, it is possible to dynamically configure the Read and Write community names through the SNMP Configuration web page. Follow the steps in the Getting Started section to upload the web pages to non-volatile memory, then access the SNMP Configuration web page through the navigation bar. Use "admin" for the username and "microchip" for the password.

### 7.2.1.2.6.6 Functions

#### Functions

	Name	Description
☞	SendNotification ( <a href="#">see page 110</a> )	Prepare, validate remote node which will receive trap and send trap pdu.
☞	SNMPGetTimeStamp ( <a href="#">see page 111</a> )	Obtains the current Tick value for the SNMP time stamp.

#### Module

Network Management (SNMP) Server ([see page 98](#))

#### 7.2.1.2.6.6.1 SendNotification Function

##### File

CustomSNMPApp.c

##### C

```
static BOOL SendNotification(
    BYTE receiverIndex,
    SNMP_ID var,
    SNMP_VAL val,
    UINT8 targetIndex
);
```

##### Description

This routine prepares the trap notification pdu, sends ARP and get remote device MAC address to which notification to sent, sends the notification. Notification state machine is getting updated if there is any ARP resolution failure for a particular trap destination address.

##### Remarks

None.

##### Preconditions

SNMPTrapDemo() is called.

##### Parameters

Parameters	Description
receiverIndex	The index to array where remote ip address is stored.
var	SNMP var ID that is to be used in notification

val	Value of var. Only value of BYTE, WORD or DWORD can be sent.
targetIndex	snmpv3 target index

**Return Values**

Return Values	Description
TRUE	If notification send is successful.
FALSE	If send notification failed.

**7.2.1.2.6.6.2 SNMPGetTimeStamp Function****File**

CustomSNMPApp.c

**C**

```
static DWORD SNMPGetTimeStamp();
```

**Description**

This function retrieves the absolute time measurements for SNMP time stamp. Use TickGet (see page 512) and TickGetDiv64K (see page 513) to collect all 48bits of the internal Tick Timer.

**Remarks**

None.

**Preconditions**

None

**Return Values**

Return Values	Description
timeStamp	DWORD timevalue

**7.2.1.2.6.7 Variables****Module**

Network Management (SNMP) Server (see page 98)

**Variables**

	Name	Description
◆	gSendTrapSMstate (see page 111)	This is variable gSendTrapSMstate.
◆	gSnmpv3UserSecurityName (see page 111)	This is variable gSnmpv3UserSecurityName.

**7.2.1.2.6.7.1 gSendTrapSMstate Variable****File**

CustomSNMPApp.c

**C**

```
UINT8 gSendTrapSMstate = 0;
```

**Description**

This is variable gSendTrapSMstate.

**7.2.1.2.6.7.2 gSnmpv3UserSecurityName Variable****File**

CustomSNMPApp.c

**C**

```
BYTE gSnmpv3UserSecurityName[USER_SECURITY_NAME_LEN];
```

**Description**

This is variable gSnmpv3UserSecurityName.

**7.2.1.2.6.8 Macros****Macros**

	Name	Description
	MAX_TRY_TO_SEND_TRAP ( <a href="#">see page 112</a> )	

**Module**

Network Management (SNMP) Server ([see page 98](#))

**7.2.1.2.6.8.1 MAX\_TRY\_TO\_SEND\_TRAP Macro****File**

CustomSNMPApp.c

**C**

```
#define MAX_TRY_TO_SEND_TRAP (10)
```

**Section**

Global Variables

\*\*\*\*\*

This Macro is used to provide maximum try for a failure Trap server address

**7.2.1.2.7 UART-to-TCP Bridge****Overview**

The UART-to-TCP bridge feature of the TCP/IP Demo App transmits all incoming TCP bytes on a socket out of the PIC's UART module and all incoming UART bytes out of a TCP socket.

**Instructions**

1. Compile your MPLAB project and program the demo board.
2. Connect the RS-232 port on your demo board to an RS-232 port on your computer. On a newer computer you may need an RS-232 to USB converter cable. On your computer, open a terminal program (such as HyperTerminal). Set it to use the COM port you connected your board to, at 19200 baud, with 8 data bits, no parity, 1 stop bit, and no flow control.
3. Connect the programmed demo board to a computer either directly or through a router. For Ethernet, a direct connection may require a crossover cable; for WiFi, the board may need to be in AdHoc mode to establish a direct connection.
4. Determine the IP address of the demo board. This can be done several different ways.
  1. If you are using a demo setup with an LCD display (e.g. Explorer 16 or PICDEM.net 2), the IP address should be displayed on the second line of the display.
  2. Open the Microchip Ethernet Device Discoverer from the start menu. Press the "Discover Devices" button to see the addresses and host names of all devices with the Announce ([see page 149](#)) Protocol enabled on your network. You may have to configure your computer's firewall to prevent it from blocking UDP port 30303 for this solution.
  3. If your board is connected directly with your computer with a crossover cable:
    1. Open a command/DOS prompt and type 'ipconfig'. Find the network adaptor that is connected to the board. The IP address of the board is located in the 'Default Gateway' field

2. Open up the network status for the network adaptor that connects the two devices. This can be done by right clicking on the network connection icon in the network settings folder and select 'status' from the menu. Find the 'Default Gateway' field.
5. Open a command/DOS prompt. Type "telnet ip\_address 9761" where ip\_address is the IP address that you got from step 4.

```

C:\WINDOWS\system32\cmd.exe
C:\>telnet 192.168.8.131 9761_

```

6. As you type characters in the command prompt, they will be transmitted over the Telnet (see page 481) TCP port to the PIC, and then transmitted out of the PIC's UART to appear on your terminal program. As you type characters in the terminal program, they will be transmitted to the PIC through the UART module, and then retransmitted over the TCP connection to appear in the command prompt Telnet (see page 481) session.

New IP Address: 169.254.1.1

New IP Address: 192.168.8.131

This string was typed in the telnet session.



```

Telnet mchpboard
This one was typed in the terminal.

```

### 7.2.1.2.8 Zero Configuration (ZeroConf)

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network. It also provides for a more human-like naming convention, instead of relying on IP addresses alone. Zeroconf also goes by the names Bonjour (Apple) and Avahi (Linux), and is an IETF standard.

#### Enabling

Zeroconf can be enabled by setting the following two defines in TCPIPConfig.h:

- STACK\_USE\_ZEROCONF\_LINK\_LOCAL
- STACK\_USE\_ZEROCONF\_MDNS\_SD

Currently, the use of Zeroconf is limited to the WiFi demo applications (and the MRF24WB0M module). Future versions of the stack should enable Zeroconf support across all Ethernet solutions.

#### Link Local

The first component of Zeroconf is the ability to self-assign an IP address to each member of a network. Normally, a DHCP server would handle such situations. However, in cases where no DHCP server exists, Zeroconf enabled devices negotiate unique IP addresses amongst themselves.

#### mDNS

The second component of Zeroconf is the ability to self-assign human-readable hostnames for themselves. Multicast DNS provides a local network the ability to have the features of a DNS server. Users can use easily remembered hostnames to access the devices on the network. In the event that devices elect to use the same hostname, as in the IP address resolution, each of the devices will auto-negotiate new names for themselves (usually by appending a number to the end of the name).

#### Service Discovery

The last component of Zeroconf is service discovery. All Zeroconf devices can broadcast what services they provide. For instance, a printer can broadcast that it has printing services available. A thermostat can broadcast that it has an HVAC control service. Other interested parties on the network who are looking for certain services can then see a list of devices that have the capability of providing the service, and connect (see page 165) directly to it. This further eliminates the need to know whether something exists on a network (and what its IP or hostname is). As an end-user, all you would need to do

is query the network if a certain service exists, and easily connect (see page 165) to it.

### Demo

The demo, when enabled, shows all three items above working together. Each development kit in the network assumes the hostname of MCHPBOARD-x.local, where x is an incrementing number from 1 (only in the case where multiple kits are programmed for the network). Each board will broadcast it's service, which is the DemoWebServer.

### Zeroconf Enabled Environments

All Apple products have Zeroconf enabled by default. On Windows, you'll need to download the Safari web browser, and during the install, enable support for Bonjour. Note that in the Safari browser, you can browse and see a list of all Bonjour enabled devices, and click through to them automatically.

---

## 7.2.2 Internet Bootloader

The Internet Bootloader is a stand alone application allowing new application firmware to be uploaded directly into the Flash memory of a PIC18F microcontroller over an Ethernet network or the Internet. For other PIC and dsPIC architectures, third-party TCP/IP bootloaders can be obtained from <http://www.brushelectronics.com/>. This Internet Bootloader application implements its own private UDP/IP stack as well as a Trivial File Transfer Protocol (TFTP) server. The bootloader operates independently of the main application and cannot update itself. Safeguards are implemented internally to minimize the risk of non-recoverable failed upgrades.

Important attributes of the Internet bootloader include:

- Self contained TFTP, UDP, IP, ARP, and Ethernet protocol handling
- Executes on Power-on Reset instead of during main application
- Waits approximately 4 seconds before starting main application
- Requires 8KB of program Flash
- Requires 0B of RAM (all used RAM is overlaid with main application)
- Requires no CPU time while executing main application
- Requires minimal or no changes to main application code and linker script
- Does not interfere with application interrupt vector locations or add interrupt latency
- Can reprogram configuration words
- Can reuse MAC and IP address provided by main application
- Client update software is already available on most computers

### 7.2.2.1 Bootloader Design

#### Bootloader Entry

The bootloader is a TFTP server which starts automatically on Power-on Reset (POR). It can be located anywhere within program memory. To cause the automatic startup, the bootloader transparently performs a replacement of the instruction(s) at program memory locations 000000h-000003h. The .hex file to be programmed to the chip by the bootloader will normally contain a GOTO instruction at address 000000h which branches to the main application. Instead of writing the original instruction at address zero, the bootloader creates a new GOTO instruction which always branches to the start address of the bootloader code. The original application instruction at address zero is moved to a jump table, which is later called to exit the bootloader. The jump table also contains a GOTO 000004h instruction to ensure normal application operation if the first instruction was not a GOTO.

If the device is programmed with only the bootloader (no application), address 000000h through the start address of the bootloader code will be in an unprogrammed state (FFFFh). These are NOP instructions which will quickly execute until the

program counter reaches the start of the bootloader. This ensures entry into the bootloader for both programmed and unprogrammed parts.

### Bootloader Re-entry

If the running application wants to reenter the bootloader, it should clear the `RCON<NOT_POR>` bit and then execute a `RESET` instruction. When the bootloader returns control to the main application, the `NOT_POR` bit will be in the set state. If an application needs to reset quickly without waiting for the bootloader timeout, it should leave this bit in the set state. This will cause the bootloader to skip its normal operation and return immediately to the application.

Prior to executing the `RESET` instruction to reenter the bootloader, the main application can specify the MAC and IP address for the bootloader to use. To do this:

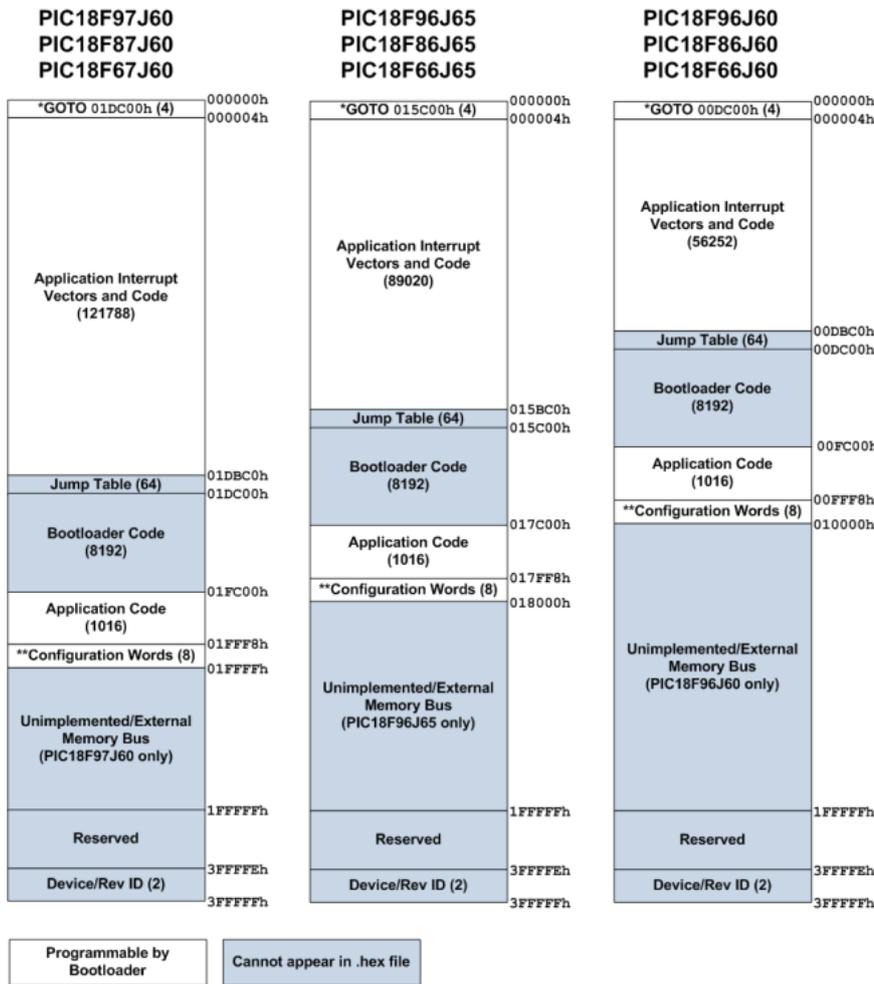
1. Select a random memory location where 12 bytes can be written
2. Copy the MAC address to the chosen memory location at offset 0
3. Copy the IP address to the chosen memory location at offset 6
4. Compute an IP checksum of the MAC and IP address stored at the memory locations 0 through 9
5. Write the IP checksum at the chosen memory location at offset 10
6. Store the address of the chosen memory location into the `PRODH:PRODL` registers
7. Clear the `RCON<NOT_POR>` bit and execute a `RESET` instruction to enter the bootloader

Upon entry, the bootloader will detect that the `RCON<NOT_RI>` bit is clear, indicating the bootloader was entered from the main application instead of a genuine POR event. In this case, the bootloader will dereference the `PRODH:PRODL` pointer, validate the checksum, and if valid, use the MAC and IP address specified. If the checksum is invalid, the bootloader will use its default compiled MAC and IP address.

The Microchip TCP/IP Stack library provides a Reboot (see page 311) module which can perform the above procedure upon detection of a TFTP packet. When the Reboot (see page 311) module is used, the application IP address (possibly obtained automatically via DHCP) can be used as the TFTP bootloader target. If the IP address used by the main application is Internet routable, then the bootloader itself will be accessible via the Internet.

### Memory Map

The entire program memory map when using the bootloader is shown below.



\*: GOTO instruction is automatically generated by bootloader when writing. Application instruction at address 000000h is moved to Bootloader Jump Table.

\*\* : Some configuration options are not supported and will automatically be changed by the bootloader before flashing. The bootloader requires HS or HS+PLL oscillator mode and does not support 1:1 and 1:2 Watchdog Timer Postscale modes. Switching between Extended and non-Extended mode is not supported either. (The bootloader must be recompiled to change modes.)

The bootloader uses a block of 8256 bytes of program memory. To prevent the application from inadvertently using this block, you should modify the linker script in your application prior to compiling. For example, for the MPLAB<sup>®</sup> C18 C compiler, the linker script will contain a CODEPAGE line describing the available Flash memory in the device. For the PIC18F97J60 product with 128kB of program memory, the linker script (18f97j60i.lkr) will contain a line such as:

```
CODEPAGE NAME=page START=0x2A END=0x1FFF7
```

This line indicates that the linker can place application constants and code anywhere between 00002Ah and 01FFF7h. This line must be split into two CODEPAGE lines to describe the gap in available program memory occupied by the bootloader. Ex:

```
CODEPAGE NAME=page START=0x2A END=0x1DBBF
CODEPAGE NAME=page2 START=0x1FC00 END=0x1FFF7
```

The above example removes the Jump Table and Bootloader Code blocks for the PIC18F97J60 with 128kB of Flash memory. Other devices with less Flash memory will need to use different start and end values according to the Jump Table start address and Bootloader Code end address described in the memory map figure above.

**Erase Operations**

The TFTP server performs a "bulk erase" before starting any TFTP put (write) operation. The erase is not a true bulk erase

because the bootloader and configuration words remain intact. However, all other locations are reverted to their unprogrammed state. The erase procedure starts with the Flash page containing the Jump Table and continues backwards in memory towards address 000000h. After address 000000h is erased, the last program memory page containing the device configuration words is erased. For example, assuming a PIC18F97J60 with 128kB of Flash, the erase procedure will follow these steps:

1. Erase 01D800h-01DBFFh
2. Erase 01D400h-01D7FFh
3. Erase 01D000h-01D3FFh
4. ...
5. Erase 000400h-0007FFh
6. Erase 000000h-0003FFh
7. Erase 01FC00h-01FFFFh

After the last page containing the configuration words are erased, the configuration words are immediately reprogrammed to their previous value. This algorithm provides very robust operation with an extremely low likelihood of destroying access to the bootloader due to an unexpected event (ex: power or network connectivity is lost while bootloading). Unexpected events will leave the first GOTO instruction at address 000000h intact, ensuring that the bootloader will start up again. Because the configuration words are erased last, there will not be any means of circumventing the internal code protect feature while application code still remains in the device.

### Program Operations

Program operations are performed sequentially starting at address 000000h and growing upwards, as presented in the .hex file to be programmed. The device configuration words are typically the last values encountered in the .hex file. Because the erase procedure involves clearing the configuration words and then immediately reprogramming them, the configuration words will already be programmed by the time the configuration words are encountered in the .hex file. Therefore, if the .hex file contains different configuration words from what are already stored in the Flash memory, the bootloader will have to perform a new erase operation on the last page prior to programming the new configuration words. This extra erase/write cycle will reduce overall Flash endurance on the last page as compared to the rest of the device. However, the bootloader will not perform this erase/write if the configuration words have not changed. This feature preserves endurance for most application firmware upgrades, which typically do not require different configuration options to be programmed.

### Read Operations

To save code space, the bootloader currently only supports reading through the TFTP server as binary data. Instead of getting a .hex file from a TFTP get operation, the bootloader will send back a binary file sized to the amount of internal Flash memory available (128kB for PIC18F97J60, 64kB for PIC18F66J60, etc.). The bootloader verifies code immediately after programming devices, so the read feature is primarily for debugging only.

Read operations are disabled if the currently programmed application has the PIC<sup>®</sup> microcontroller Code Protect feature turned on.

## 7.2.2.2 Using the Bootloader

### Operation

After the bootloader has started, the code will enable the Ethernet module and begin running a private UDP/IP stack. It will use the following default addresses out of Power-on Reset:

- IP Address (see page 141): 192.168.97.60
- MAC Address (see page 141): 00-04-A3-00-00-00

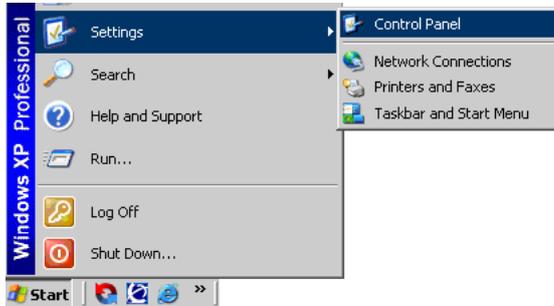
These default addresses are statically defined and can only be changed by recompiling the bootloader itself. However, if the bootloader is called from the main application, such as with the Reboot (see page 311) module, then the bootloader will use the application assigned IP and MAC addresses (if provided).

The only services that are available during bootloader operation are TFTP and ARP. ICMP (ping) and other services are not implemented.

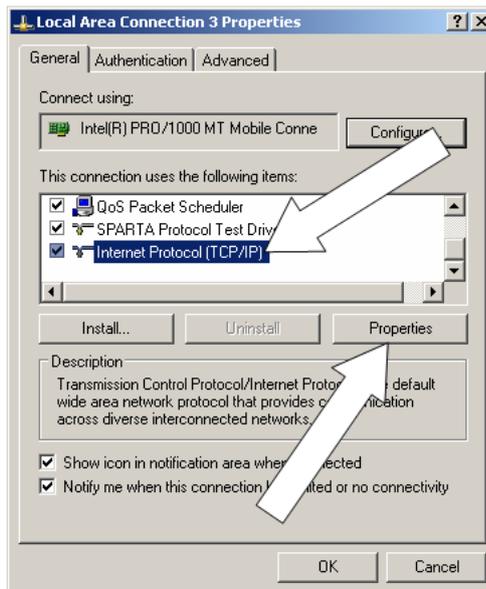
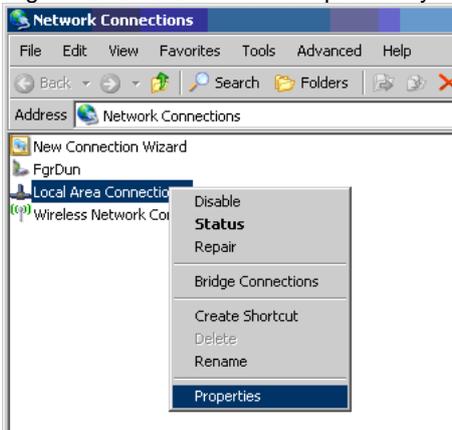
### Configuring Your PC (Power-on Reset entry)

To access the bootloader, the bootloader's IP address must be on the same subnet as your computer. For the default 192.168.97.60 IP address, you must temporarily change the settings on your PC. If the bootloader's IP address was application specified and already on the same subnet as your PC, then this section should be skipped.

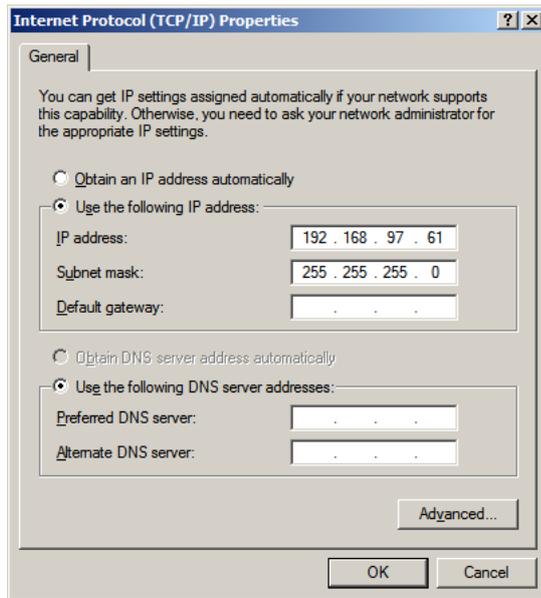
The following instructions assume you are using Microsoft® Windows® XP and will vary for other operating systems.



1. Open **Network Connections**.
2. Right click on the network adapter that you are using to communicate with the bootloader and choose **Properties**.



3. Select **Internet Protocol (TCP/IP)** and click **Properties**.
4. Select **Use the following IP address** and then enter the IP address **192.168.97.61**.



5. Click **OK** and then **Close** on the previous dialog to close them and set the new address.

### TFTP Operation (Power-on Reset entry)

Most operating systems come with a TFTP client built in. In Microsoft Windows, this utility is named `tftp.exe`. This utility is a very simple console application which can be used to upload your application .hex file over the network to the bootloader. To perform a Flash upgrade using the `tftp.exe` client, follow these procedures:

1. At a console, type the following command, but do not execute it. Make appropriate path changes to the .hex file. `tftp 192.168.97.60 put "C:\Microchip Solutions\TCPIP Demo App\TCPIP Demo App-C18.hex"`
2. Power cycle the target board or if the device has a MCLR reset button, press it.
3. Quickly press enter to execute the `tftp` command. If firmware is already in the device, the bootloader will automatically terminate after approximately 4 seconds, so you must execute the `tftp` command within the 4 second window.
4. If successful, the TFTP client will indicate how long the transfer took. Actual programming time will vary based on numerous factors, including need to erase the Flash first, .hex file size, .hex file complexity, and internal programming time. The reported transfer rate is therefore not a good metric of network performance in embedded applications.

```

c:\>tftp 192.168.97.60 put "C:\Microchip Solutions\TCPIP Demo App\TCPIP Demo App-C18.hex"
Transfer successful: 153591 bytes in 8 seconds, 19198 bytes/s
c:\>

```

The bootloader does data read back verification shortly after writing and does not need a second step to read back the Flash contents. If a verification error occurs, the error will be immediately reported to the TFTP client.

```

c:\>tftp 192.168.97.60 put "C:\Microchip Solutions\TCPIP Demo App\TCPIP Demo App-C30.hex"
Error on server : Verification failed. Attempting to overwrite bootloader memory?
c:\>

```

The most likely cause of a verification failure is not a Flash endurance problem, but rather, an invalid .hex file given as input. As shown in the bootloader memory map, .hex files cannot contain any data within the 8KB area of Flash where the bootloader is stored. The bootloader internally masks off this region of Flash and treats it as read only to prevent bootloader corruption. As a result, if the .hex file contains data in the read-only region, the write will fail and verification will show a mismatch.

5. After a successful write, the bootloader will time out after approximately 4 seconds and begin executing the main application that was just loaded.

After completing the TFTP upload process, restore your PC's IP address settings to allow normal network activity and access to the application you bootloaded.

### TFTP Operation (Application entry)

If using an application which auto-detects TFTP packets and enters the bootloader as needed, such as the Reboot (see

page 311) module in the Microchip TCP/IP Stack, then there will generally be no need to reconfigure your PC or go through a time-sensitive power cycling process. Instead, you can execute a TFTP operation directly on the device without any interactive steps.

1. At a console, type the following command and execute it. Make appropriate IP address/hostname and path changes.

```
tftp mchpboard put "C:\Microchip Solutions\TCPIP Demo App\TCPIP Demo App-C18.hex"
```

If the bootloader process is interrupted due to a network failure or user cancellation, you can simply retry the tftp command. The bootloader will not attempt to run a partially bootloaded application. The application specified MAC and IP address will be retained indefinitely until the device is power cycled or otherwise reset.

If the bootloader operation is interrupted due to a power failure, the bootloader will start back up using the Power-on Reset default MAC and IP addresses. In this case, you must follow the Power-on Reset entry directions to recover.

---

## 7.2.3 WebVend

The TCPIP WebVend App is a sample web-enabled vending machine application. It is used by the TCP/IP Webinar series:

1. TCP/IP Networking Part 1: Web-Based Status Monitoring ([view](#))
2. TCP/IP Networking Part 2: Web-Based Control ([view](#))
3. TCP/IP Networking Part 3: Advanced Web-Based Control ([view](#))

---

## 7.2.4 Internet Radio

**IMPORTANT: Because of changes to the SHOUTcast protocol, the Internet Radio demo app is no longer able to perform its intended function. This demo now exists only as an TCP/IP Stack code example.**

The Internet Radio app demonstrates the use of the TCP/IP Stack for a stand-alone embedded application. This application is capable of contacting various [SHOUTcast](#) servers and playing back the audio stream to a pair of stereo speakers. The demo requires the Internet Radio Demonstration board. A PIC18F67J60 is used for the processing of Ethernet interface, while an external MP3 decoder handles the audio playback. Application note [AN1128](#) "TCP/IP Networking: Internet Radio Using OLED Display and MP3 Audio Decoder (DS01128)" describes the Internet Radio application in detail.

To run the demo, first make sure the Internet Radio board has the correct firmware programmed. Next, connect ([see page 165](#)) the board to the internet, plug in an audio headset or speaker. By default, the program will not play a radio station automatically until a genre is selected. Follow the OLED display's on screen menu to change genre, station, and volume.

The board can also be controlled via the web browser interface. To connect ([see page 165](#)) to the board's web server, use the IP address shown on the board's OLED display. Shown below is a screen shot of the webpage. To start, first select a genre from the drop down list box, and click 'Select'. To change station, click 'Prev' or 'Next'. To adjust volume, click 'Down' or 'Up'. If a station does not play, it could be that the port is blocked, try a different station.

Each Internet Radio board also has a sticker containing a unique MAC address. This unique MAC address can be saved to the board by using the web interface's configurations section.



## 7.2.5 WiFi Console

The TCPIP WiFi Console Demo App (previously the TCPIP WiFi Iperf Demo App) is a command line interface (CLI) to the MRF24WB0M. It allows for command line debugging and setup of network information for the wireless LAN. It also has iperf built in for doing WLAN bandwidth testing. This application is meant more as a development debug tool, and should be disabled in end user applications.

### 7.2.5.1 Standalone Commands

These CLI commands are not related to the wireless or networking interface directly.

help

Lists all the available CLI commands for the MRF24WB0M.

getwfvver

Lists the WiFi firmware and host driver version numbers.

```
reset
```

Issues a host reset.

```
cls
```

Resets the prompt.

```
kill iperf
```

Kills a running iperf session. See the section on iperf ([see page 125](#)) for more information.

## 7.2.5.2 iwconfig Commands

Note that most of these items should not be changed while the device is in a connected state to a network.

iwconfig commands take the following structure:

```
iwconfig [ ssid <name> ]
         [ mode <idle|managed|adhoc> ]
         [ channel <channel list|all> ]
         [ power <reenable|disable|unicast|all> ]
         [ domain <name> ]
         [ rts <length> ]
         [ txrate <0|1|2> ]
         [ scan ]
         [ hibernate ]
         [ wakeup ]
```

Note: iwconfig with no options will display wireless status.

ssid	
name	1-32 ASCII characters.  Currently doesn't accept ( <a href="#">see page 163</a> ) spaces in the SSID name.
mode	
idle	Forces the MRF24WB0M to disconnect from any currently connected network (adhoc or infrastructure).
managed	The MRF24WB0M will connect ( <a href="#">see page 165</a> ) to the SSID in infrastructure mode. Note that all the network parameters must be correct before this command is called.
adhoc	The MRF24WB0M will connect ( <a href="#">see page 165</a> ) to the SSID in adhoc mode. Note that all the network parameters must be correct before this command is called.
channel	
channel list	A comma separated list of all the channels to scan.
all	Sets the MRF24WB0M to scan all channels in the given regulatory domain.
power	
reenable	Enables all power saving features (PS_POLL) of the MRF24WB0M.
disable	Disables any power savings features. The MRF24WB0M will always be in an active power state.

unicast	The MRF24WB0M will be in it's deepest sleep state, only waking up at periodic intervals to check for unicast data. The MRF24WB0M will not wake up on the DTIM period for broadcast or multicast traffic.
all	The MRF24WB0M will wake up to check for all types of traffic (unicast, multicast, and broadcast).
domain	
fcc	United States channels 1-11.
ic	Canada channels 1-11.
etsi	European channels 1-13.
spain	Spanish channels 10-11.
france	French channels 10-13.
japana	Japanese channel 14.
japanb	Japanese channels 1-11.
rts	
length	Set the requested number of bytes to send. Default max is 2347.
txrate	
0	The MRF24WB0M will do automatic rate adaptation between 1 and 2 Mbps.
1	Sets the MRF24WB0M to fixed data rate of 1Mbps.
2	Sets the MRF24WB0M to fixed data rate of 2Mbps.
scan	
	Instructs the MRF24WB0M to perform an active site scan. Scan results will be displayed to the output terminal.
hibernate	
	Removes power to the MRF24WB0M.
wakeup	
	Restores power to the MRF24WB0M and reconnects.

Note: scan is only supported by the WiFi EZConfig demo.

### 7.2.5.3 ifconfig Commands

Note that these items should not be changed while the device is in a connected state to a network.

ifconfig commands take the following structure:

```
ifconfig [ <IP address> ]
         [ <MAC address> ]
         [ netmask <IP address> ]
         [ gateway <IP address> ]
         [ auto-dhcp <start|drop> ]
```

Note ifconfig by itself will give network status.

IP address	
	Use a static IP address. IP address must be in dot-decimal notation.  Note that this command will return an invalid parameter if the DHCP client is enabled. First disable the DHCP attempts ( <code>ifconfig auto-dhcp drop</code> ) before running this command.
MAC address	
	Redefine the device MAC address. MAC address must be specified in hexadecimal colon notation.  This command can only be issued when the MRF24WB0M is in idle mode. Doing so at other times can have unexpected results.
netmask	
IP address	Use the specified IP address for the netmask. The netmask value is specified in dot-decimal notation.
gateway	
IP address	Configure the gateway address. The gateway value is specified in dot-decimal notation.
auto-dhcp	
start	Starts the DHCP client.  Only valid if the DHCP module has been compiled in. DHCP client is started by default.
drop	Stops the DHCP client. A static IP address will need to be assigned to the device.  Only valid if the DHCP module has been compiled in.

### 7.2.5.4 iwpriv Commands

Note that these items should not be changed while the device is in a connected state to a network.

iwpriv commands take the following structure:

```
iwpriv [ enc <none|wep|wpa-psk|wpa-phrase> ]
       [ key <[1][2][3][4]> <value> ]
       [ psk <value> ]
       [ phrase <value> ]
```

Note iwpriv by itself will display network security settings.

enc	
none	The MRF24WB0M will not use any encryption to connect (see page 165) to the specified network.
wep	The MRF24WB0M will use either WEP-40 (short) or WEP-104 (long) encryption to connect (see page 165) to the specified network.
wpa-psk	The MRF24WB0M will use the specified 32-byte PSK to connect (see page 165) to the WPA/WPA2 network.

wpa-phrase	The MRF24WB0M will take the given 1-32 ASCII character passphrase, along with the SSID, and compute the required 32-byte PSK for the network.  Note that doing so takes approximately 30 seconds to complete the calculation.
key	
[ 1 ] [ 2 ] [ 3 ] [ 4 ]	Instructs the MRF24WB0M to use this key for connecting to the WEP encrypted network.  Note that only key 1 is considered safe to use among different AP vendors. Keys 2-4 can have implementation specific entries that may not be compatible from AP to AP.
value	If value is specified, this will instruct the MRF24WB0M to use the specified key number and also program the device with this key value. For WEP-40 networks, this implies the key is either 5 ASCII characters or 10 hex characters in length. For WEP-104 networks, this implies the key is either 13 ASCII characters or 26 hex characters in length.  The console only accepts hex WEP keys. Therefore, the user must do the ASCII to hex conversion for their ASCII keys.
psk	
value	32-byte hex value for the PSK.  This value can be calculated from the following website hosted on the <a href="#">Wireshark website</a> .
phrase	
value	An 8-63 ASCII character phrase (delimited with quotes if using spaces). This phrase will be used along with the SSID to generate the 32-byte PSK value for the network.

### 7.2.5.5 iperf Example

iperf is a networking tool that helps to measure networking bandwidth and performance. The console demo application has a built-in iperf application, that can act as both a client and server for testing. iperf has the ability to test both UDP and TCP. In the case of UDP, you can specify the size of the UDP datagrams. For TCP, iperf measures the throughput of the payload.

In order to run iperf, you'll need a PC that has an iperf application on it as well. There is an open source version that is maintained, as well as many other variants across the internet. iperf is meant to be run at the command line. However, if a GUI is desired, a variant called jperf can be used.

In the case of the demo application, iperf measures performance data in a unidirectional format. Therefore, the side that the server is running on is considered the receiver, and provides the most accurate performance values.

#### Command Synopsis

iperf	[ -s -c <IP addr> ] [ -u ] [ -i <sec> ] [ -b <bandwidth> ] [ -t <time> ]
-s	Runs the iperf server. No IP address needs to be specified.
-c <IP addr>	Runs the iperf client. The IP address is the IP address of the server.

-u	Server side only. Sends UDP datagrams.
-i <sec>	Specified the time interval, in seconds, that the display will be updated.
-b <bandwidth>	Specifies the amount of data to try and send. This option is only valid with UDP datagrams.
-t <time>	Amount of time, in seconds, to run the test.

## Running the Demo

After powering on the development board and associating with your wireless network, you'll need to start the server side iperf application first. If you start iperf as a server on the development board in the console, then this implies that you are trying to measure the MRF24WB0M receiver performance. If you start the iperf server on a PC, then you will be measuring MRF24WB0M transmit performance. Below are two images that show receiver and transmitter performance, respectively.

```

C:\Windows\System32>cmd.exe
C:\Windows\System32>iperf -c 192.168.1.51 -b 2M -i 1 -t 5
WARNING: option -b implies udp testing
-----
Client connecting to 192.168.1.51, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
-----
[1912] local 192.168.1.66 port 1183 connected with 192.168.1.51 port 5001
[ ID] Interval      Transfer      Bandwidth
[1912] 0.0- 1.0 sec   211 KBytes    1.73 Mbits/sec
[1912] 1.0- 2.0 sec   223 KBytes    1.82 Mbits/sec
[1912] 2.0- 3.0 sec   121 KBytes    988 Kbits/sec
[1912] 3.0- 4.0 sec   112 KBytes    917 Kbits/sec
[1912] 4.0- 5.0 sec   89.0 KBytes   729 Kbits/sec
[1912] 0.0- 5.0 sec   757 KBytes    1.23 Mbits/sec
[1912] Server Report:
[1912] 0.0- 5.6 sec   478 KBytes    697 Kbits/sec  0.000 ms 194/ 527 (37%)
[1912] Sent 527 datagrams
C:\Windows\System32>_

>
> iperf -s -u -i 1
> -----
Server listening on UDP port 5001
Session started ...
- Local 192.168.1.51 port 5001 connected with
- Remote 192.168.1.66 port 1183
- [ 0- 1 sec] 0/ 47 ( 0%)    553 Kbps
- [ 1- 2 sec] 92/ 155 (59%)  737 Kbps
- [ 2- 3 sec] 25/ 90 (27%)   757 Kbps
- [ 3- 4 sec] 34/ 94 (36%)   706 Kbps
- [ 4- 5 sec] 36/ 96 (37%)   706 Kbps
- [0.0- 5.6 sec] 194/ 527 (36%)    698 Kbps
Session completed ...

Rx done. Socket closed.
Ready for the next session.

```

```

C:\Windows\System32\cmd.exe - iperf -s -u -i 1
C:\Windows\System32>iperf -s -u -i 1
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
-----
[1928] local 192.168.1.66 port 5001 connected with 192.168.1.51 port 4097
[ ID] Interval      Transfer       Bandwidth       Jitter    Lost/Total Datagrams
[1928] 0.0- 1.0 sec  51.7 KBytes    423 Kbits/sec   37.858 ms  909586484/ 36 (2.5e+009%)
[1928] 1.0- 2.0 sec  63.2 KBytes    517 Kbits/sec   33.919 ms  0/ 44 (0%)
[1928] 2.0- 3.0 sec  76.1 KBytes    623 Kbits/sec   28.374 ms  0/ 53 (0%)
[1928] 3.0- 4.0 sec  108 KBytes     882 Kbits/sec   16.877 ms  0/ 75 (0%)
[1928] 4.0- 5.0 sec  84.7 KBytes    694 Kbits/sec   19.388 ms  1/ 60 (1.7%)
[1928] 0.0- 5.0 sec  386 KBytes     629 Kbits/sec   18.872 ms  1/ 270 (0.37%)

> iperf -c 192.168.1.66 -b 2M -i 1 -t 5
-----
Client connecting to 192.168.1.66, UDP port 5001
-----
Session started ...
- Local 192.168.1.51 port 4097 connected with
- Remote 192.168.1.66 port 5001
- Target rate = 20000000 bps, period = 6 ms
- [ 0- 1 sec] 0/ 43 ( 0%) 497 Kbps
- [ 1- 2 sec] 0/ 41 ( 0%) 469 Kbps
- [ 2- 3 sec] 0/ 54 ( 0%) 620 Kbps
- [ 3- 4 sec] 0/ 79 ( 0%) 922 Kbps
- [0.0- 5.0 sec] 0/ 270 ( 0%) 636 Kbps
- [0.0- 5.2 sec] 0/ 279 ( 0%) 624 Kbps
Session completed ...
Tx done. Socket closed.
Iperf completed.

```

## 7.2.6 WiFi EZConfig

### Overview

WLAN networks provide a unique challenge for configuring embedded wireless without a natural user interface. Unlike wired networks, wireless networks require unique items such as the SSID and network type and keys, which have to be sent to the device in some form or another. Traditionally, this means a user would enter this information using a keyboard and display.

EasyConfig is a mechanism to allow for configuration of an embedded device on a wireless network. It utilizes the web server of the TCP/IP stack, as well as a wireless adhoc (IBSS) network to allow the user to input the desired network information from a client browser, and then reset the device to connect (see page 165) to the desired network.

The EasyConfig demo works in roughly the following manner:

1. Upon power up the device, it broadcasts an adhoc network with SSID "EasyConfig".
2. A client device (laptop, iPod Touch/iPhone/iPad) can then connect (see page 165) to the EasyConfig network.
3. Upon connecting, the user can then use a standard web browser to go to the IP address of the demo (<http://169.254.1.1>).
4. The user will then be presented with some web pages from the web server. The index.htm web page has some additional information on EasyConfig, and also shows the continually updating status of the LEDs, buttons, and potentiometer on the development board. The configure.htm page will allow the user to scan for networks, and connect (see page 165) to a network of their choosing.
5. The device will then reset itself, using the parameters for the new network. In order to continue using the demo, the client device will now need to reconnect to the same network that the development board is on.

Note that the demo will always attempt to connect (see page 165) to the last known network. If the user wants to reset the demo to startup in adhoc mode again, then button S3 on the Explorer 16 development board needs to be held down for 4 seconds.

## Adhoc Networks

Upon starting the demo, the network will either connect (see page 165) to another adhoc network, or will start its own if one is not found. Adhoc networks are peer-to-peer networks, with no centralized coordinator for the network. All the devices in the network share the responsibilities of keeping the network going.

One downfall of adhoc networks is that typically security is not employed on them. The MRF24WB0M module can secure the network with WEP (40-bit/104-bit) security, as can most laptops and adhoc devices. Almost no devices in the market can secure an adhoc network with WPA level security due to the tremendous overhead in doing so.

The demo starts an adhoc network with no security. This means that all the network information that is being configured on the device is going over-the-air in the open. For most applications, unless somebody is specifically attempting to eavesdrop on this network, there should be little to no impact on security. However, for applications that do require some baseline level of security, then WEP can be employed on the network. SSL can also be used to encrypt the traffic between the web server and client browser. Additionally, some other form of data-level security can be employed to obfuscate the ASCII network information being sent to the device.

## Network Parameters

Below is some information on the parameters that are being sent via HTTP POST from the client browser to the device. All this information is being parsed and handled in the function `HTTPPostWifiConfig()` in `CustomHTTPApp.c`.

WLAN Type	Either adhoc or infrastructure
SSID	Name of network (1-32 ASCII characters)
Security Type	<ul style="list-style-type: none"> <li>• None</li> <li>• WEP-40 (5 ASCII characters or 10 hex numbers)</li> <li>• WEP-104 (13 ASCII characters or 26 hex numbers)</li> <li>• WPA/WPA2 passphrase (8-63 ASCII characters)</li> </ul>

## Configured vs Un-configured State

When the demo is running in an unconfigured state (i.e, serving the default EasyConfig SSID in adhoc mode), then the heartbeat LED (LED0) will blink twice per second to indicate that it hasn't been configured yet. Once the network has been configured, then the heartbeat LED will change to blink once per second, in a similar fashion to the other TCP/IP demo applications.

## EasyConfig Demo Additional Features

There are four defines that enable EasyConfig as well as extend it with natural features.

<code>STACK_USE_EZ_CONFIG</code>	The top level define to enable EasyConfig features.
<code>EZ_CONFIG_SCAN</code>	Adds additional ability to instruct the MRF24WB0M module to scan for available networks nearby. This can be done when you are already connected to a network.
<code>EZ_CONFIG_STORE</code>	Store the configuration information for the new network to non-volatile memory. In the event that WPA/WPA2 level security is used, the 32-byte PSK will be saved to NVM.
<code>EZ_CONFIG_STALL</code>	Before switching networks, forces the configuration state machine to pause. This gives the client device additional time to request resources from the development platform before it attempts to connect (see page 165) to a new network.

### **EZ\_CONFIG\_SCAN**

The MRF24WB0M has the ability to scan for nearby networks. This is similar to a laptop that can show available wireless networks that can be connected to. The scan results are stored on the MRF24WB0M module, and can be retrieved one at a time from the device. This helps to reduce the impact of storing all the scan results on the host itself.

The scan can be performed when idle, or when connected to either an adhoc or infrastructure network.

### **EZ\_CONFIG\_STORE**

The new network parameters can also be stored to non-volatile memory. For the Explorer 16 development board, this information is stored to the 32kB EEPROM on the board.

One extremely useful feature of storing the information surfaces when connecting to a network with WPA/WPA2 security. The computation of the 32-byte PSK is computationally heavy, and can take the MRF24WB0M up to 30 seconds to calculate the key. In a normal application, it would be unacceptable to have to wait 30 seconds every time the device started up before connection to the network was established.

`EZ_CONFIG_STORE` helps to alleviate doing the calculation each time by storing the 32-byte PSK to NVM. In doing so, there is only one 30-second hit the very first time the key is calculated only. Successive connections to the network will be significantly faster.

### **EZ\_CONFIG\_STALL**

The configuration state machine that controls the network connections within EasyConfig can employ a wait state between switching networks. From an end user experience, this becomes vital. If the switch between different networks was instantaneous, a client browser would never get an indication that the HTTP session was closed after the POST information was sent. The end user would see this as a browser that was continually waiting, which would eventually timeout.

To make the switch more natural and complete, `EZ_CONFIG_STALL` adds additional time to allow the client to get the remaining web page information. For the demo, this includes a HTTP redirect to a page that highlights the new network information.

### **Current Incompatibilities**

The javascript being used in EasyConfig is not compatible with Internet Explorer 7. EasyConfig does work with many other flavors of browser on different architectures, not limited to Internet Explorer 8, Mozilla Firefox, Apple Safari and Google Chrome. The incompatibility is something that is being investigated, and should be fixed in a future stack release.

---

## **7.2.7 Demo App MDD**

The TCPIP MDD Demo App is a variant of TCPIP Demo App that uses an SD card or USB Thumb Drive to store web pages. For more information, see the TCPIP MDD Demo App Getting Started guide, installed in the stack's documentation folder.

---

## **7.2.8 Google PowerMeter**

This demo implements a power monitoring application that uploads data to Google PowerMeter. For more information about this demo application, consult the help file "Reference Implementation for Google PowerMeter Help.chm/pdf" in the Microchip Application Libraries help folder.

## 7.2.9 Google PowerMeter EZConfig

This demo integrates the device configuration methodology found in the WiFi EZConfig demo into the basic Google PowerMeter demo. For more information about this demo application, consult the help file "Reference Implementation for Google PowerMeter Help.chm/pdf" in the Microchip Application Libraries help folder.

---

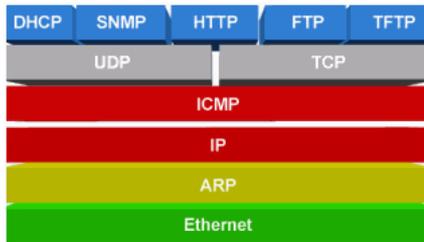
## 7.2.10 Energy Monitoring

This demo implements a power monitoring application that uploads data to Google PowerMeter. In this application, actual power consumption data is obtained from a PIC18F87J72 Energy Monitoring PICtail Plus Daughter Board. For more information about this demo application, consult the help file "Reference Implementation for Google PowerMeter Help.chm/pdf" in the Microchip Application Libraries help folder.

# 8 Using the Stack

This section describes how to use Microchip's TCP/IP Stack.

## 8.1 Stack Architecture



The TCP/IP stack is modular in design and written in the 'C' programming language. It follows the TCP/IP (Internet) protocol suite. The stack currently supports the TCP and UDP transport layer modules, the IPv4 (and part of the ICMP) Internet Layer modules, the ARP link layer modules, and a variety of application layer modules. Most of the Media Access Control link layer functionality is provided by the hardware MAC/PHY chips (see page 56) used with the stack.

## 8.2 How the Stack Works

This topic contains information about how the stack works, what is required to use the stack, and how your code can be structured to work cohesively with the TCP/IP stack.

### 8.2.1 Required Files

There are several base files that must be included in every project using Microchip's TCP/IP stack. They are:

- A main file (see page 132)- this is the file with your application code in it.
- `ARP.c` and `ARP.h`- These files are used by the stack to discover the MAC address associated with a given IP address.
- `Delay.c` and `Delay.h` – These files are used to provide delays for some stack functions. Note that it would be best to not use these delays in your own code, as they do create blocking conditions.
- Physical layer files – These files are used to enable a specified physical layer. More information on which files to include can be found in the Hardware Configuration (see page 136) section.
- `Helpers.c` and `Helpers.h` – These files contain helper functions used for miscellaneous stack tasks.
- `IP.c` and `IP.h` – These files provide internet layer functionality for the stack.
- `StackTsk.c` and `StackTsk.h` – These files contain the code to initialize the stack and perform the callbacks that keep the stack going.
- `Tick.c` and `Tick.h` – These files implement a tick timer that is used to implement some timing functionality within the stack.
- `HardwareProfile.h` – This configuration file is used to set up hardware options.

- `TCPIPConfig.h` – This configuration file is used to set up firmware options.
- `MAC.h` – This header file provides macros and structures relating to the hardware MAC layer.
- `TCPIP.h` – This is the primary include file for the stack. Your main file should include `TCPIP.h`.

You may choose to include additional files to support additional protocols and features. The list of protocols and their required files can be found in the Protocol Macros and Files (see page 144) topic in the Protocol Configuration (see page 143) topic.

---

## 8.2.2 APP\_CONFIG Structure

Most of the stack-related application variables are stored in an `APP_CONFIG` structure. These include addresses (see page 141), flags, and NBNS/SSID name strings. You will need to declare one of these structures (named "AppConfig") for your application and initialize it with the default values defined in `TCPIPConfig.h`. For example, you would set the bytes of the `MyIPAddr` field to the values of the `MY_DEFAULT_IP_ADDR_BYTEn` macros in `TCPIPConfig.h`. The `InitAppConfig` function in the file `MainDemo.c` of the `TCPIP Demo App` project demonstrates how to populate this structure completely. The full list of parameters in the `APP_CONFIG` structure is defined in the file `StackTsk.h`.

At the beginning of most stack demonstration applications, the code will check an EEPROM to determine if it contains a valid image of an `APP_CONFIG` structure. If so, it will read the image and use it to populate the `AppConfig` instance in the demo project. Otherwise, it will load the application variables from your statically defined values and/or configure them based on application protocols (DHCP/AutoIP). This allows a board to retain its configured settings even if the application loses power.

---

## 8.2.3 Main File

Because there is a huge variety of ways in which you could write your application, this section will provide an outline of what your main file should contain. It also provides some description of the stack operation, and of best-practice programming techniques to prevent stack problems.

### 8.2.3.1 Initialization

You should start by initializing your hardware. This includes PPS pins, oscillators, LEDs, LCDs, any SPI or PMP modules you're using to control your hardware MAC/PHY chip, etc.

Next, call the hardware initialization functions for the library. `TickInit` (see page 513)() should be called first; it will initialize the tick timer that manages your stack timing. Then call any additional initialization functions that require hardware initialization. For example, the `MPFSInit` (see page 278)() function will need to initialize an SPI port to communicate to a memory storage (see page 136) device to store web pages, so it should be called now.

Once your hardware is initialized, you can begin configuring your stack. Most of the stack-related application variables are stored in the `AppConfig` (see page 132) structure. At this point, you should initialize the `AppConfig` structure with your default values, or provide another means of initializing the `AppConfig` structure.

Finally, you can initialize the stack by calling the `StackInit`() function. This function will automatically call the initialization functions for other firmware protocols if they have been enabled in `TCPIPConfig.h` (i.e. `TCPInit` (see page 463)() for the TCP protocol, `HTTPInit` (see page 244)() for HTTP2,...). After `StackInit`() has been called, you can call other application-specific firmware initialization functions.

### 8.2.3.2 Main Loop

Once your program has been initialized, you should enter an infinite loop which will handle your application tasks. Within this

loop, there are two functions that you must call regularly: `StackTask` and `StackApplications`.

The `StackTask` function will perform any timed operations that the stack requires, and will handle transmission and reception of data packets. This function will also route any packets that have been received to the appropriate application protocol-level function to handle it.

The `StackApplications` function will call loaded application modules. For example, if an application is using an HTTP2 server, `StackApplications` will automatically call the `HTTPServer` (see page 245) function to process any HTTP2 tasks that are queued.

Most sub-tasks within `StackTask` and `StackApplications` are implemented as state-machine controlled cooperative multitasking functions. Since these sub-tasks consists of multiple steps (which may occur at varying times) this call-back system ensures that no single task will monopolize control of the processor.

Within this main loop, you may also want to poll for any I/O changes in your application and call any application specific tasks that you've implemented. To avoid causing buffer overflows in your hardware or protocol timing violations you should try to implement your own application tasks in callback functions with timing-based triggers. A method to do this is described in the next topic.

You must make one call to `StackTask` for each call of `StackApplications` but you aren't required to call these functions with any specific frequency. Calling `StackTask` too infrequently could limit your throughput, though, as each call of `StackTask` can retrieve one packet (at most) from the packet buffer. Similarly, application tasks that are time-dependant (like an ICMP ping response) may produce undesirable results if `StackApplications` is not called frequently enough.

The amount of time that the main loop takes to complete one iteration depends on several factors. If data is ready to be transmitted, or if a packet of received data was received, the `StackTask` function will take more time than it would otherwise. Each additional protocol included in your application will cause the main loop to take additional time as well, with the amount of time for each varying from the length of the shortest state machine state in the task to the longest.

Once your application is complete, you can set up a test case to determine the min/average/approximate maximum time that your loop will take to run. You can set your code up to use an internal timer to measure the duration of each iteration of the main loop, or you can set the code up to trigger an output pin each time the main loop completes, and use an oscilloscope to capture the network execution time. You can then provide application inputs or additional network traffic with a PC program (or other PICs) to simulate real-world operating conditions.

---

## 8.2.4 Cooperative Multitasking

If you implement the TCP/IP stack using a cooperative multitasking approach, you must make periodic calls to task functions to transmit/receive packets and to maintain protocol functionality. To prevent conflicts with the stack, you should write your own custom tasks in a way that will allow them to give up the processor if it's not needed. If you create a protocol or application task with multiple steps, it may be beneficial to divide them up between states. You can then use a global or static variable to track your state, and call that task function periodically to move through the state machine.

The following example contains a sample application for transferring data from a machine of some type to an external target. It includes a task function called `ApplicationTask` that has states to wait for button inputs, update the display, and transfer data from the machine. The functions in the example are used to represent other actions: `ButtonPressDetected` represents the code needed to check for an input from the user, `LCDDisplay` represents the code needed to update a display on the machine, `SampleData` gets data from the machine, `DataBufferIsFull` indicates that the buffer used to hold data samples needs to be sent, and `TransferData` is a function that writes the data to an open TCP or UDP socket. In between each of these states, the `ApplicationTask` function returns to the main loop, and the `StackTask` and `StackApplications` functions are called. This flow will allow the `StackApplications` function to maintain any module tasks. The `StackTask` function will periodically transmit the data from the socket buffers to its destination, which will prevent the transmit buffers from overflowing.

```
unsigned char gAppState;    // State tracking variable

int main (void)
{
```

```

// Pseudo-initialization function
InitializeCode();

// Setup application state
gAppState = STATE_DISPLAY_MENU;

// Main Loop
while (1)
{
    StackTask();
    StackApplications();
    ApplicationTask();
}
}

void ApplicationTask (void)
{
    switch (gAppState)
    {
        case STATE_DISPLAY_MENU:
            LCDDisplay (stringMainMenu);
            gAppState = STATE_MAIN_MENU;
            break;
        case STATE_MAIN_MENU:
            if (ButtonPressDetected (BUTTON_1) ) // Check an input
                gAppState = STATE_MONITOR_MACHINERY;
            break;
        case STATE_MONITOR_MACHINERY:
            LCDDisplay (stringTransferringData);
            // Generate or send data
            if (DataBufferIsFull())
            {
                TransferData();
            }
            else
            {
                SampleData();
            }
            if (ButtonPressDetected (BACK_BUTTON) )
                gAppState = STATE_DISPLAY_MENU;
            break;
    }
}
}

```

Some of the states in your application may be time based. Suppose, for example, that our sample application needs to send data for 5 seconds every time an input is detected. Stack problems could occur if the application used a delay loop to wait for 5 seconds until it was time to stop, so this functionality should be implemented using the stack's built-in tick timer. When the request to send data is received, the code will get the current tick time using the `TickGet` (see page 512) function, add enough ticks to make up 5 seconds, save it in a static variable called `tickCounter`, and then switch to a transmit state. Every time the `ApplicationTask` function gets called, it will enter this state in the state machine, call `TickGet` (see page 512) again, and then compare it to the value stored in that static variable. If the current time is later than the initial time plus the delay, the code will restore the display and re-enter the main menu state.

```

void ApplicationTask (void)
{
    static DWORD tickCounter;
    switch (gAppState)
    {
        case STATE_DISPLAY_MENU:
            LCDDisplay (stringMainMenu);
            gAppState = STATE_MAIN_MENU;
            break;
        case STATE_MAIN_MENU:
            if (ButtonPressDetected (BUTTON_1) ) // Check an input
                gAppState = STATE_MONITOR_MACHINERY;
            break;
        case STATE_MONITOR_MACHINERY:
            LCDDisplay (stringTransferringData);
            // Save the current time, and add 5 seconds to it
            tickCounter = TickGet() + (5 * TICK_SECOND);

```

```
        gAppState = STATE_CONTINUE_MONITORING;
        break;
    case STATE_CONTINUE_MONITORING:
        if ((long)(TickGet() - tickCounter) > 0)
            gAppState = STATE_DISPLAY_MENU;
        else
        {
            // Generate or send data
            if (DataBufferIsFull())
            {
                TransferData();
            }
            else
            {
                SampleData();
            }
        }
        break;
    }
}
```

There are three tick timing macros declared to help with delays: `TICK_SECOND` (see page 511) defines the number of ticks in a second, `TICK_MINUTE` (see page 511) defines the number of ticks in a minute, and `TICK_HOUR` (see page 511) defines the number of ticks in an hour. By using the tick timer to implement delays, you can ensure that your code won't block critical functions for too long.

---

## 8.2.5 RTOS

As an alternative to implementing your stack application in a cooperative multitasking format, you can integrate the stack into a Real-Time Operating System. For more information, see [Application Note 1264](#) on the Microchip web site.

# 9 Configuring the Stack

There is a wide range of configuration options available for Microchip's TCP/IP Stack. This topic will discuss the functionality of these options, and how to implement them.

---

## 9.1 Hardware Configuration

Most hardware configuration is performed by commenting, uncommenting, or defining a series of macros in the one of the variants of the header file `HardwareProfile.h`. You can see sample versions of how to set these options in the copies of `HardwareProfile.h` that are included with the stack's demo projects (see page 77).

In most cases, the macro to enable a device is the same macro used to define the device's chip select pin. In the default copies of `HardwareProfile.h` included with the demonstration projects, there are example sections defined for most demo boards, delimited by preprocessor statements. For example, the section for the Explorer 16 begins with the macro `"#elif defined (EXPLORER_16)"` and continues until the next demo board preprocessor statement. If you use one of these files as a base for your project, make sure you are modifying the macros in the correct section.

---

### 9.1.1 Clock Frequency

Many TCP/IP operations are time-dependant. By specifying the oscillator frequency that you're using in your application for the stack, you can enable automatic handling of these operations. To set the clock value, substitute your oscillator frequency (in Hertz) for the value in the following macro in `HardwareProfile.h`:

```
#define GetSystemClock()      xxxxxxxxxxxxxxxxx
```

There are also two other clock macros. `GetInstructionClock()` and `GetPeripheralClock()` provide frequency values for the instruction clock and peripheral clock in your microcontroller. These values will usually be set as a fraction of the system clock (i.e. `GetInstructionClock()` would be defined as `(GetSystemClock() / 2)` for PIC24F processors).

---

### 9.1.2 External Storage

There are several features in the TCP/IP stack that use external storage to maintain structures or web pages. Support for a few storage devices is included with the stack; the support files can be used as a template to write drivers for other devices as well. The `HardwareProfile.h` pin definitions are roughly equivalent for each storage device, except for the first word of the macro, which indicates which type of storage device it applies to (e.g. `EEPROM_CS_IO` vs `SPIFLASH_CS_IO`). There are three different storage media.

#### EEPROM

A EEPROM can be used to store MPFS2 (see page 265) web page images and custom application structures. To indicate to the stack that it should use a EEPROM to store MPFS2 images, define the macro `MPFS_USE_EEPROM` in the `TCPIPConfig.h` header file. By default, the stack includes a driver for Microchip's 25LC256 EEPROM family (to use the 1 Mbit EEPROM, you must also define the macro `USE_EEPROM_25LC1024` in `TCPIPConfig.h`). The macros to control communication with the EEPROM will be prepended with the string `EEPROM_` in this case. To enable communication, define `EEPROM_CS_TRIS` and include the files `SPIEEPROM.c` and `XEEPROM.h` in your application. These files may require

some changes to support additional EEPROM devices.

### Serial Flash

Storage for MPFS images and custom structures is also available on serial flash devices (tested with SST 25VF016B and Spansion 25FL040A). To indicate that the stack should use serial flash to store web pages, define `MPFS_USE_SPI_FLASH` in `TCPIPConfig.h`. The communication macros will be prepended with the string `SPIFLASH_` in this case. To enable communication functionality, define `SPIFLASH_CS_TRIS` and include the files `SPIFlash.c` and `SPIFlash.h` in your application. These files may require some changes to support additional flash devices. There are several macros included within “`SPIFlash.h`” that must also be defined, including macros to define the sector and page sizes, and macros to describe whether the SST or Spansion flash device is being used.

### SRAM

A serial RAM can be used to store FIFO blocks and TCP Control Blocks for sockets (see page 146) (tested with AMT Semiconductor’s N256S0830HDA). The macros will be prepended with the string `SPIRAM_` in this case. To use this functionality, define `EEPROM_CS_TRIS` and include the files “`SPIRAM.c`” and “`SPIRAM.h`” in your application. These files may require some changes to support additional RAM devices.

Macro	Purpose	Sample Value
<code>xxxxxx_CS_IO</code>	Defines the LAT (or PORT, where applicable) register bit that corresponds to the chip select pin. Defining this macro will indicate that the stack should use the specified type of external storage.	<code>LATDbits.LATD12</code>
<code>xxxxxx_CS_TRIS</code>	Defines the TRIS bit that corresponds to the chip select pin on the device.	<code>TRISDbits.TRISD12</code>
<code>xxxxxx_SCK_TRIS</code>	Defines the TRIS bit that corresponds to the clock pin of the SPI module connected to the device.	<code>TRISGbits.TRISG6</code>
<code>xxxxxx_SDI_TRIS</code>	Defines the TRIS bit that corresponds to the data-in pin of the SPI module connected to the device.	<code>TRISGbits.TRISG7</code>
<code>xxxxxx_SDO_TRIS</code>	Defines the TRIS bit that corresponds to the data-out pin of the SPI module connected to the device.	<code>TRISGbits.TRISG8</code>
<code>xxxxxx_SPI_IF</code>	Points to the interrupt flag for the SPI module connected to the device.	<code>IFS2bits.SPI2IF</code>
<code>xxxxxx_SSPBUF</code>	Points to the SPI buffer register for the SPI module connected to the device.	<code>SPI2BUF</code>
<code>xxxxxx_SPICON1</code>	Points to the SPI control register for the SPI module connected to the device.	<code>SPI2CON1</code>
<code>xxxxxx_SPICON1bits</code>	Provides bitwise access to the SPI control register for the SPI module connected to the device. The <code>____bits</code> registers are typically defined in the processor’s header files.	<code>SPI2CON1bits</code>
<code>xxxxxx_SPICON2</code>	Points to the second SPI control register for the SPI module connected to the device. If your device doesn’t have an <code>SPICON2</code> register (e.g. PIC32) just omit this definition.	<code>SPI2CON2</code>
<code>xxxxxx_SPISTAT</code>	Points to the SPI status register for the SPI module connected to the device.	<code>SPI2STAT</code>
<code>xxxxxx_SPISTATbits</code>	Provides bitwise access to the SPI status register for the SPI module connected to the device.	<code>SPI2STATbits</code>
<code>xxxxxx_SPIBRG</code>	Points to the SPI Baud Rate Generator register for the SPI module connected to the device. If your device doesn’t have a BRG-based SPI module, just omit this definition.	<code>SPI2BRG</code>

## 9.1.3 ENC28J60 Config

To use the ENC28J60 in your project, include the files ‘`ENC28J60.c`’ and “`ENC28J60.h`” in your project and uncomment the following macro in `HardwareProfile.h`:

```
#define ENC_CS_TRIS xxxxxxxxxxxxxxxxxx
```

Several macros need to be mapped to registers or register bits when using the ENC28J60. They include:

Macro	Purpose	Sample Value
ENC_CS_IO	Defines the LAT (or PORT, where applicable) register bit that corresponds to the chip select pin. Defining this macro will also indicate that the stack should use the ENC28J60.	LATDbits.LATD14
ENC_CS_TRIS	Defines the TRIS bit that corresponds to the chip select pin.	TRISDbits.TRISD14
ENC_RST_IO	Defines the LAT (or PORT, where applicable) register bit that corresponds to the reset pin. If you leave the reset pin unconnected in your design, comment this macro out.	LATDbits.LATD15
ENC_RST_TRIS	Defines the TRIS bit that corresponds to the reset pin.	TRISDbits.TRISD15
ENC_SPI_IF	Points to the interrupt flag for the SPI module connected to the chip.	IFS0bits.SPI1IF
ENC_SSPBUF	Points to the SPI buffer register for the SPI module connected to the chip.	SPI1BUF
ENC_SPISTAT	Points to the SPI status register for the SPI module connected to the chip.	SPI1STAT
ENC_SPISTATbits	Provides bitwise access to the SPI status register for the SPI module connected to the chip. The ____bits registers are typically defined in the processor's header files.	SPI1STATbits
ENC_SPICON1	Points to the SPI control register for the SPI module connected to the chip.	SPI1CON1
ENC_SPICON1bits	Provides bitwise access to the SPI control register for the SPI module connected to the chip (see ENC_SPISTATbits entry).	SPI1CON1bits
ENC_SPICON2	Points to the second SPI control register for the SPI module connected to the chip. If your device doesn't have an SPICON2 register (e.g. PIC32) just omit this definition.	SPI1CON2
ENC_SPIBRG	Points to the SPI Baud Rate Generator register for the SPI module connected to the chip. If your device doesn't have a BRG-based SPI module, just omit this definition.	SPI1BRG

## 9.1.4 ENCX24J600 Config

To use the ENC624J600 or -424J600 in your project, include "ENCX24J600.c" and "ENCX24J600.h" and uncomment the following macro in `HardwareProfile.h`:

```
#define ENC100_INTERFACE_MODE 0
```

The parameter '0' indicates that you'll be using the device in SPI mode. Potential usable parameters include:

Parameter	Functionality
0	SPI mode using CS, SCK, SI, and SO pins
1	8-bit demultiplexed PSP Mode 1 with RD and WR pins
2	8-bit demultiplexed PSP Mode 2 with R/~W and EN pins
3	16-bit demultiplexed PSP Mode 3 with Rd, WRL, and WRH pins
4	16-bit demultiplexed PSP Mode 4 with R/~W, B0SEL, and B1SEL pins
5	8-bit multiplexed PSP Mode 5 with RD and WR pins
6	8-bit multiplexed PSP Mode 6 with R/~W and EN pins
9	16-bit multiplexed PSP Mode 9 with AL, RD, WRL, and WRH pins
10	16-bit multiplexed PSP Mode 10 with AL, R/~W, B0SEL, and B1SEL pins

More information on the functionality of each mode is available in the ENC624J600 family datasheet. Note, however, that the

44-pin ENC424J600 only supports communication using the SPI mode and PSP Modes 5 and 6. Also, because of board conflicts, PSP Modes 2, 4, 6, and 10 shouldn't be used with the Explorer 16 (and PSP Mode 3 may cause bus contention with the 25LC256 EEPROM).

Several macros need to be mapped to registers or register bits when using the ENCX24J600 as well. In addition, some features can be enabled/disabled for this device by defining certain macros. Macros include:

Macro	Purpose	Sample Value
ENC100_INTERFACE_MODE	Indicates which communication mode the stack should use to interface to the chip. This macro will also indicate that the stack should use the ENCX24J600.	0
ENC100_PSP_USE_INDIRECT_RAM_ADDRESSING	Un-commenting this macro will allow the stack to indirectly address the RAM of the ENCX24J600 (to save some address wires). For SPI mode or PSP Modes 9 and 10, this option will be ignored.	N/A
ENC100_TRANSLATE_TO_PIN_ADDR(a)	This macro will actually remap the addresses passed into the parallel interface to fit the configuration of the pins (if you are using indirect addressing).	$((a \& 0x100) \ll 6)   (a \& 0x00FF)$
ENC100_MDIX_IO	If you design an Auto-crossover (Auto-MDIX) circuit into your board, this macro will define the pin to use for it. See the Fast 100Mbps Ethernet PICtail/PICtail Plus board schematic for an example circuit.	LATBbits.LATB3
ENC100_MDIX_TRIS	Defines the TRIS bit to use for the Auto-crossover circuit.	TRISBbits.TRISB3
ENC100_INT_IO	Defines an I/O pin to use for the chip's interrupt signal pin. This feature is currently unused by the stack.	PORTAbits.RA13
ENC100_INT_TRIS	Defines a TRIS bit to use for the chip's interrupt signal pin.	TRISAbits.TRISA13
ENC100_CS_IO	Defines a port bit for use with the chip select pin. Optional in PSP modes.	LATAbits.LATA5
ENC100_CS_TRIS	Defines a TRIS bit to use for the chip select pin.	TRISAbits.TRISA5
ENC100_POR_IO	Defines the port bit to use with a power disconnect circuit. If your application doesn't have this feature implemented, comment out this bit.	LATCbits.LATC1
ENC100_POR_TRIS	Defines the TRIS bit to use with a power disconnect circuit.	TRISCbits.TRISC1
ENC100_SO_WR_B0SEL_EN_IO	Defines a pin used for communication. The functionality of this pin depends on which communication mode is selected. It can be equivalent to the ENCX24J600 serial out pin, the parallel mode WR strobe, the B0SEL pin, or the EN pin.	LATDbits.LATD4
ENC100_SO_WR_B0SEL_EN_TRIS	Defines the TRIS bit to use with the ENC100_SO_WR_B0SEL_EN_IO pin.	TRISDbits.TRISD4
ENC100_SI_RD_RW_IO	Defines a pin used for communication. The functionality of this pin depends on which communication mode is selected. It can be equivalent to the ENCX24J600 serial in pin, the parallel mode RD strobe, or the R/~W pin.	LATDbits.LATD5
ENC100_SI_RD_RW_TRIS	Defines the TRIS bit to use with the ENC100_SI_RD_RW_IO pin.	TRISDbits.TRISD5
ENC100_SCK_AL_IO	Defines a pin used for communication. The functionality of this pin depends on which communication mode is selected. It can be equivalent to the ENCX24J600 serial clock pin or the parallel mode address latch strobe.	LATDbits.LATDB15
ENC100_SCK_AL_TRIS	Defines the TRIS bit to use with the ENC100_SCK_AL_IO pin.	TRISDbits.TRISD15
ENC100_ISR_ENABLE	Points to the bit to enable the interrupt for the I/O based ENCX24J600-triggered interrupt. This feature is not currently implemented.	IEC1bits.INT2IE

ENC100_ISR_FLAG	Points to the interrupt flag bit for the I/O based ENCX24J600-triggered interrupt. This feature is not currently implemented.	IFS1bits.INT2IF
ENC100_ISR_POLARITY	Points to the interrupt polarity bit for the I/O based ENCX24J600-triggered interrupt. This feature is not currently implemented.	INTCON2bits.INT2EP
ENC100_ISR_PRIORITY	Points to the interrupt priority bit for the I/O based ENCX24J600-triggered interrupt. This feature is not currently implemented.	IPC7bits.INT2IP
ENC100_SPI_ENABLE	Points to the SPI module enable bit if SPI mode is used.	SPI1STATbits.SPIEN
ENC100_SPI_IF	Points to the interrupt flag for the SPI module if SPI mode is used.	IFS0bits.SPI1IF
ENC100_SSPBUF	Points to the SPI buffer register for the SPI module if SPI mode is used.	SPI1BUF
ENC100_SPISTAT	Points to the SPI status register for the SPI module if SPI mode is used.	SPI1STAT
ENC100_SPISTATbits	Provides bitwise access to the SPI status register for the SPI if SPI mode is used. The ____bits registers are typically defined in the processor's header files.	SPI1STATbits
ENC100_SPICON1	Points to the SPI control register for the SPI module if SPI mode is used.	SPI1CON1
ENC100_SPICON1bits	Provides bitwise access to the SPI control register for the SPI module if SPI mode is used (see ENC_SPISTATbits entry).	SPI1CON1bits
ENC100_SPICON2	Points to the second SPI control register for the SPI module if SPI mode is used. If your device doesn't have an SPICON2 register (e.g. PIC32) just omit this definition.	SPI1CON2
ENC100_SPIBRG	Points to the SPI Baud Rate Generator register for the SPI module if SPI mode is used. If your device doesn't have a BRG-based SPI module, just omit this definition.	SPI1BRG

## 9.1.5 PIC18F97J60 Config

The 18F97J60 can be used in your application by selecting it as the processor in MPLAB, ensuring that the ENC\_CS\_TRIS macro is commented out, and including the files "ETH97J60.c" and "ETH97J60.h." There are no additional macros to define for the 97J60; since it uses its own internal MAC and PHY for communication all of the register names and bit names are fixed.

## 9.1.6 PIC32MX7XX Config

To use the PIC32MX795 in your project, include the files `ETHPIC32IntMac.c` and `ETHPIC32ExtPhy.c` in your project. You'll also have to add a specific PHY implementation file (by default `ETHPIC32ExtPhyDP83848.c` is provided) depending on your actual external PHY selection.

Update the following definitions in `HardwareProfile.h`:

Macro	Purpose	Sample Value
PHY_RMII	Define this macro if the external PHY runs in RMII mode. Comment it out if you're using an MII PHY.	-
PHY_CONFIG_ALTERNATE	Define this symbol if the PIC32MX7XX uses the alternate configuration pins to connect (see page 165) to the PHY. Comment it out for the default configuration pins.	-
PHY_ADDRESS	Update with the MIIM address of the external PHY you are using (the address on which the PHY responds to MIIM transactions. See the PHY datasheet).	0x1

Update the following definitions in `TCPIPConfig.h`:

Macro	Purpose	Sample Value
ETH_CFG_LINK	Set to 0 to use the default connection characteristics (depends on the selected PHY). Set to 1 to configure the Ethernet link to the following specific parameters. Auto-negotiation will always be enabled if supported by the PHY.	0
ETH_CFG_AUTO	Set to 1 to use auto negotiation. Strongly recommended.	1
ETH_CFG_10	Use/advertise 10 Mbps capability.	1
ETH_CFG_100	Use/advertise 100 Mbps capability.	1
ETH_CFG_HDUPLEX	Use/advertise half duplex capability.	1
ETH_CFG_FDUPLEX	Use/advertise full duplex capability.	1
ETH_CFG_AUTO_MDIX	Use/advertise auto MDIX capability (effective only when ETH_CFG_AUTO is enabled).	1
ETH_CFG_SWAP_MDIX	Use swapped MDIX if defined. Otherwise, use normal MDIX.	1

## 9.2 Address

A TCP/IP application will need to have both a Media Access Control (MAC) address and an Internet Protocol (IP) address. There are multiple methods for obtaining or setting these addresses.

### 9.2.1 MAC Address

The 6-byte MAC address provides addressing for the Media Access Control protocol layer of the TCP/IP stack. MAC addresses are permanent addresses tied to hardware. Blocks of MAC addresses are sold to organizations and individuals by the IEEE; if you aren't using a Microchip device with a built-in MAC address, you will need to purchase one of these blocks to assign MAC addresses to your products.

The MAC address is defined in the firmware configuration header "TCPIPConfig.h." There are six macros that must be defined in this file to set the MAC address. They are:

Macro	Sample Value
MY_DEFAULT_MAC_BYTE1	(0x00)
MY_DEFAULT_MAC_BYTE2	(0x04)
MY_DEFAULT_MAC_BYTE3	(0xA3)
MY_DEFAULT_MAC_BYTE4	(0x00)

MY_DEFAULT_MAC_BYTE5	(0x00)
MY_DEFAULT_MAC_BYTE6	(0x00)

Each of these macros represents a byte of the MAC address (note that 00:04:A3:xx:xx:xx is the block of MAC addresses reserved for Microchip products). Once you obtain your block of addresses, you will need to specify a unique address for every device you produce. The "TCP/IP Demo App" demonstration project describes a method for using Microchip's MPLAB PM3 programmer to serially program a range of MAC addresses into multiple parts without recompiling your project.

The ENCX24J600, MRF24WB0M, and PIC32MX7XX/6XX feature a pre-programmed MAC address (from Microchip's address block). If you are using either of these part families in your project, you can define your MAC address as "00:04:A3:00:00:00" and the stack will automatically use the part's pre-programmed address for your application.

Microchip also provides a [family of EEPROMs](#) that include a unique, pre-programmed EUI-48 (MAC) or EUI-64 address. When using one of these devices, you can write your AppConfig (see page 132) initialization code so it will obtain the device's MAC address from one of these EEPROMs instead of the default MAC address macros.

## 9.2.2 IP Address

The IP address is used to address nodes on an Internet Protocol network. You will need to configure your application with an IP address, or enable a method to obtain one. You may also want to define a few other parameters that describe how your device will try to fit into its network, by default.

The macros that you will need to define include:

Macro	Property	Sample Value
MY_DEFAULT_IP_ADDR_BYTE1	Default IP address byte 1	192ul
MY_DEFAULT_IP_ADDR_BYTE2	Default IP address byte 2	168ul
MY_DEFAULT_IP_ADDR_BYTE3	Default IP address byte 3	1ul
MY_DEFAULT_IP_ADDR_BYTE4	Default IP address byte 4	100ul
MY_DEFAULT_MASK_BYTE1	Default subnet mask byte 1	255ul
MY_DEFAULT_MASK_BYTE2	Default subnet mask byte 2	255ul
MY_DEFAULT_MASK_BYTE3	Default subnet mask byte 3	255ul
MY_DEFAULT_MASK_BYTE4	Default subnet mask byte 4	0ul
MY_DEFAULT_GATE_BYTE1	Default gateway byte 1	192ul
MY_DEFAULT_GATE_BYTE2	Default gateway byte 2	168ul
MY_DEFAULT_GATE_BYTE3	Default gateway byte 3	1ul
MY_DEFAULT_GATE_BYTE4	Default gateway byte 4	1ul

The subnet address is a bit mask that defines the scope of the network. If your IP address is 192.168.5.100, and you specify a subnet mask of 255.255.255.0, the stack will assume that addresses in the range 192.168.5.x are on the same subnet that you are, and that packets sent to any of those addresses won't have to be routed anywhere else.

The default gateway is the IP address of the node on the network that your application will send packets to if it doesn't know how to route them to the address it wants to send them to. If your application is on the 192.268.5.x subnet, if it wants to send a packet to 198.175.253.160 and it doesn't know exactly how to get there, it will send it to the default gateway.

Note that if you write your own code instead of starting with a demo application, you will need to populate your AppConfig (see page 132) structure with these values. Also note that these are only default values. Other protocols (or your application itself) may modify any of the APP\_CONFIG fields that represent these parameters.

There are three methods that you can use to set or obtain an IP address: static, DHCP, or AutoIP.

### Static IP Addressing

Using a static address will allow you to specify a set IP address. This can either be done at compile time, by setting the default IP address to the value you'd like to use and using the demo code (which populated your AppConfig structure automatically), or during run-time, by programming your application to set the IP address in your AppConfig structure based on some input. If you'd like to include the code for DHCP and AutoIP address acquisition if your project but still use static addressing, you can call the DHCP and AutoIP functions that disable those modules to prevent them from overwriting your IP address. Use of static addresses will usually only work if the server is configured to support that address.

### DHCP

The DHCP client module will allow your application to dynamically obtain an IP address from a DHCP server on the same network. Doing this will reset the IP address, subnet mask, gateway address, and some other configuration parameters in your AppConfig structure. To use DHCP, include the files `DHCP.c`, `DHCPs.c`, and `DHCP.h` in your project, and add or uncomment the definition `#define STACK_USE_DHCP_CLIENT` to `TCPIPConfig.h`. The TCP/IP stack also includes a simple DHCP server that can supply an IP address to one DHCP client. To enable this functionality, add the macro `#define STACK_USE_DHCP_SERVER` to `TCPIPConfig.h`.

### AutoIP

The AutoIP module will allow your application to choose an IP address and claim it for itself. These addresses are link-local, meaning they cannot be routed, and will only work on your local link. This functionality is based on the specification for allocating dynamic link-local addresses, but is modified to take the form used by Microsoft's APIPA link-local address allocation scheme. To enable this feature, include the files `AutoIP.c` and `AutoIP.h` and add the macro `#define STACK_USE_AUTO_IP` to `TCPIPConfig.h`.

### IP Address (see page 141) Module Interaction

It is possible to configure a default static address and enable DHCP and AutoIP at the same time. If you don't disable one or the other, the AutoIP module will immediately choose an address in the specified address range and begin attempting to claim it. DHCP will also begin sending messages to attempt to lease a DHCP IP address from a DHCP server. In most cases the DHCP module will complete all of its transactions before AutoIP finishes claiming its address. In this case, the DHCP address will be copied to the AppConfig structure and the AutoIP module will stop trying to claim its address. Since a routable DHCP address is always preferred to a link-local AutoIP address, the stack will also immediately start using a DHCP address if it becomes available, even if an AutoIP address was already in use (i.e. if you enable DHCP after AutoIP has already claimed an address). This may cause existing open sockets to lose communication; they should be re-opened with the new address. In this situation, you can also use a static address if you disable DHCP and AutoIP and set the static address in the AppConfig structure.

If AutoIP is used in conjunction with the DHCP Server module, the AutoIP module will generate an address in the 169.254.x.x subnet and then serve another address in the same subnet to the DHCP client connected to the board.

---

## 9.3 Protocol Configuration

There are a few steps that you must take to include each protocol in your application. Most of this configuration is performed by setting options in one of the variants of the `TCPIPConfig.h` header file. Nearly all protocols will require you to enable them by defining one or more macros in `TCPIPConfig.h`. You will also need to include the files needed by your protocols in your project. Some protocols will require you to define sockets for them to use in `TCPIPConfig.h`, and allocate memory to them.

The TCP/IP Configuration Wizard, included with the stack, will allow you to select the features that you want while handling most complex firmware configuration automatically. Because of this, it is the easiest (and safest) way to set up your application protocols.

The Module APIs topic has a description of each of the modules.

## 9.3.1 Protocol Macros and Files

You will need to define some macros in TCPIPConfig.h and include some files in your project to enable each protocol. These include:

Module	Macro	Function	Required Files
ICMP (see page 257)	STACK_USE_ICMP_SERVER	Provides the ability to query and respond to pings.	ICMP.c, ICMP.h
ICMP (see page 257)	STACK_USE_ICMP_CLIENT	Provides the ability to transmit pings.	ICMP.c, ICMP.h
HTTP2 (see page 224)	STACK_USE_HTTP2_SERVER	Provides HTTP server functionality with dynamic variables, POST, Cookies (see page 86), Authentication (see page 84), and other features	HTTP2.c, HTTP2.h, TCP.c, TCP.h, CustomHTTPApp.c and HTTPPrint.h (see HTTP2 (see page 224) section for information on these files)
SSL (see page 373)	STACK_USE_SSL_SERVER	Provides support for SSL server sockets.	SSL.c, SSL.h, ARCFOUR.c, ARCFOUR.h, BigInt.c, BigInt.h, Random.c, Random.h, RSA.c, RSA.h
SSL (see page 373)	STACK_USE_SSL_CLIENT	Provides support for SSL client sockets.	SSL.c, SSL.h, ARCFOUR.c, ARCFOUR.h, BigInt.c, BigInt.h, Random.c, Random.h, RSA.c, RSA.h
FTP	STACK_USE_FTP_SERVER	Provides ability to remotely upload MPFS2 images to HTTP2 servers via FTP	FTP.c, FTP.h, TCP.c, TCP.h
SMTP (see page 291)	STACK_USE_SMTP_CLIENT	Provides the ability to send email	SMTP.c, SMTP.h, TCP.c, TCP.h, Helpers.c, Helpers.h
SNMP (see page 312)	STACK_USE_SNMP_SERVER	Provides a network-based machine control/monitoring protocol	SNMP.c, SNMP.h, UDP.c, UDP.h
TFTP (see page 485)	STACK_USE_TFTP_CLIENT	Provides unreliable file upload/download services	TFTPC.c, TFTPC.h, TCP.c, TCP.h
Telnet (see page 481)	STACK_USE_TELNET_SERVER	Provides telnet services.	Telnet.c, Telnet.h, TCP.c, TCP.h
Announce (see page 149)	STACK_USE_ANNOUNCE	Provides device hostname and IP address discovery on a local Ethernet subnet	Announce.c, Announce.h

DNS (see page 178)	STACK_USE_DNS	Provides the ability to resolve hostnames to IP addresses	DNS.c, DNS.h, UDP.c, UDP.h
NBNS (see page 284)	STACK_USE_NBNS	Provides the ability to resolve hostnames to IP addresses on the same subnet.	NBNS.c, NBNS.h, UDP.c, UDP.h
SNTP (see page 368)	STACK_USE_SNTP_CLIENT	Provides the ability to get the date/time from the internet	SNTP.c, SNTP.h, UDP.c, UDP.h
Dynamic DNS (see page 187)	STACK_USE_DYNAMICDNS_CLIENT	Provides the ability to resolve hostnames to IP addresses that change frequently.	DynDNS.c, DynDNS.h, TCP.c, TCP.h
MPFS2 (see page 265)	STACK_USE_MPFS2	Provides MPFS2 services for custom applications. This functionality will be enabled/required automatically by stack-based protocols that require MPFS2.	MPFS2.c, MPFS2.h
TCP (see page 434)	STACK_USE_TCP	Provides TCP transport layer services for custom protocols. This functionality is automatically enabled/required by stack-based protocols that require TCP sockets.	TCP.c, TCP.h
UDP (see page 516)	STACK_USE_UDP	Provides UDP transport layer services for custom protocols. This functionality is automatically enabled/required by stack-based protocols that require UDP sockets.	UDP.c, UDP.h

## 9.3.2 Additional Features

The TCP/IP stack includes some additional functionality that can be enabled in `TCPIPConfig.h`.

Feature	Macro	Description	Required Files
UART Demo	STACK_USE_UART	Application demo using UART for IP address display and stack configuration.	UART.c, UART.h
UART-to-TCP Bridge	STACK_USE_UART2TCP_BRIDGE	UART to TCP Bridge application example	UART2TCPBridge.c, UART2TCPBridge.h
IP Gleaning	STACK_USE_IP_GLEANING	Allows assignment of an IP address via reception of an ICMP packet with a valid IP during configuration mode	-
Reboot Server (see page 311)	STACK_USE_REBOOT_SERVER	Allows the PIC to be reset remotely (useful for bootloaders).	Reboot.c, Reboot.h
UDP Performance Test (see page 287)	STACK_USE_UDP_PERFORMANCE_TEST	UDP performance test. Monitor a local area network for UDP packets with a packet sniffer. This test will transmit 1024 packets. Use the timestamps of the first and last packets to calculate throughput.	UDPPerformanceTest.c, UDPPerformanceTest.h

TCP Performance Test (see page 287)	STACK_USE_TCP_PERFORMANCE_TEST	TCP performance test. Connect a demo board to a PC via UART, execute the code, and monitor the throughput on the PC terminal.	TCPPerformanceTest.c, TCPPerformanceTest.h
Berkeley API (see page 161)	STACK_USE_BERKELEY_API	Provides a Berkeley Sockets (see page 146) API abstraction layer.	BerkeleyAPI.c, BerkeleyAPI.h

## 9.3.3 Sockets

Most of your application protocols will require you to allocate memory for each connection (socket) that you have open. Like the other firmware configuration options, this is controlled by the definition of macros in `TCPIPConfig.h`. For TCP sockets, you will have to specify four initialization parameters for each socket, including the purpose of that socket, the type of memory the socket should be stored in, the size of the transmit FIFO, and the size of the receive FIFO. The stack will then initialize the sockets with this information, and create a TCP Control Block (TCB) for each to control its operations. This topic will outline the socket configuration functionality in the sample version of `TCPIPConfig.h` that is included with the TCP/IP Demo App project.

### 9.3.3.1 Memory Allocation

```
#define TCP_ETH_RAM_SIZE (3900ul)
#define TCP_PIC_RAM_SIZE (0ul)
#define TCP_SPI_RAM_SIZE (0ul)
#define TCP_SPI_RAM_BASE_ADDRESS (0ul)
```

The first four macros in the socket section are used to describe the total amount of memory used to contain sockets. When data is sent from a TCP socket, it will first be copied into the socket's transmit FIFO, and then to the MAC/PHY transmit buffer. Similarly, received data will be read from the MAC/PHY chip into the receive FIFO. These FIFOs, as well as the TCB, can be stored in 3 places.

`TCP_ETH_RAM_SIZE` is used to define the RAM available for sockets on the actual TCP/IP MAC/PHY chip. This will not be the same as the total RAM on the chip; some memory must be reserved for packets being transmitted and received. By default ~1518 bytes (the maximum single-packet transmission size) will be reserved for TX packets on Microchip parts. The amount reserved for the receive packet buffer will equal the amount remaining after allocating the memory for the TX buffer and the memory for the sockets. You may receive a compile-time warning if the RX buffer is unreasonably small.

`TCP_PIC_RAM_SIZE` is used to define the RAM available for sockets on the PIC microcontroller that's driving your application.

`TCP_SPI_RAM_SIZE` defines the RAM available for sockets on an external SPI RAM (see External Storage (see page 136)). You can specify the base address in this RAM chip to use with the `TCP_SPI_RAM_BASE_ADDRESS` macro.

### 9.3.3.2 Socket Types

When creating an initialization list for your sockets, you will have to specify a socket type. This parameter will define which protocol can use the socket. You can create and delete socket types as you require. In the sample version of `TCPIPConfig.h`, the following types are defined:

```
#define TCP_SOCKET_TYPES
#define TCP_PURPOSE_GENERIC_TCP_CLIENT 0
#define TCP_PURPOSE_GENERIC_TCP_SERVER 1
#define TCP_PURPOSE_TELNET 2
```

```

#define TCP_PURPOSE_FTP_COMMAND 3
#define TCP_PURPOSE_FTP_DATA 4
#define TCP_PURPOSE_TCP_PERFORMANCE_TX 5
#define TCP_PURPOSE_TCP_PERFORMANCE_RX 6
#define TCP_PURPOSE_UART_2_TCP_BRIDGE 7
#define TCP_PURPOSE_HTTP_SERVER 8
#define TCP_PURPOSE_DEFAULT 9
#define TCP_PURPOSE_BERKELEY_SERVER 10
#define TCP_PURPOSE_BERKELEY_CLIENT 11
#define END_OF_TCP_SOCKET_TYPES

```

The `TCP_PURPOSE_GENERIC_TCP_CLIENT` and `TCP_PURPOSE_GENERIC_TCP_SERVER` socket types are used by the generic TCP client and server examples (see `GenericTCPClient.c` and `GenericTCPServer.c`). These files are used as an example of how to create a new, custom TCP client or server application.

If you are trying to open a Telnet (see page 481) connection, the stack will try to use a `TCP_PURPOSE_TELNET` socket.

The `TCP_PURPOSE_FTP_COMMAND` and `TCP_PURPOSE_FTP_DATA` socket types are used to receive FTP commands and data.

The two `TCP_PERFORMANCE_x` socket types are used solely to conduct TCP performance testing.

The `TCP_PURPOSE_UART_2_TCP_BRIDGE` socket type is used for the UART-to-TCP bridge example.

The `TCP_PURPOSE_HTTP_SERVER` socket type is used for sockets on HTTP servers that listen (see page 169) for web page view requests.

The `TCP_PURPOSE_DEFAULT` socket type can be used for miscellaneous applications, or for applications that only need sockets temporarily. Dynamic DNS connections and SMTP connections use default sockets, and the legacy wrapper implementation for the `TCPListen` (see page 451) and `TCPConnect` (see page 441) functions try to open them.

The `TCP_PURPOSE_BERKELEY_SERVER` and `TCP_PURPOSE_BERKELEY_CLIENT` socket types indicate that a socket is available for the use of the Berkeley API (see page 161) layer (also see BSD Sockets (see page 148)).

### 9.3.3.3 Initialization Structure

In the `TCPIPConfig.h` header file, you must also define an array of structures to declare and initialize any sockets that you need. The sample structure is:

```

#define TCP_CONFIGURATION ROM struct {
BYTE vSocketPurpose, BYTE vMemoryMedium, WORD wTXBufferSize, WORD wRXBufferSize }
TCPSocketInitializer[] =
{
    {TCP_PURPOSE_GENERIC_TCP_CLIENT, TCP_ETH_RAM, 125, 100},
    {TCP_PURPOSE_GENERIC_TCP_SERVER, TCP_ETH_RAM, 20, 20},
    {TCP_PURPOSE_TELNET, TCP_ETH_RAM, 200, 150},
    // {TCP_PURPOSE_TELNET, TCP_ETH_RAM, 200, 150},
    // {TCP_PURPOSE_TELNET, TCP_ETH_RAM, 200, 150},
    // {TCP_PURPOSE_FTP_COMMAND, TCP_ETH_RAM, 100, 40},
    // {TCP_PURPOSE_FTP_DATA, TCP_ETH_RAM, 0, 128},
    {TCP_PURPOSE_TCP_PERFORMANCE_TX, TCP_ETH_RAM, 200, 1},
    // {TCP_PURPOSE_TCP_PERFORMANCE_RX, TCP_ETH_RAM, 40, 1500},
    {TCP_PURPOSE_UART_2_TCP_BRIDGE, TCP_ETH_RAM, 256, 256},
    {TCP_PURPOSE_HTTP_SERVER, TCP_ETH_RAM, 200, 200},
    {TCP_PURPOSE_HTTP_SERVER, TCP_ETH_RAM, 200, 200},
    {TCP_PURPOSE_DEFAULT, TCP_ETH_RAM, 200, 200},
    {TCP_PURPOSE_BERKELEY_SERVER, TCP_ETH_RAM, 25, 20},
    // {TCP_PURPOSE_BERKELEY_SERVER, TCP_ETH_RAM, 25, 20},
    // {TCP_PURPOSE_BERKELEY_SERVER, TCP_ETH_RAM, 25, 20},
    // {TCP_PURPOSE_BERKELEY_CLIENT, TCP_ETH_RAM, 125, 100},
};
#define END_OF_TCP_CONFIGURATION

```

As you can see from the structure parameters, the four parameters you'll need to include in each of your socket declarations are:

- Socket purpose/type
- RAM storage location
- TX FIFO buffer size
- RX FIFO buffer size

Several example socket declarations are listed. The socket purpose for each corresponds to one of the socket types (see page 146). The RAM storage for each socket example sets the location to `TCP_ETH_RAM` (the MAC/PHY chip RAM). Other options are `TCP_PIC_RAM` (the PIC's own RAM) and `TCP_SPI_RAM` (an external SPI RAM device). Finally, the TX and RX FIFOs are declared. Each RX buffer must contain at least one byte, to handle the SYN and FIN messages required by TCP. Each socket you declare will require up to 48 bytes of PIC RAM, and 40 + (TX FIFO size) + (RX FIFO size) bytes of RAM on the storage medium that you select.

### 9.3.3.4 UDP Sockets

UDP sockets are somewhat easier to declare than TCP sockets. Since UDP transmissions don't have to be processed in a particular order and responses aren't required by the sender, you don't have to declare separate buffers for these sockets. There are two options to define when using UDP:

```
#define MAX_UDP_SOCKETS      (10u)
//#define UDP_USE_TX_CHECKSUM
```

The `MAX_UDP_SOCKETS` definition defines the size of an array of `UDP_SOCKET_INFO` (see page 534) structures. These structures contain two sixteen-bit identifiers for the remote node's and local node's UDP port numbers, and a 10-byte structure used to hold the remote node's MAC address and IP address (these structures use the packed attribute, so the actual size of the `UDP_SOCKET_INFO` (see page 534) structure may vary slightly depending on the PIC architecture you use).

The `UDP_USE_TX_CHECKSUM` definition will cause the stack to generate checksums for transmitted data, and include them with transmitted packets. This can provide some data integrity verification, but it will also decrease TX performance by nearly 50% unless the ENCX24J600 is used (the ENCX24J600 chips include hardware checksum calculators).

### 9.3.3.5 BSD Sockets

The Berkeley API socket configuration option (see page 145) will require Berkeley sockets. Each one of these internally uses one TCP or UDP socket, defined by the `TCPsocketInitializer[]` array (see page 147) and the `MAX_UDP_SOCKETS` (see page 148) definition. Because of this, the number of Berkeley sockets you declare must be less than or equal to the sum of the number of UDP sockets you declare and the number of TCP Berkeley-type sockets you declare. The `TCPIPConfig.h` macro to define the number of Berkeley sockets is:

```
#define BSD_SOCKET_COUNT      (5u)
```

# 10 Stack API

## Modules

Name	Description
Announce ( <a href="#">see page 149</a> )	Provides a UDP MAC address announcement feature.
ARP ( <a href="#">see page 151</a> )	Provides Address ( <a href="#">see page 141</a> ) Resolution Protocol support.
Berkeley (BSD) Sockets ( <a href="#">see page 161</a> )	Provides a BSD socket wrapper to the Microchip TCP/IP Stack.
DNS Client ( <a href="#">see page 178</a> )	Provides Domain Name Service resolution.
Dynamic DNS Client ( <a href="#">see page 187</a> )	Updates an external IP address to a Dynamic DNS service.
Hashes ( <a href="#">see page 196</a> )	Calculates MD5 and SHA-1 hash sums.
Helpers ( <a href="#">see page 206</a> )	Provides several helper function for stack operation.
HTTP2 Server ( <a href="#">see page 224</a> )	Provides an advanced embedded web server.
ICMP ( <a href="#">see page 257</a> )	Provides Ping functionality.
MPFS2 ( <a href="#">see page 265</a> )	Provides a light-weight file system.
NBNS ( <a href="#">see page 284</a> )	Describes the NetBIOS Name Service protocol.
Performance Tests ( <a href="#">see page 287</a> )	Tests TCP and UDP performance of an application.
SMTP Client ( <a href="#">see page 291</a> )	Sends e-mail messages across the internet.
Reboot ( <a href="#">see page 311</a> )	Provides a service to remotely reboot the PIC.
SNMP ( <a href="#">see page 312</a> )	Provides an Simple Network Management Protocol agent.
SNTP Client ( <a href="#">see page 368</a> )	Obtains absolute time stamps from a pool of network time servers.
SSL ( <a href="#">see page 373</a> )	Implements SSL encryption for TCP connections.
TCP ( <a href="#">see page 434</a> )	Implements the TCP transport layer protocol.
Telnet ( <a href="#">see page 481</a> )	Describes the operation of the Telnet module.
TFTP ( <a href="#">see page 485</a> )	Describes the TFTP module.
Tick Module ( <a href="#">see page 509</a> )	Provides accurate time-keeping capabilities.
UDP ( <a href="#">see page 516</a> )	Implements the UDP transport layer protocol.

## Description

The Microchip TCP/IP Stack is implemented as a suite of modules. Each module exists on its own layer in the TCP/IP layer model, and has its own set of APIs. These APIs are described in this section

## 10.1 Announce

This module will facilitate device discovery on DHCP enabled networks by broadcasting a UDP message on port 30303 whenever the local IP address changes. You can change the port used by the announce module by changing the following macro definition in `Announce.c`.

```
#define ANNOUNCE_PORT 30303
```

The Announce protocol is designed to be used with the TCP/IP Discoverer ([see page 60](#)) PC program.

## 10.1.1 Announce Stack Members

### Functions

	Name	Description
☞	AnnounceIP (☞ see page 150)	Transmits an Announce (☞ see page 149) packet.
☞	DiscoveryTask (☞ see page 150)	Announce (☞ see page 149) callback task.

### Module

Announce (☞ see page 149)

### Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.1.1.1 AnnounceIP Function

#### File

Announce.h

#### C

```
void AnnounceIP();
```

#### Side Effects

None

#### Returns

None

#### Description

AnnounceIP opens a UDP socket and transmits a broadcast packet to port 30303. If a computer is on the same subnet and a utility is looking for packets on the UDP port, it will receive the broadcast. For this application, it is used to announce the change of this board's IP address. The messages can be viewed with the TCP/IP Discoverer software tool.

#### Remarks

A UDP socket must be available before this function is called. It is freed at the end of the function. MAX\_UDP\_SOCKETS may need to be increased if other modules use UDP sockets.

#### Preconditions

Stack is initialized()

### 10.1.1.2 DiscoveryTask Function

#### File

Announce.h

#### C

```
void DiscoveryTask();
```

**Side Effects**

None

**Returns**

None

**Description**

Recurring task used to listen (see page 169) for Discovery messages on the specified ANNOUNCE\_PORT. These messages can be sent using the Microchip Device Discoverer tool. If one is received, this function will transmit a reply.

**Remarks**

A UDP socket must be available before this function is called. It is freed at the end of the function. MAX\_UDP\_SOCKETS may need to be increased if other modules use UDP sockets.

**Preconditions**

Stack is initialized()

# 10.2 ARP

**Types**

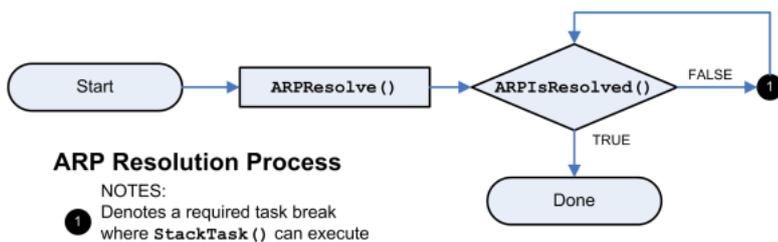
	Name	Description
	ARP_PACKET (see page 160)	ARP packet structure

**Description**

The Address (see page 141) Resolution Protocol, or ARP, is a foundation layer of TCP/IP. It translates IP addresses to physical MAC addresses, or locates a gateway through which a machine may be located.

TCP and UDP applications will not need to access ARP directly. The TCPOpen (see page 451) and UDPOpen (see page 520) functions will handle both ARP and DNS operations transparently.

Responses to incoming ARP requests are processed automatically. Resolution of ARP requests follows a simple state machine, as indicated in the following diagram.



## 10.2.1 ARP Public Members

**Functions**

	Name	Description
	ARPResolve (see page 152)	Transmits an ARP request to resolve an IP address.
	ARPIsResolved (see page 153)	Determines if an ARP request has been resolved yet.

	ARPDeRegisterCallbacks ( <a href="#">↗</a> see page 153)	De-Registering callbacks with ARP module that are registered previously.
	ARPRegisterCallbacks ( <a href="#">↗</a> see page 154)	Registering callback with ARP module to get notified about certian events.
	ARPSendPkt ( <a href="#">↗</a> see page 154)	Transmits an ARP request/Reply initiated by Application or external module.

**Macros**

	Name	Description
	ARP_REQ ( <a href="#">↗</a> see page 155)	Operation code indicating an ARP Request
	ARP_RESP ( <a href="#">↗</a> see page 155)	Operation code indicating an ARP Response
	MAX_REG_APPS ( <a href="#">↗</a> see page 155)	MAX num allowed registrations of Modules/Apps

**Module**

ARP ([↗](#) see page 151)

**Structures**

	Name	Description
	arp_app_callbacks ( <a href="#">↗</a> see page 155)	This is record arp_app_callbacks.

**Description**

The following functions and variables are available to the stack application.

## 10.2.1.1 ARPResolve Function

**File**

ARP.h

**C**

```
void ARPResolve(
    IP_ADDR* IPAddr
);
```

**Returns**

None

**Description**

This function transmits and ARP request to determine the hardware address of a given IP address.

**Remarks**

This function is only required when the stack is a client, and therefore is only enabled when STACK\_CLIENT\_MODE is enabled.

To retrieve the ARP query result, call the ARPisResolved ([↗](#) see page 153)() function.

**Preconditions**

None

**Parameters**

Parameters	Description
IPAddr	The IP address to be resolved. The address must be specified in network byte order (big endian).

## 10.2.1.2 ARPisResolved Function

**File**

ARP.h

**C**

```

BOOL ARPisResolved(
    IP_ADDR* IPAddr,
    MAC_ADDR* MACAddr
);
    
```

**Description**

This function checks if an ARP request has been resolved yet, and if so, stores the resolved MAC address in the pointer provided.

**Remarks**

This function is only required when the stack is a client, and therefore is only enabled when STACK\_CLIENT\_MODE is enabled.

**Preconditions**

ARP packet is ready in the MAC buffer.

**Parameters**

Parameters	Description
IPAddr	The IP address to be resolved. This must match the IP address provided to the ARPResolve (see page 152)() function call.
MACAddr	A buffer to store the corresponding MAC address retrieved from the ARP query.

**Return Values**

Return Values	Description
TRUE	The IP address has been resolved and MACAddr MAC address field indicates the response.
FALSE	The IP address is not yet resolved. Try calling ARPisResolved() again at a later time. If you don't get a response after a application specific timeout period, you may want to call ARPResolve (see page 152)() again to transmit another ARP query (in case if the original query or response was lost on the network). If you never receive an ARP response, this may indicate that the IP address isn't in use.

## 10.2.1.3 ARPDeRegisterCallbacks Function

**File**

ARP.h

**C**

```

BOOL ARPDeRegisterCallbacks(
    CHAR id
);
    
```

**Returns**

TRUE - On success FALSE - Failure to indicate invalid reg\_id

**Description**

This function allows end user-application to de-register with callbacks, which were registered previously. This is called by user-application, when its no longer interested in notifications from ARP-Module. This allows the other application to get registered with ARP-module.

**Preconditions**

None

**Parameters**

Parameters	Description
reg_id	Registration-id returned in ARPRegisterCallbacks (see page 154) call

## 10.2.1.4 ARPRegisterCallbacks Function

**File**

ARP.h

**C**

```
CHAR ARPRegisterCallbacks(
    struct arp_app_callbacks * app
);
```

**Returns**

id > 0 - Returns non-negative value that represents the id of registration The same id needs to be used in de-registration -1 - When registered applications exceed MAX\_REG\_APPS (see page 155) and there is no free slot for registration

**Description**

This function allows end user application to register with callbacks, which will be called by ARP module to give notification to user-application about events occurred at ARP layer. For ex: when a ARP-packet is received, which is conflicting with our own pair of addresses (MAC-Address (see page 141) and IP-address). This is an extension for zeroconf protocol implementation (ZeroconfLL.c)

**Preconditions**

None

**Parameters**

Parameters	Description
app	ARP-Application callbacks structure supplied by user-application

## 10.2.1.5 ARPSendPkt Function

**File**

ARP.h

**C**

```
BOOL ARPSendPkt(
    DWORD SrcIPAddr,
    DWORD DestIPAddr,
    BYTE op_req
);
```

**Returns**

TRUE - The ARP packet was generated properly FALSE - Not possible return value

**Description**

This function transmits and ARP request/reply to determine the hardware address of a given IP address (or) Announce (see page 149) self-address to all nodes in network. Extended for zeroconf protocol.

**Remarks**

This API is to give control over AR-packet to external modules.

**Preconditions**

ARP packet is ready in the MAC buffer.

**Parameters**

Parameters	Description
SrcIPAddr	The Source IP-address
DestIPAddr	The Destination IP-Address ( <a href="#">↗</a> see page 141)
op_req	Operation Request (ARP_REQ ( <a href="#">↗</a> see page 155)/ARP_RESP ( <a href="#">↗</a> see page 155))

**10.2.1.6 arp\_app\_callbacks Structure****File**

ARP.h

**C**

```

struct arp_app_callbacks {
    BOOL used;
    void (* ARPPkt_notify)(DWORD SenderIPAddr, DWORD TargetIPAddr, MAC_ADDR* SenderMACAddr,
    MAC_ADDR* TargetMACAddr, BYTE op_req);
};

```

**Description**

This is record arp\_app\_callbacks.

**10.2.1.7 ARP\_REQ Macro****File**

ARP.h

**C**

```

#define ARP_REQ 0x0001u           // Operation code indicating an ARP Request

```

**Description**

Operation code indicating an ARP Request

**10.2.1.8 ARP\_RESP Macro****File**

ARP.h

**C**

```

#define ARP_RESP 0x0002u        // Operation code indicating an ARP Response

```

**Description**

Operation code indicating an ARP Response

**10.2.1.9 MAX\_REG\_APPS Macro****File**

ARP.c

**C**

```
#define MAX_REG_APPS 2           // MAX num allowed registrations of Modules/Apps
```

**Description**

MAX num allowed registrations of Modules/Apps

---

## 10.2.2 ARP Stack Members

**Functions**

	Name	Description
⇒	ARPInit (see page 156)	Initializes the ARP module.
⇒	ARPPProcess (see page 156)	Processes an incoming ARP packet.

**Module**

ARP (see page 151)

**Description**

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.2.2.1 ARPInit Function

**File**

ARP.h

**C**

```
void ARPInit();
```

**Returns**

None

**Description**

Initializes the ARP module. Call this function once at boot to invalidate the cached lookup.

**Remarks**

This function is only required when the stack is a client, and therefore is only enabled when STACK\_CLIENT\_MODE is enabled.

**Preconditions**

None

### 10.2.2.2 ARPPProcess Function

**File**

ARP.h

**C**

```
BOOL ARPPProcess();
```

**Description**

Retrieves an ARP packet from the MAC buffer and determines if it is a response to our request (in which case the ARP is resolved) or if it is a request requiring our response (in which case we transmit one.)

**Preconditions**

ARP packet is ready in the MAC buffer.

**Return Values**

Return Values	Description
TRUE	All processing of this ARP packet is complete. Do not call again until a new ARP packet is waiting in the RX buffer.
FALSE	This function must be called again. More time is needed to send an ARP response.

## 10.2.3 ARP Internal Members

**Functions**

	Name	Description
⇒	ARPPut (see page 158)	Writes an ARP packet to the MAC.
⇒	SwapARPPacket (see page 158)	Swaps endian-ness of header information in an ARP packet.

**Macros**

	Name	Description
⇒	ARP_OPERATION_REQ (see page 159)	Operation code indicating an ARP Request
⇒	ARP_OPERATION_RESP (see page 159)	Operation code indicating an ARP Response
⇒	HW_ETHERNET (see page 159)	ARP Hardware type as defined by IEEE 802.3
⇒	ARP_IP (see page 159)	ARP IP packet type as defined by IEEE 802.3

**Module**

ARP (see page 151)

**Variables**

	Name	Description
◆	Cache (see page 159)	Cache for one ARP response
◆	reg_apps (see page 160)	Call-Backs storage for MAX of two Modules/Apps <pre>// ARP packet structure typedef struct __attribute__((aligned(2), packed)) { WORD HardwareType; WORD Protocol; BYTE MACAddrLen; BYTE ProtocolLen; WORD Operation; MAC_ADDR SenderMACAddr; IP_ADDR SenderIPAddr; MAC_ADDR TargetMACAddr; IP_ADDR TargetIPAddr; } ARP_PACKET (see page 160);</pre>

**Description**

The following functions and variables are designated as internal to the ARP module.

## 10.2.3.1 ARPPut Function

### File

ARP.c

### C

```
static BOOL ARPPut(
    ARP_PACKET* packet
);
```

### Description

Writes an ARP packet to the MAC.

### Preconditions

None

### Parameters

Parameters	Description
packet	A pointer to an ARP_PACKET (see page 160) structure with correct operation and target preconfigured.

### Return Values

Return Values	Description
TRUE	The ARP packet was generated properly
FALSE	Not a possible return value

### Section

Helper Function Prototypes

## 10.2.3.2 SwapARPPacket Function

### File

ARP.h

### C

```
void SwapARPPacket(
    ARP_PACKET* p
);
```

### Returns

None

### Description

Swaps endian-ness of header information in an ARP packet.

### Preconditions

None

### Parameters

Parameters	Description
p	The ARP packet to be swapped

### 10.2.3.3 ARP\_OPERATION\_REQ Macro

**File**

ARP.h

**C**

```
#define ARP_OPERATION_REQ 0x0001u // Operation code indicating an ARP Request
```

**Description**

Operation code indicating an ARP Request

### 10.2.3.4 ARP\_OPERATION\_RESP Macro

**File**

ARP.h

**C**

```
#define ARP_OPERATION_RESP 0x0002u // Operation code indicating an ARP Response
```

**Description**

Operation code indicating an ARP Response

### 10.2.3.5 HW\_ETHERNET Macro

**File**

ARP.c

**C**

```
#define HW_ETHERNET (0x0001u) // ARP Hardware type as defined by IEEE 802.3
```

**Description**

ARP Hardware type as defined by IEEE 802.3

### 10.2.3.6 ARP\_IP Macro

**File**

ARP.c

**C**

```
#define ARP_IP (0x0800u) // ARP IP packet type as defined by IEEE 802.3
```

**Description**

ARP IP packet type as defined by IEEE 802.3

### 10.2.3.7 Cache Variable

**File**

ARP.c

**C**

NODE\_INFO Cache;

**Description**

Cache for one ARP response

### 10.2.3.8 reg\_apps Variable

**File**

ARP.c

**C**

```
struct arp_app_callbacks reg_apps[MAX_REG_APPS];
```

**Description**

Call-Backs storage for MAX of two Modules/Apps

```
// ARP packet structure typedef struct __attribute__((aligned(2), packed)) { WORD HardwareType; WORD Protocol; BYTE MACAddrLen; BYTE ProtocolLen; WORD Operation; MAC_ADDR SenderMACAddr; IP_ADDR SenderIPAddr; MAC_ADDR TargetMACAddr; IP_ADDR TargetIPAddr; } ARP_PACKET (see page 160);
```

## 10.2.4 Types

**Module**

ARP (see page 151)

**Structures**

	Name	Description
	ARP_PACKET (see page 160)	ARP packet structure

### 10.2.4.1 ARP\_PACKET Structure

**File**

ARP.h

**C**

```
typedef struct {
    WORD HardwareType;
    WORD Protocol;
    BYTE MACAddrLen;
    BYTE ProtocolLen;
    WORD Operation;
    MAC_ADDR SenderMACAddr;
    IP_ADDR SenderIPAddr;
    MAC_ADDR TargetMACAddr;
    IP_ADDR TargetIPAddr;
} ARP_PACKET;
```

**Members**

Members	Description
WORD HardwareType;	Link-layer protocol type (Ethernet is 1).
WORD Protocol;	The upper-layer protocol issuing an ARP request (0x0800 for IPv4)..

BYTE MACAddrLen;	MAC address length (6).
BYTE ProtocolLen;	Length of addresses used in the upper-layer protocol (4).
WORD Operation;	The operation the sender is performing (ARP_REQ (see page 155) or ARP_RESP (see page 155)).
MAC_ADDR SenderMACAddr;	The sender's hardware (MAC) address.
IP_ADDR SenderIPAddr;	The sender's IP address.
MAC_ADDR TargetMACAddr;	The target node's hardware (MAC) address.
IP_ADDR TargetIPAddr;	The target node's IP address.

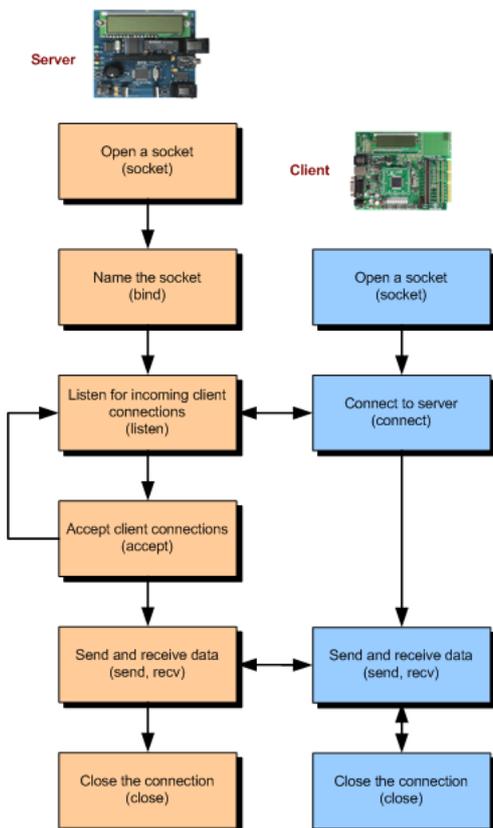
**Description**

ARP packet structure

# 10.3 Berkeley (BSD) Sockets

The Berkeley Socket Distribution (BSD) APIs provide a BSD wrapper to the native Microchip TCP/IP Stack APIs. Using this interface, programmers familiar with BSD sockets can quickly develop applications using Microchip's TCP/IP Stack.

The illustration below shows a typical interaction for a TCP server or client using the Berkeley socket APIs.



## 10.3.1 BSD Wrapper Public Members

### Functions

	Name	Description
⇒	<a href="#">accept</a> ( <a href="#">see page 163</a> )	This function accepts connection requests queued for a listening socket.
⇒	<a href="#">bind</a> ( <a href="#">see page 164</a> )	This function assigns a name to the socket descriptor.
⇒	<a href="#">closesocket</a> ( <a href="#">see page 165</a> )	The closesocket function closes an existing socket.
⇒	<a href="#">connect</a> ( <a href="#">see page 165</a> )	This function connects to the peer communications end point.
⇒	<a href="#">gethostname</a> ( <a href="#">see page 166</a> )	Returns the standard host name for the system.
⇒	<a href="#">listen</a> ( <a href="#">see page 169</a> )	The listen function sets the specified socket in a listen mode
⇒	<a href="#">recv</a> ( <a href="#">see page 169</a> )	The recv() function is used to receive incoming data that has been queued for a socket.
⇒	<a href="#">recvfrom</a> ( <a href="#">see page 170</a> )	The recvfrom() function is used to receive incoming data that has been queued for a socket.
⇒	<a href="#">send</a> ( <a href="#">see page 171</a> )	The send function is used to send outgoing data on an already connected socket.
⇒	<a href="#">sendto</a> ( <a href="#">see page 171</a> )	This function used to send the data for both connection oriented and connection-less sockets.
⇒	<a href="#">socket</a> ( <a href="#">see page 174</a> )	This function creates a new Berkeley socket.

### Macros

	Name	Description
⇒	<a href="#">AF_INET</a> ( <a href="#">see page 164</a> )	Internet Address ( <a href="#">see page 141</a> ) Family - UDP, TCP, etc.
⇒	<a href="#">INADDR_ANY</a> ( <a href="#">see page 167</a> )	IP address for server binding.
⇒	<a href="#">INVALID_TCP_PORT</a> ( <a href="#">see page 168</a> )	Invalide TCP port
⇒	<a href="#">IP_ADDR_ANY</a> ( <a href="#">see page 168</a> )	IP Address ( <a href="#">see page 141</a> ) for server binding
⇒	<a href="#">IPPROTO_IP</a> ( <a href="#">see page 168</a> )	Indicates IP pseudo-protocol.
⇒	<a href="#">IPPROTO_TCP</a> ( <a href="#">see page 168</a> )	Indicates TCP for the internet address family.
⇒	<a href="#">IPPROTO_UDP</a> ( <a href="#">see page 168</a> )	Indicates UDP for the internet address family.
⇒	<a href="#">SOCK_DGRAM</a> ( <a href="#">see page 172</a> )	Connectionless datagram socket. Use UDP for the internet address family.
⇒	<a href="#">SOCK_STREAM</a> ( <a href="#">see page 172</a> )	Connection based byte streams. Use TCP for the internet address family.
⇒	<a href="#">SOCKET_CNXN_IN_PROGRESS</a> ( <a href="#">see page 174</a> )	Socket connection state.
⇒	<a href="#">SOCKET_DISCONNECTED</a> ( <a href="#">see page 175</a> )	Socket disconnected
⇒	<a href="#">SOCKET_ERROR</a> ( <a href="#">see page 175</a> )	Socket error

### Module

Berkeley (BSD) Sockets ([see page 161](#))

**Structures**

	Name	Description
	BSDSocket ( <a href="#">see page 164</a> )	Berkeley Socket structure
	in_addr ( <a href="#">see page 167</a> )	in_addr structure
	sockaddr ( <a href="#">see page 172</a> )	generic address structure for all address families
	sockaddr_in ( <a href="#">see page 173</a> )	In the Internet address family

**Types**

	Name	Description
	SOCKADDR ( <a href="#">see page 173</a> )	generic address structure for all address families
	SOCKADDR_IN ( <a href="#">see page 173</a> )	In the Internet address family
	SOCKET ( <a href="#">see page 174</a> )	Socket descriptor

**Description**

The following functions and variables are available to the stack application.

## 10.3.1.1 accept Function

**File**

BerkeleyAPI.h

**C**

```
SOCKET accept (
    SOCKET s,
    struct sockaddr* addr,
    int* addrlen
);
```

**Returns**

If the accept function succeeds, it returns a non-negative integer that is a descriptor for the accepted socket. Otherwise, the value INVALID\_SOCKET ([see page 437](#)) is returned.

**Description**

The accept function is used to accept connection requests queued for a listening socket. If a connection request is pending, accept removes the request from the queue, and a new socket is created for the connection. The original listening socket remains open and continues to queue new connection requests. The socket must be a SOCK\_STREAM ([see page 172](#)) type socket.

**Remarks**

None.

**Preconditions**

listen ([see page 169](#)) function should be called.

**Parameters**

Parameters	Description
s	Socket descriptor returned from a previous call to socket. must be bound to a local name and in listening mode.
addr	Optional pointer to a buffer that receives the address of the connecting entity.
addrlen	Optional pointer to an integer that contains the length of the address addr

## 10.3.1.2 AF\_INET Macro

### File

BerkeleyAPI.h

### C

```
#define AF_INET 2 // Internet Address Family - UDP, TCP, etc.
```

### Description

Internet Address (see page 141) Family - UDP, TCP, etc.

## 10.3.1.3 bind Function

### File

BerkeleyAPI.h

### C

```
int bind(
    SOCKET s,
    const struct sockaddr* name,
    int namelen
);
```

### Returns

If bind is successful, a value of 0 is returned. A return value of SOCKET\_ERROR (see page 175) indicates an error.

### Description

The bind function assigns a name to an unnamed socket. The name represents the local address of the communication endpoint. For sockets of type SOCK\_STREAM (see page 172), the name of the remote endpoint is assigned when a connect (see page 165) or accept (see page 163) function is executed.

### Remarks

None.

### Preconditions

socket function should be called.

### Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket.
name	pointer to the sockaddr (see page 172) structure containing the local address of the socket.
namelen	length of the sockaddr (see page 172) structure.

## 10.3.1.4 BSDSocket Structure

### File

BerkeleyAPI.h

### C

```
struct BSDSocket {
    int SocketType;
    BSD_SCK_STATE bsdState;
    WORD localPort;
};
```

```

WORD remotePort;
DWORD remoteIP;
int backlog;
BOOL isServer;
TCP_SOCKET SocketID;
};

```

**Members**

Members	Description
int SocketType;	Socket type
BSD_SCK_STATE bsdState;	Socket state
WORD localPort;	local port
WORD remotePort;	remote port
DWORD remoteIP;	remote IP
int backlog;	maximum number of client connections
BOOL isServer;	server/client check
TCP_SOCKET SocketID;	Socket ID

**Description**

Berkeley Socket structure

### 10.3.1.5 closesocket Function

**File**

BerkeleyAPI.h

**C**

```

int closesocket(
    SOCKET s
);

```

**Returns**

If closesocket is successful, a value of 0 is returned. A return value of SOCKET\_ERROR (see page 175) (-1) indicates an error.

**Description**

The closesocket function closes an existing socket. This function releases the socket descriptor s. Any data buffered at the socket is discarded. If the socket s is no longer needed, closesocket() must be called in order to release all resources associated with s.

**Remarks**

None.

**Preconditions**

None.

**Parameters**

Parameters	Description
s	Socket descriptor returned from a previous call to socket

### 10.3.1.6 connect Function

**File**

BerkeleyAPI.h

**C**

```
int connect(
    SOCKET s,
    struct sockaddr* name,
    int namelen
);
```

**Returns**

If the connect() function succeeds, it returns 0. Otherwise, the value SOCKET\_ERROR (see page 175) is returned to indicate an error condition. For stream based socket, if the connection is not established yet, connect returns SOCKET\_CNXXN\_IN\_PROGRESS (see page 174).

**Description**

The connect function assigns the address of the peer communications endpoint. For stream sockets, connection is established between the endpoints. For datagram sockets, an address filter is established between the endpoints until changed with another connect() function.

**Remarks**

None.

**Preconditions**

socket function should be called.

**Parameters**

Parameters	Description
s	Socket descriptor returned from a previous call to socket.
name	pointer to the sockaddr (see page 172) structure containing the peer address and port number.
namelen	length of the sockaddr (see page 172) structure.

## 10.3.1.7 gethostname Function

**File**

BerkeleyAPI.h

**C**

```
int gethostname(
    char* name,
    int namelen
);
```

**Returns**

Success will return a value of 0. If name is too short to hold the host name or any other error occurs, SOCKET\_ERROR (see page 175) (-1) will be returned. On error, \*name will be unmodified and no null terminator will be generated.

**Description**

This function returns the standard host name of the system which is calling this function. The returned name is null-terminated.

**Remarks**

None.

**Preconditions**

None.

**Parameters**

Parameters	Description
name	Pointer to a buffer that receives the local host name.
namelen	size of the name array.

**10.3.1.8 in\_addr Structure****File**

BerkeleyAPI.h

**C**

```

struct in_addr {
    union {
        struct {
            BYTE s_b1, s_b2, s_b3, s_b4;
        } S_un_b;
        struct {
            WORD s_w1, s_w2;
        } S_un_w;
        DWORD S_addr;
    } S_un;
};

```

**Members**

Members	Description
union { struct { BYTE s_b1, s_b2, s_b3, s_b4; } S_un_b; struct { WORD s_w1, s_w2; } S_un_w; DWORD S_addr; } S_un;	union of IP address
struct { BYTE s_b1, s_b2, s_b3, s_b4; } S_un_b;	IP address in Byte
struct { WORD s_w1, s_w2; } S_un_w;	IP address in Word
DWORD S_addr;	IP address

**Description**

in\_addr structure

**10.3.1.9 INADDR\_ANY Macro****File**

BerkeleyAPI.h

**C**

```

#define INADDR_ANY 0x00000000u // IP address for server binding.

```

**Description**

IP address for server binding.

### 10.3.1.10 INVALID\_TCP\_PORT Macro

**File**

BerkeleyAPI.h

**C**

```
#define INVALID_TCP_PORT (0L) //Invalide TCP port
```

**Description**

Invalide TCP port

### 10.3.1.11 IP\_ADDR\_ANY Macro

**File**

BerkeleyAPI.h

**C**

```
#define IP_ADDR_ANY 0u // IP Address for server binding
```

**Description**

IP Address (see page 141) for server binding

### 10.3.1.12 IPPROTO\_IP Macro

**File**

BerkeleyAPI.h

**C**

```
#define IPPROTO_IP 0 // Indicates IP pseudo-protocol.
```

**Description**

Indicates IP pseudo-protocol.

### 10.3.1.13 IPPROTO\_TCP Macro

**File**

BerkeleyAPI.h

**C**

```
#define IPPROTO_TCP 6 // Indicates TCP for the internet address family.
```

**Description**

Indicates TCP for the internet address family.

### 10.3.1.14 IPPROTO\_UDP Macro

**File**

BerkeleyAPI.h

**C**

```
#define IPPROTO_UDP 17 // Indicates UDP for the internet address family.
```

**Description**

Indicates UDP for the internet address family.

## 10.3.1.15 listen Function

**File**

BerkeleyAPI.h

**C**

```
int listen(
    SOCKET s,
    int backlog
);
```

**Returns**

Returns 0 on success, else return SOCKET\_ERROR (see page 175).

**Description**

This function sets the specified socket in a listen mode. Calling the listen function indicates that the application is ready to accept (see page 163) connection requests arriving at a socket of type SOCK\_STREAM (see page 172). The connection request is queued (if possible) until accepted with an accept (see page 163) function. The backlog parameter defines the maximum number of pending connections that may be queued.

**Remarks**

None

**Preconditions**

bind() must have been called on the s socket first.

**Parameters**

Parameters	Description
s	Socket identifier returned from a prior socket() call.
backlog	Maximum number of connection requests that can be queued. Note that each backlog requires a TCP_PURPOSE_BERKELEY_SERVER type TCP socket to be allocated in the TCPsocketInitializer[] in TCPIPConfig.h. Also, ensure that BSD_SOCKET_COUNT (also in TCPIPConfig.h) is greater than the backlog by at least 1 (more if you have other BSD sockets in use).

## 10.3.1.16 recv Function

**File**

BerkeleyAPI.h

**C**

```
int recv(
    SOCKET s,
    char* buf,
    int len,
    int flags
);
```

**Returns**

If recv is successful, the number of bytes copied to application buffer buf is returned. A value of zero indicates no data

available. A return value of `SOCKET_ERROR` (see page 175) (-1) indicates an error condition. A return value of `SOCKET_DISCONNECTED` (see page 175) indicates the connection no longer exists.

### Description

The `recv()` function is used to receive incoming data that has been queued for a socket. This function can be used with both datagram and stream socket. If the available data is too large to fit in the supplied application buffer `buf`, excess bytes are discarded in case of `SOCK_DGRAM` (see page 172) type sockets. For `SOCK_STREAM` (see page 172) types, the data is buffered internally so the application can retrieve all data by multiple calls of `recvfrom` (see page 170).

### Remarks

None.

### Preconditions

`connect` (see page 165) function should be called for TCP and UDP sockets. Server side, `accept` (see page 163) function should be called.

### Parameters

Parameters	Description
<code>s</code>	Socket descriptor returned from a previous call to <code>socket</code> .
<code>buf</code>	application data receive buffer.
<code>len</code>	buffer length in bytes.
<code>flags</code>	no significance in this implementation

## 10.3.1.17 recvfrom Function

### File

BerkeleyAPI.h

### C

```
int recvfrom(
    SOCKET s,
    char* buf,
    int len,
    int flags,
    struct sockaddr* from,
    int* fromlen
);
```

### Returns

If `recvfrom` is successful, the number of bytes copied to application buffer `buf` is returned. A value of zero indicates no data available. A return value of `SOCKET_ERROR` (see page 175) (-1) indicates an error condition.

### Description

The `recvfrom()` function is used to receive incoming data that has been queued for a socket. This function can be used with both datagram and stream type sockets. If the available data is too large to fit in the supplied application buffer `buf`, excess bytes are discarded in case of `SOCK_DGRAM` (see page 172) type sockets. For `SOCK_STREAM` (see page 172) types, the data is buffered internally so the application can retrieve all data by multiple calls of `recvfrom`.

### Remarks

None.

### Preconditions

`socket` function should be called.

### Parameters

Parameters	Description
<code>s</code>	Socket descriptor returned from a previous call to <code>socket</code> .

buf	application data receive buffer.
len	buffer length in bytes.
flags	message flags. Currently this is not supported.
from	pointer to the sockaddr (see page 172) structure that will be filled in with the destination address.
fromlen	size of buffer pointed by from.

## 10.3.1.18 send Function

### File

BerkeleyAPI.h

### C

```
int send(
    SOCKET s,
    const char* buf,
    int len,
    int flags
);
```

### Returns

On success, send returns number of bytes sent. In case of error, returns SOCKET\_ERROR (see page 175). a zero indicates no data send.

### Description

The send function is used to send outgoing data on an already connected socket. This function is used to send a reliable, ordered stream of data bytes on a socket of type SOCK\_STREAM (see page 172) but can also be used to send datagrams on a socket of type SOCK\_DGRAM (see page 172).

### Remarks

None.

### Preconditions

connect (see page 165) function should be called for TCP and UDP sockets. Server side, accept (see page 163) function should be called.

### Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket.
buf	application data buffer containing data to transmit.
len	length of data in bytes.
flags	message flags. Currently this field is not supported.

## 10.3.1.19 sendto Function

### File

BerkeleyAPI.h

### C

```
int sendto(
    SOCKET s,
    const char* buf,
    int len,
    int flags,
    const struct sockaddr* to,
    int tolen
);
```

```
);
```

**Returns**

On success, sendto returns number of bytes sent. In case of error returns SOCKET\_ERROR (see page 175)

**Description**

The sendto function is used to send outgoing data on a socket. The destination address is given by to and tolen. Both Datagram and stream sockets are supported.

**Remarks**

None.

**Preconditions**

socket function should be called.

**Parameters**

Parameters	Description
s	Socket descriptor returned from a previous call to socket.
buf	application data buffer containing data to transmit.
len	length of data in bytes.
flags	message flags. Currently this field is not supported.
to	Optional pointer to the the sockaddr (see page 172) structure containing the destination address. If NULL, the currently bound remote port and IP address are used as the destination.
tolen	length of the sockaddr (see page 172) structure.

## 10.3.1.20 SOCK\_DGRAM Macro

**File**

BerkeleyAPI.h

**C**

```
#define SOCK_DGRAM 110 //Connectionless datagram socket. Use UDP for the internet address family.
```

**Description**

Connectionless datagram socket. Use UDP for the internet address family.

## 10.3.1.21 SOCK\_STREAM Macro

**File**

BerkeleyAPI.h

**C**

```
#define SOCK_STREAM 100 //Connection based byte streams. Use TCP for the internet address family.
```

**Description**

Connection based byte streams. Use TCP for the internet address family.

## 10.3.1.22 sockaddr Structure

**File**

BerkeleyAPI.h

**C**

```

struct sockaddr {
    unsigned short sa_family;
    char sa_data[14];
};

```

**Members**

Members	Description
unsigned short sa_family;	address family
char sa_data[14];	up to 14 bytes of direct address

**Description**

generic address structure for all address families

## 10.3.1.23 SOCKADDR Type

**File**

BerkeleyAPI.h

**C**

```

typedef struct sockaddr SOCKADDR;

```

**Description**

generic address structure for all address families

## 10.3.1.24 sockaddr\_in Structure

**File**

BerkeleyAPI.h

**C**

```

struct sockaddr_in {
    short sin_family;
    WORD sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};

```

**Members**

Members	Description
short sin_family;	Address ( <a href="#">see page 141</a> ) family; must be AF_INET ( <a href="#">see page 164</a> ).
WORD sin_port;	Internet Protocol (IP) port.
struct in_addr sin_addr;	IP address in network byte order.
char sin_zero[8];	Padding to make structure the same size as SOCKADDR ( <a href="#">see page 173</a> ).

**Description**

In the Internet address family

## 10.3.1.25 SOCKADDR\_IN Type

**File**

BerkeleyAPI.h

**C**

```
typedef struct sockaddr_in SOCKADDR_IN;
```

**Description**

In the Internet address family

## 10.3.1.26 socket Function

**File**

BerkeleyAPI.h

**C**

```
SOCKET socket(
    int af,
    int type,
    int protocol
);
```

**Returns**

New socket descriptor. INVALID\_SOCKET ([see page 437](#)) in case of error.

**Description**

This function creates a new BSD socket for the microchip TCPIP stack. The return socket descriptor is used for the subsequent BSD operations.

**Remarks**

None.

**Preconditions**

BerkeleySocketInit ([see page 175](#)) function should be called.

**Parameters**

Parameters	Description
af	address family - AF_INET ( <a href="#">see page 164</a> ).
type	socket type SOCK_DGRAM ( <a href="#">see page 172</a> ) or SOCK_STREAM ( <a href="#">see page 172</a> ).
protocol	IP protocol IPPROTO_UDP ( <a href="#">see page 168</a> ) or IPPROTO_TCP ( <a href="#">see page 168</a> ).

## 10.3.1.27 SOCKET Type

**File**

BerkeleyAPI.h

**C**

```
typedef BYTE SOCKET;
```

**Description**

Socket descriptor

## 10.3.1.28 SOCKET\_CNXXN\_IN\_PROGRESS Macro

**File**

BerkeleyAPI.h

**C**

```
#define SOCKET_CNXXN_IN_PROGRESS (-2) //Socket connection state.
```

**Description**

Socket connection state.

**10.3.1.29 SOCKET\_DISCONNECTED Macro****File**

BerkeleyAPI.h

**C**

```
#define SOCKET_DISCONNECTED (-3) //Socket disconnected
```

**Description**

Socket disconnected

**10.3.1.30 SOCKET\_ERROR Macro****File**

BerkeleyAPI.h

**C**

```
#define SOCKET_ERROR (-1) //Socket error
```

**Description**

Socket error

**10.3.2 BSD Wrapper Stack Members****Functions**

	Name	Description
	BerkeleySocketInit ( <a href="#">see page 175</a> )	Initializes the Berkeley socket structure array.

**Module**

Berkeley (BSD) Sockets ([see page 161](#))

**Description**

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

**10.3.2.1 BerkeleySocketInit Function****File**

BerkeleyAPI.h

**C**

```
void BerkeleySocketInit();
```

**Returns**

None

**Description**

This function initializes the Berkeley socket array. This function should be called before any BSD socket call.

**Remarks**

None.

**Preconditions**

None.

---

## 10.3.3 BSD Wrapper Internal Members

**Enumerations**

	Name	Description
	BSD_SCK_STATE ( <a href="#">see page 176</a> )	Berkeley Socket (BSD) states

**Functions**

	Name	Description
	HandlePossibleTCPDisconnection ( <a href="#">see page 177</a> )	Internal function that checks for asynchronous TCP connection state changes and resynchs the BSD socket descriptor state to match.

**Module**

Berkeley (BSD) Sockets ([see page 161](#))

**Variables**

	Name	Description
	BSDSocketArray ( <a href="#">see page 177</a> )	Array of BSDSocket ( <a href="#">see page 164</a> ) elements; used to track all socket state and connection information.
	gAutoPortNumber ( <a href="#">see page 177</a> )	Contains the next local port number to associate with a socket.

**Description**

The following functions and variables are designated as internal to the module.

### 10.3.3.1 BSD\_SCK\_STATE Enumeration

**File**

BerkeleyAPI.h

**C**

```
typedef enum {
    SKT_CLOSED,
    SKT_CREATED,
    SKT_BOUND,
    SKT_BSD_LISTEN,
    SKT_LISTEN,
    SKT_IN_PROGRESS,
    SKT_EST,
    SKT_DISCONNECTED
} BSD_SCK_STATE;
```

**Members**

Members	Description
SKT_CLOSED	Socket closed state indicating a free descriptor
SKT_CREATED	Socket created state for TCP and UDP sockets
SKT_BOUND	Socket bound state for TCP and UDP sockets
SKT_BSD_LISTEN	Listening state for TCP BSD listener handle "socket"
SKT_LISTEN	TCP server listen (see page 169) state
SKT_IN_PROGRESS	TCP client connection in progress state
SKT_EST	TCP client or server established state
SKT_DISCONNECTED	TCP client or server no longer connected to the remote host (but was historically)

**Description**

Berkeley Socket (BSD) states

### 10.3.3.2 BSDSocketArray Variable

**File**

BerkeleyAPI.c

**C**

```
struct BSDSocket BSDSocketArray[BSD_SOCKET_COUNT];
```

**Description**

Array of BSDSocket (see page 164) elements; used to track all socket state and connection information.

### 10.3.3.3 gAutoPortNumber Variable

**File**

BerkeleyAPI.c

**C**

```
WORD gAutoPortNumber = 1024;
```

**Description**

Contains the next local port number to associate with a socket.

### 10.3.3.4 HandlePossibleTCPDisconnection Function

**File**

BerkeleyAPI.c

**C**

```
static BOOL HandlePossibleTCPDisconnection(
    SOCKET s
);
```

**Returns**

TRUE - Socket is disconnected FALSE - Socket is

**Description**

Internal function that checks for asynchronous TCP connection state changes and resynchs the BSD socket descriptor state

to match.

**Preconditions**

None

**Parameters**

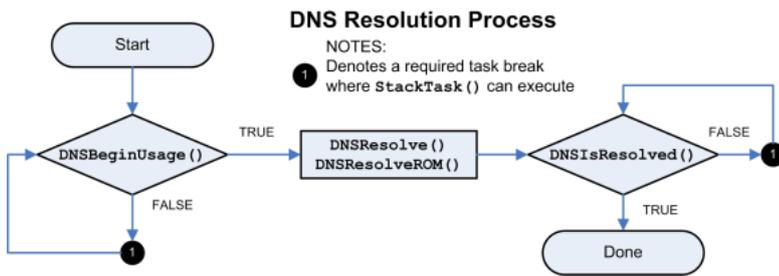
Parameters	Description
s	TCP type socket descriptor returned from a previous call to socket. This socket must be in the SKT_LISTEN, SKT_IN_PROGRESS, SKT_EST, or SKT_DISCONNECTED states.

## 10.4 DNS Client

The Domain Name Service associates host names (such as www.microchip.com) with IP addresses (such as 10.0.54.2). The DNS Client module provides DNS resolution capabilities to the stack.

TCP applications do not need to use the DNS module. Any necessary DNS operations can be handled by the TCPOpen (see page 451) function. Applications built using UDP may need to use DNS when the IP address of the remote server is unknown.

DNS resolution operations follow a simple state machine, as indicated in the diagram below.



### 10.4.1 DNS Public Members

**Functions**

	Name	Description
≡	DNSBeginUsage (see page 179)	Claims access to the DNS module.
≡	DNSEndUsage (see page 179)	Releases control of the DNS module.
≡	DNSResolve (see page 180)	Begins resolution of an address.
≡	DNSResolveROM (see page 180)	Begins resolution of an address.
≡	DNSIsResolved (see page 181)	Determines if the DNS resolution is complete and provides the IP.

**Macros**

	Name	Description
→	DNS_TYPE_A (see page 181)	Constant for record type in DNSResolve (see page 180). Indicates an A (standard address) record.

	DNS_TYPE_MX ( <a href="#">see page 182</a> )	Constant for record type in DNSResolve ( <a href="#">see page 180</a> ). Indicates an MX (mail exchanger) record.
---	--	---

**Module**

DNS Client ([see page 178](#))

**Description**

The following functions and variables are available to the stack application.

## 10.4.1.1 DNSBeginUsage Function

**File**

DNS.h

**C**

```
BOOL DNSBeginUsage();
```

**Description**

This function acts as a semaphore to obtain usage of the DNS module. Call this function and ensure that it returns TRUE before calling any other DNS APIs. Call DNSEndUsage ([see page 179](#)) when this application no longer needs the DNS module so that other applications may make use of it.

**Remarks**

Ensure that DNSEndUsage ([see page 179](#)) is always called once your application has obtained control of the DNS module. If this is not done, the stack will hang for all future applications requiring DNS access.

**Preconditions**

Stack is initialized.

**Return Values**

Return Values	Description
TRUE	No other DNS resolutions are in progress and the calling application has successfully taken ownership of the DNS module
FALSE	The DNS module is currently in use. Yield to the stack and attempt this call again later.

## 10.4.1.2 DNSEndUsage Function

**File**

DNS.h

**C**

```
BOOL DNSEndUsage();
```

**Description**

This function acts as a semaphore to obtain usage of the DNS module. Call this function when this application no longer needs the DNS module so that other applications may make use of it.

**Remarks**

Ensure that DNSEndUsage is always called once your application has obtained control of the DNS module. If this is not done, the stack will hang for all future applications requiring DNS access.

**Preconditions**

DNSBeginUsage ([see page 179](#)) returned TRUE on a previous call.

**Return Values**

Return Values	Description
TRUE	The address to the host name was successfully resolved.
FALSE	The DNS failed or the address does not exist.

### 10.4.1.3 DNSResolve Function

**File**

DNS.h

**C**

```
void DNSResolve(
    BYTE* HostName,
    BYTE Type
);
```

**Returns**

None

**Description**

This function attempts to resolve a host name to an IP address. When called, it starts the DNS state machine. Call `DNSIsResolved` (see page 181) repeatedly to determine if the resolution is complete.

Only one DNS resolution may be executed at a time. The `Hostname` must not be modified in memory until the resolution is complete.

**Remarks**

This function requires access to one UDP socket. If none are available, `MAX_UDP_SOCKETS` may need to be increased.

**Preconditions**

`DNSBeginUsage` (see page 179) returned TRUE on a previous call.

**Parameters**

Parameters	Description
Hostname	A pointer to the null terminated string specifying the host for which to resolve an IP.
RecordType (see page 185)	DNS_TYPE_A (see page 181) or DNS_TYPE_MX (see page 182) depending on what type of record resolution is desired.

### 10.4.1.4 DNSResolveROM Function

**File**

DNS.h

**C**

```
void DNSResolveROM(
    ROM BYTE* Hostname,
    BYTE Type
);
```

**Returns**

None

**Description**

This function attempts to resolve a host name to an IP address. When called, it starts the DNS state machine. Call `DNSIsResolved` (see page 181) repeatedly to determine if the resolution is complete.

Only one DNS resolution may be executed at a time. The Hostname must not be modified in memory until the resolution is complete.

#### Remarks

This function requires access to one UDP socket. If none are available, MAX\_UDP\_SOCKETS may need to be increased.

This function is aliased to DNSResolve (see page 180) on non-PIC18 platforms.

#### Preconditions

DNSBeginUsage (see page 179) returned TRUE on a previous call.

#### Parameters

Parameters	Description
Hostname	A pointer to the null terminated string specifying the host for which to resolve an IP.
RecordType (see page 185)	DNS_TYPE_A (see page 181) or DNS_TYPE_MX (see page 182) depending on what type of record resolution is desired.

### 10.4.1.5 DNSIsResolved Function

#### File

DNS.h

#### C

```

BOOL DNSIsResolved(
    IP_ADDR* HostIP
);

```

#### Description

Call this function to determine if the DNS resolution of an address has been completed. If so, the resolved address will be provided in HostIP.

#### Preconditions

DNSResolve (see page 180) or DNSResolveROM (see page 180) has been called.

#### Parameters

Parameters	Description
HostIP	A pointer to an IP_ADDR structure in which to store the resolved IP address once resolution is complete.

#### Return Values

Return Values	Description
TRUE	The DNS client has obtained an IP, or the DNS process has encountered an error. HostIP will be 0.0.0.0 on error. Possible errors include server timeout (i.e. DNS server not available), hostname not in the DNS, or DNS server errors.
FALSE	The resolution process is still in progress.

### 10.4.1.6 DNS\_TYPE\_A Macro

#### File

DNS.h

#### C

```

#define DNS_TYPE_A (1u) // Constant for record type in DNSResolve. Indicates an A
                        (standard address) record.

```

**Description**

Constant for record type in DNSResolve (see page 180). Indicates an A (standard address) record.

### 10.4.1.7 DNS\_TYPE\_MX Macro

**File**

DNS.h

**C**

```
#define DNS_TYPE_MX (15u) // Constant for record type in DNSResolve. Indicates an
MX (mail exchanger) record.
```

**Description**

Constant for record type in DNSResolve (see page 180). Indicates an MX (mail exchanger) record.

## 10.4.2 DNS Internal Members

**Functions**

	Name	Description
	DNSPutString (see page 183)	Writes a string to the DNS socket.
	DNSPutROMString (see page 183)	Writes a ROM string to the DNS socket.
	DNSDiscardName (see page 186)	Reads a name string or string pointer from the DNS socket and discards it.

**Macros**

	Name	Description
	DNS_PORT (see page 184)	Default port for DNS resolutions
	DNS_TIMEOUT (see page 184)	Elapsed time after which a DNS resolution is considered to have timed out

**Module**

DNS Client (see page 178)

**Structures**

	Name	Description
	DNS_HEADER (see page 186)	Structure for the DNS header

**Variables**

	Name	Description
	DNSHostName (see page 184)	Host name in RAM to look up
	DNSHostNameROM (see page 184)	Host name in ROM to look up
	Flags (see page 185)	Stores various flags for the UDP module
	RecordType (see page 185)	Record type being queried

	ResolvedInfo ( <a href="#">see page 185</a> )	Node information about the resolved node
	smDNS ( <a href="#">see page 185</a> )	State machine for a DNS query

**Description**

The following functions and variables are designated as internal to the DNS module.

## 10.4.2.1 DNSPutString Function

**File**

DNS.c

**C**

```
static void DNSPutString(
    BYTE* String
);
```

**Returns**

None

**Description**

This function writes a string to the DNS socket, ensuring that it is properly formatted.

**Preconditions**

UDP socket is obtained and ready for writing.

**Parameters**

Parameters	Description
String	the string to write to the UDP socket.

**Section**

Function Prototypes

## 10.4.2.2 DNSPutROMString Function

**File**

DNS.c

**C**

```
static void DNSPutROMString(
    ROM BYTE* String
);
```

**Returns**

None

**Description**

This function writes a string to the DNS socket, ensuring that it is properly formatted.

**Remarks**

This function is aliased to DNSPutString ([see page 183](#)) on non-PIC18 platforms.

**Preconditions**

UDP socket is obtained and ready for writing.

**Parameters**

Parameters	Description
String	the string to write to the UDP socket.

### 10.4.2.3 DNS\_PORT Macro

**File**

DNS.c

**C**

```
#define DNS_PORT 53u // Default port for DNS resolutions
```

**Description**

Default port for DNS resolutions

### 10.4.2.4 DNS\_TIMEOUT Macro

**File**

DNS.c

**C**

```
#define DNS_TIMEOUT (TICK_SECOND*1) // Elapsed time after which a DNS resolution is  
considered to have timed out
```

**Description**

Elapsed time after which a DNS resolution is considered to have timed out

### 10.4.2.5 DNSHostName Variable

**File**

DNS.c

**C**

```
BYTE * DNSHostName;
```

**Description**

Host name in RAM to look up

### 10.4.2.6 DNSHostNameROM Variable

**File**

DNS.c

**C**

```
ROM BYTE * DNSHostNameROM;
```

**Description**

Host name in ROM to look up

## 10.4.2.7 Flags Variable

### File

UDP.c

### C

```
struct {
    unsigned char bFirstRead : 1;
    unsigned char bWasDiscarded : 1;
} Flags;
```

### Members

Members	Description
unsigned char bFirstRead : 1;	No data has been read from this segment yet
unsigned char bWasDiscarded : 1;	The data in this segment has been discarded

### Description

Stores various flags for the UDP module

## 10.4.2.8 RecordType Variable

### File

DNS.c

### C

```
BYTE RecordType;
```

### Description

Record type being queried

## 10.4.2.9 ResolvedInfo Variable

### File

DNS.c

### C

```
NODE_INFO ResolvedInfo;
```

### Description

Node information about the resolved node

## 10.4.2.10 smDNS Variable

### File

DNS.c

### C

```
enum {
    DNS_START = 0,
    DNS_ARP_START_RESOLVE,
    DNS_ARP_RESOLVE,
    DNS_OPEN_SOCKET,
    DNS_QUERY,
```

```

    DNS_GET_RESULT,
    DNS_FAIL,
    DNS_DONE
} smDNS;

```

**Members**

Members	Description
DNS_START = 0	Initial state to reset client state variables
DNS_ARP_START_RESOLVE	Send ARP resolution of DNS server or gateway MAC address
DNS_ARP_RESOLVE	Wait for response to ARP request
DNS_OPEN_SOCKET	Open UDP socket
DNS_QUERY	Send DNS query to DNS server
DNS_GET_RESULT	Wait for response from DNS server
DNS_FAIL	ARP or DNS server not responding
DNS_DONE	DNS query is finished

**Description**

State machine for a DNS query

## 10.4.2.11 DNS\_HEADER Structure

**File**

DNS.c

**C**

```

typedef struct {
    WORD_VAL TransactionID;
    WORD_VAL Flags;
    WORD_VAL Questions;
    WORD_VAL Answers;
    WORD_VAL AuthoritativeRecords;
    WORD_VAL AdditionalRecords;
} DNS_HEADER;

```

**Description**

Structure for the DNS header

## 10.4.2.12 DNSDiscardName Function

**File**

DNS.c

**C**

```

static void DNSDiscardName();

```

**Returns**

None

**Description**

This function reads a name string from the DNS socket. Each string consists of a series of labels. Each label consists of a length prefix byte, followed by the label bytes. At the end of the string, a zero length label is found as termination. If name compression is used, this function will automatically detect the pointer and discard it.

**Preconditions**

UDP socket is obtained and ready for reading a DNS name

## 10.5 Dynamic DNS Client

The Dynamic DNS Client module provides a method for updating a dynamic IP address to a public DDNS service. These services can be used to provide DNS hostname mapping to devices that behind routers, firewalls, and/or on networks that dynamically assign IP addresses.

Note that this only solves one of the two problems for communicating to devices on local subnets from the Internet. While Dynamic DNS can help to locate the device, the router or firewall it sits behind must still properly forward the incoming connection request. This generally requires port forwarding to be configured for the router behind which the device is located.

The Dynamic DNS client supports the popular interface used by [DynDNS.org](http://DynDNS.org), [No-IP.com](http://No-IP.com), and [DNS-O-Matic.com](http://DNS-O-Matic.com).

**IMPORTANT:** The dynamic DNS services stipulate that updates should be made no more frequently than 10 minutes, and only when the IP address has changed. Updates made more often than that are considered abusive, and may eventually cause your account to be disabled. Production devices that get rebooted frequently may need to store the last known IP in non-volatile memory. You also should not enable this module while testing the rest of your application.

### 10.5.1 Dynamic DNS Public Members

#### Enumerations

	Name	Description
	DDNS_SERVICES ( <a href="#">see page 189</a> )	Dynamic DNS Services. Must support the DynDNS ( <a href="#">see page 187</a> ) API (Auxlang) and correspond to <code>ddnsServiceHosts</code> ( <a href="#">see page 194</a> ) and <code>ddnsServicePorts</code> ( <a href="#">see page 194</a> ) in DynDNS.c.
	DDNS_STATUS ( <a href="#">see page 189</a> )	Status message for DynDNS ( <a href="#">see page 187</a> ) client. GOOD and NOCHG are ok, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

#### Functions

	Name	Description
	DDNSForceUpdate ( <a href="#">see page 190</a> )	Forces an immediate DDNS update
	DDNSGetLastIP ( <a href="#">see page 191</a> )	Returns the last known external IP address of the device.
	DDNSGetLastStatus ( <a href="#">see page 191</a> )	Returns the status of the most recent update.
	DDNSSetService ( <a href="#">see page 191</a> )	Selects a pre-configured Dynamic DNS service

#### Module

Dynamic DNS Client ([see page 187](#))

#### Structures

	Name	Description
	DDNS_POINTERS ( <a href="#">see page 188</a> )	Configuration parameters for the Dynamic DNS Client

#### Variables

	Name	Description
	DDNSClient ( <a href="#">see page 190</a> )	Configuration parameters for the module

**Description**

These functions and variables are meant to be called by your stack application.

**10.5.1.1 DDNS\_POINTERS Structure****File**

DynDNS.h

**C**

```
typedef struct {
    union {
        BYTE * szRAM;
        ROM BYTE * szROM;
    } CheckIPServer;
    WORD CheckIPPort;
    union {
        BYTE * szRAM;
        ROM BYTE * szROM;
    } UpdateServer;
    WORD UpdatePort;
    union {
        BYTE * szRAM;
        ROM BYTE * szROM;
    } Username;
    union {
        BYTE * szRAM;
        ROM BYTE * szROM;
    } Password;
    union {
        BYTE * szRAM;
        ROM BYTE * szROM;
    } Host;
    struct {
        unsigned char CheckIPServer : 1;
        unsigned char UpdateServer : 1;
        unsigned char Username : 1;
        unsigned char Password : 1;
        unsigned char Host : 1;
    } ROMPointers;
} DDNS_POINTERS;
```

**Description**

This structure of pointers configures the Dynamic DNS Client. Initially, all pointers will be null and the client will be disabled. Set `DDNSClient` (see page 190).`[field name].szRAM` to use a string stored in RAM, or `DDNSClient` (see page 190).`[field name].szROM` to use a string stored in ROM. (Where `[field name]` is one of the parameters below.)

If a ROM string is specified, `DDNSClient.ROMPointers.[field name]` must also be set to 1 to indicate that this field should be retrieved from ROM instead of RAM.

**Parameters**

Parameters	Description
CheckIPServer	The server used to determine the external IP address
CheckIPPort	Port on the above server to connect (see page 165) to
UpdateServer	The server where updates should be posted
UpdatePort	Port on the above server to connect (see page 165) to
Username	The user name for the dynamic DNS server
Password	The password to supply when making updates
Host	The host name you wish to update
ROMPointers	Indicates which parameters to read from ROM instead of RAM.

## 10.5.1.2 DDNS\_SERVICES Enumeration

**File**

DynDNS.h

**C**

```
typedef enum {
    DYNDNS_ORG = 0u,
    NO_IP_COM,
    DNSOMATIC_COM
} DDNS_SERVICES;
```

**Members**

Members	Description
DYNDNS_ORG = 0u	www.dyndns.org
NO_IP_COM	www.no-ip.com
DNSOMATIC_COM	www.dnsomatic.com

**Description**

Dynamic DNS Services. Must support the DynDNS (see page 187) API (Auxlang) and correspond to ddnsServiceHosts (see page 194) and ddnsServicePorts (see page 194) in DynDNS.c.

## 10.5.1.3 DDNS\_STATUS Enumeration

**File**

DynDNS.h

**C**

```
typedef enum {
    DDNS_STATUS_GOOD = 0u,
    DDNS_STATUS_NOCHG,
    DDNS_STATUS_ABUSE,
    DDNS_STATUS_BADSYS,
    DDNS_STATUS_BADAGENT,
    DDNS_STATUS_BDAUTH,
    DDNS_STATUS_NOT_DONATOR,
    DDNS_STATUS_NOT_FQDN,
    DDNS_STATUS_NOHOST,
    DDNS_STATUS_NOT_YOURS,
    DDNS_STATUS_NUMHOST,
    DDNS_STATUS_DNSERR,
    DDNS_STATUS_911,
    DDNS_STATUS_UPDATE_ERROR,
    DDNS_STATUS_UNCHANGED,
    DDNS_STATUS_CHECKIP_ERROR,
    DDNS_STATUS_INVALID,
    DDNS_STATUS_UNKNOWN
} DDNS_STATUS;
```

**Members**

Members	Description
DDNS_STATUS_GOOD = 0u	Update successful, hostname is now updated
DDNS_STATUS_NOCHG	Update changed no setting and is considered abusive. Additional 'nochg' updates will cause hostname to be blocked.
DDNS_STATUS_ABUSE	The hostname specified is blocked for update abuse.
DDNS_STATUS_BADSYS	System parameter not valid. Should be dyndns, statdns or custom.
DDNS_STATUS_BADAGENT	The user agent was blocked or not sent.

DDNS_STATUS_BDAUTH	The username and password pair do not match a real user.
DDNS_STATUS_NOT_DONATOR	An option available only to credited users (such as offline URL) was specified, but the user is not a credited user. If multiple hosts were specified, only a single !donator will be returned.
DDNS_STATUS_NOT_FQDN	The hostname specified is not a fully-qualified domain name (not in the form hostname.dyndns.org or domain.com).
DDNS_STATUS_NOHOST	The hostname specified does not exist in this user account (or is not in the service specified in the system parameter).
DDNS_STATUS_NOT_YOURS	The hostname specified does not belong to this user account.
DDNS_STATUS_NUMHOST	Too many hosts specified in an update.
DDNS_STATUS_DNSERR	Unspecified DNS error encountered by the DDNS service.
DDNS_STATUS_911	There is a problem or scheduled maintenance with the DDNS service.
DDNS_STATUS_UPDATE_ERROR	Error communicating with Update service.
DDNS_STATUS_UNCHANGED	The IP Check indicated that no update was necessary.
DDNS_STATUS_CHECKIP_ERROR	Error communicating with CheckIP service.
DDNS_STATUS_INVALID	DDNS Client data is not valid.
DDNS_STATUS_UNKNOWN	DDNS client has not yet been executed with this configuration.

**Description**

Status message for DynDNS (see page 187) client. GOOD and NOCHG are ok, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

## 10.5.1.4 DDNSClient Variable

**File**

DynDNS.c

**C**

```
DDNS_POINTERS DDNSClient;
```

**Description**

Configuration parameters for the module

## 10.5.1.5 DDNSForceUpdate Function

**File**

DynDNS.h

**C**

```
void DDNSForceUpdate();
```

**Returns**

None

**Description**

This function forces the DDNS Client to execute a full update immediately. Any error message is cleared, and the update will be executed whether the IP address has changed or not. Call this function every time the DDNSClient (see page 190) parameters have been modified.

**Preconditions**

DDNSInit (see page 192) must have been called.

## 10.5.1.6 DDNSGetLastIP Function

### File

DynDNS.h

### C

```
IP_ADDR DDNSGetLastIP();
```

### Returns

The last known external IP address of the device.

### Description

This function returns the last known external IP address of the device.

### Preconditions

None

## 10.5.1.7 DDNSGetLastStatus Function

### File

DynDNS.h

### C

```
DDNS_STATUS DDNSGetLastStatus();
```

### Returns

DDNS\_STATUS (see page 189) indicating the status code for the most recent update.

### Description

This function returns the status of the most recent update. See the DDNS\_STATUS (see page 189) enumeration for possible codes.

### Preconditions

None

## 10.5.1.8 DDNSSetService Function

### File

DynDNS.h

### C

```
void DDNSSetService(  
    DDNS_SERVICES svc  
);
```

### Returns

None

### Description

This function selects a Dynamic DNS service based on parameters configured in ddnsServiceHosts (see page 194) and ddnsServicePorts (see page 194). These arrays must match the DDNS\_SERVICES (see page 189) enumeration.

### Preconditions

None

**Parameters**

Parameters	Description
svc	one of the DDNS_SERVICES (see page 189) elements to indicate the selected service

---

## 10.5.2 Dynamic DNS Stack Members

**Functions**

	Name	Description
≡◆	DDNSInit (see page 192)	Initializes the Dynamic DNS module.
≡◆	DDNSTask (see page 192)	Dynamic DNS client task/state machine.

**Module**

Dynamic DNS Client (see page 187)

**Description**

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.5.2.1 DDNSInit Function

**File**

DynDNS.h

**C**

```
void DDNSInit();
```

**Returns**

None

**Description**

This function initializes the Dynamic DNS client. It clears the DDNSClient (see page 190) pointers structure, and tells the module to attempt the first update after 15 seconds have elapsed (so as to allow the DHCP configuration to stabilize).

**Remarks**

This function is called only one during lifetime of the application.

**Preconditions**

None

### 10.5.2.2 DDNSTask Function

**File**

DynDNS.h

**C**

```
void DDNSTask();
```

**Returns**

None

**Description**

This function performs the background tasks of the Dynamic DNS Client. Once the DDNSPointers structure is configured, this task attempt to update the Dynamic DNS hostname on a periodic schedule.

The task first accesses the CheckIP server to determine the device's current external IP address. If the IP address has changed, it issues an update command to the dynamic DNS service to propagate the change. This sequence executes whenever dwUpdateAt (see page 194) elapses, which by default is every 10 minutes, or when an update is forced.

**Remarks**

This function acts as a task (similar to one in an RTOS). It performs its task in a co-operative manner, and the main application must call this function periodically to ensure that its tasks get executed in a timely fashion.

**Preconditions**

DDNSInit (see page 192)() has been called.

**Section**

Function Prototypes

---

## 10.5.3 Dynamic DNS Internal Members

**Macros**

	Name	Description
	DDNS_CHECKIP_SERVER (see page 195)	Default CheckIP server for determining current IP address
	DDNS_DEFAULT_PORT (see page 196)	Default port for CheckIP server

**Module**

Dynamic DNS Client (see page 187)

**Variables**

	Name	Description
	bForceUpdate (see page 194)	Indicates that the update should be done regardless of whether or not the IP changed. Use this flag when the user/pass/hostname have changed.
	ddnsServiceHosts (see page 194)	Host names for various Dynamic DNS services
	ddnsServicePorts (see page 194)	Port numbers for various Dynamic DNS services
	dwUpdateAt (see page 194)	Indicates when the next CheckIP should be done
	lastKnownIP (see page 194)	Last known IP address of this device
	lastStatus (see page 195)	Status response from last update
	_checkIpSrvrResponse (see page 195)	Delimiter to locate IP address from CheckIP server
	_updateIpSrvrResponse (see page 195)	Response codes from DynDNS (see page 187) Update Server

**Description**

The following functions and variables are designated as internal to the Dynamic DNS module.

### 10.5.3.1 bForceUpdate Variable

**File**

DynDNS.c

**C**

```
BOOL bForceUpdate;
```

**Description**

Indicates that the update should be done regardless of whether or not the IP changed. Use this flag when the user/pass/hostname have changed.

### 10.5.3.2 ddnsServiceHosts Variable

**File**

CustomHTTPApp.c

**C**

```
ROM char * ROM ddnsServiceHosts[];
```

**Description**

Host names for various Dynamic DNS services

### 10.5.3.3 ddnsServicePorts Variable

**File**

DynDNS.c

**C**

```
ROM WORD ddnsServicePorts[] = { 80, 80, 80, };
```

**Description**

Port numbers for various Dynamic DNS services

### 10.5.3.4 dwUpdateAt Variable

**File**

DynDNS.c

**C**

```
DWORD dwUpdateAt;
```

**Description**

Indicates when the next CheckIP should be done

### 10.5.3.5 lastKnownIP Variable

**File**

DynDNS.c

**C**

```
IP_ADDR lastKnownIP;
```

**Description**

Last known IP address of this device

### 10.5.3.6 lastStatus Variable

**File**

DynDNS.c

**C**

```
DDNS_STATUS lastStatus;
```

**Description**

Status response from last update

### 10.5.3.7 \_checkIpSrvrResponse Variable

**File**

DynDNS.c

**C**

```
ROM BYTE _checkIpSrvrResponse[] = "Address:";
```

**Description**

Delimiter to locate IP address from CheckIP server

### 10.5.3.8 \_updateIpSrvrResponse Variable

**File**

DynDNS.c

**C**

```
ROM char* _updateIpSrvrResponse[] = { "good", "nochg", "abuse", "badsys", "badagent",
    "badauth", "!donator", "notfqdn", "nohost", "!yours", "numhost", "dnserr", "911", };
```

**Description**

Response codes from DynDNS (see page 187) Update Server

### 10.5.3.9 DDNS\_CHECKIP\_SERVER Macro

**File**

DynDNS.h

**C**

```
#define DDNS_CHECKIP_SERVER (ROM BYTE*)"checkip.dyndns.com" // Default CheckIP
server for determining current IP address
```

**Description**

Default CheckIP server for determining current IP address

### 10.5.3.10 DDNS\_DEFAULT\_PORT Macro

**File**

DynDNS.h

**C**

```
#define DDNS_DEFAULT_PORT (80u) // Default port for CheckIP
server
```

**Description**

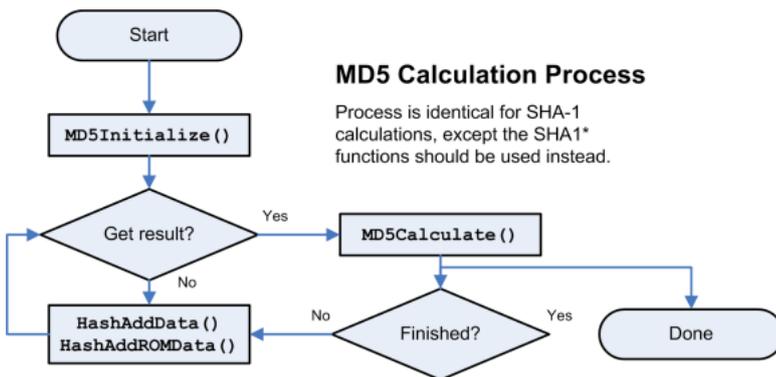
Default port for CheckIP server

## 10.6 Hashes

The Hashes module calculates MD5 and/or SHA-1 hash sums of data. Hash sums are one-way digest functions, meaning that the original message cannot be derived from the hash of the message. Collisions, while exceedingly rare, do exist. However, they are extremely difficult to create.

Hash functions are generally used for message integrity and authentication purposes. They are used extensively by encryption protocols such as SSL to verify that a message has not been tampered with during transit.

The following flow diagram demonstrates how to use this module.



To use the hash functions, first declare a HASH\_SUM (see page 200) structure and pass a pointer to it to either MD5Initialize (see page 199) or SHA1Initialize (see page 199). Then, call HashAddData (see page 197) or HashAddROMData (see page 197) as many times as are necessary to provide all the data to the hash. Call MD5Calculate (see page 198) or SHA1Calculate (see page 199) at any time to obtain the hash sum up to the current point. After calculation, continue adding data and repeating this process as many times as necessary.

### 10.6.1 Hashes Public Members

**Functions**

	Name	Description
◆	HashAddData (see page 197)	Adds data to the hash sum.
◆	HashAddROMData (see page 197)	Adds data to the hash sum.

	MD5Calculate ( <a href="#">see page 198</a> )	Calculates an MD5 hash
	MD5Initialize ( <a href="#">see page 199</a> )	Initializes a new MD5 hash.
	SHA1Calculate ( <a href="#">see page 199</a> )	Calculates a SHA-1 hash
	SHA1Initialize ( <a href="#">see page 199</a> )	Initializes a new SHA-1 hash.

**Module**Hashes ([see page 196](#))**Structures**

	Name	Description
	HASH_SUM ( <a href="#">see page 200</a> )	Context storage for a hash operation

**Description**

The following functions and variables are available to the stack application.

## 10.6.1.1 HashAddData Function

**File**

Hashes.h

**C**

```
void HashAddData(
    HASH_SUM* theSum,
    BYTE* data,
    WORD len
);
```

**Returns**

None

**Description**

Adds data to the hash sum.

**Remarks**

This function calls the appropriate hashing function based on the hash typed defined in theSum.

**Preconditions**

The hash sum has already been initialized

**Parameters**

Parameters	Description
theSum	hash context state
data	the data to be added to the hash sum
len	length of data

## 10.6.1.2 HashAddROMData Function

**File**

Hashes.h

**C**

```
void HashAddROMData(
    HASH_SUM* theSum,
    ROM BYTE* data,
    WORD len
);
```

**Returns**

None

**Description**

Adds data to the hash sum.

**Remarks**

This function calls the appropriate hashing function based on the hash typed defined in theSum.

This function is aliased to HashAddData (see page 197) on non-PIC18 platforms.

**Preconditions**

The hash sum has already been initialized

**Parameters**

Parameters	Description
theSum	hash context state
data	the data to be added to the hash sum
len	length of data

### 10.6.1.3 MD5Calculate Function

**File**

Hashes.h

**C**

```
void MD5Calculate(
    HASH_SUM* theSum,
    BYTE* result
);
```

**Returns**

None

**Description**

This function calculates the hash sum of all input data so far. It is non-destructive to the hash context, so more data may be added after this function is called.

**Preconditions**

The hash context has been properly initialized.

**Parameters**

Parameters	Description
theSum	the current hash context
result	16 byte array in which to store the resulting hash

## 10.6.1.4 MD5Initialize Function

### File

Hashes.h

### C

```
void MD5Initialize(
    HASH_SUM* theSum
);
```

### Returns

None

### Description

Initializes a new MD5 hash.

### Preconditions

None

### Parameters

Parameters	Description
theSum	pointer to the allocated HASH_SUM (see page 200) object to initialize as MD5

## 10.6.1.5 SHA1Calculate Function

### File

Hashes.h

### C

```
void SHA1Calculate(
    HASH_SUM* theSum,
    BYTE* result
);
```

### Returns

None

### Description

This function calculates the hash sum of all input data so far. It is non-destructive to the hash context, so more data may be added after this function is called.

### Preconditions

The hash context has been properly initialized.

### Parameters

Parameters	Description
theSum	the current hash context
result	20 byte array in which to store the resulting hash

## 10.6.1.6 SHA1Initialize Function

### File

Hashes.h

**C**

```
void SHA1Initialize(
    HASH_SUM* theSum
);
```

**Returns**

None

**Description**

Initializes a new SHA-1 hash.

**Preconditions**

None

**Parameters**

Parameters	Description
theSum	pointer to the allocated HASH_SUM (see page 200) object to initialize as SHA-1

**Section**

Function Prototypes

## 10.6.1.7 HASH\_SUM Structure

**File**

Hashes.h

**C**

```
typedef struct {
    DWORD h0;
    DWORD h1;
    DWORD h2;
    DWORD h3;
    DWORD h4;
    DWORD bytesSoFar;
    BYTE partialBlock[64];
    HASH_TYPE hashType;
} HASH_SUM;
```

**Members**

Members	Description
DWORD h0;	Hash state h0
DWORD h1;	Hash state h1
DWORD h2;	Hash state h2
DWORD h3;	Hash state h3
DWORD h4;	Hash state h4
DWORD bytesSoFar;	Total number of bytes hashed so far
BYTE partialBlock[64];	Beginning of next 64 byte block
HASH_TYPE hashType;	Type of hash being calculated

**Description**

Context storage for a hash operation

## 10.6.2 Hashes Stack Members

### Functions

	Name	Description
☞	MD5AddROMData (☞ see page 201)	Adds data to an MD5 hash calculation.
☞	SHA1AddROMData (☞ see page 202)	Adds data to a SHA-1 hash calculation.
☞	SHA1AddData (☞ see page 202)	Adds data to a SHA-1 hash calculation.
☞	MD5AddData (☞ see page 203)	Adds data to an MD5 hash calculation.

### Module

Hashes (☞ see page 196)

### Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.6.2.1 MD5AddROMData Function

#### File

Hashes.h

#### C

```
void MD5AddROMData(
    HASH_SUM* theSum,
    ROM_BYTE* data,
    WORD len
);
```

#### Returns

None

#### Description

Adds data to an MD5 hash calculation.

#### Remarks

This function is aliased to MD5AddData (☞ see page 203) on non-PIC18 platforms.

#### Preconditions

The hash context has already been initialized.

#### Parameters

Parameters	Description
theSum	a pointer to the hash context structure
data	the data to add to the hash
len	the length of the data to add

## 10.6.2.2 SHA1AddROMData Function

### File

Hashes.h

### C

```
void SHA1AddROMData(  
    HASH_SUM* theSum,  
    ROM BYTE* data,  
    WORD len  
);
```

### Returns

None

### Description

Adds data to a SHA-1 hash calculation.

### Remarks

This function is aliased to SHA1AddData (see page 202) on non-PIC18 platforms.

### Preconditions

The hash context has already been initialized.

### Parameters

Parameters	Description
theSum	a pointer to the hash context structure
data	the data to add to the hash
len	the length of the data to add

## 10.6.2.3 SHA1AddData Function

### File

Hashes.h

### C

```
void SHA1AddData(  
    HASH_SUM* theSum,  
    BYTE* data,  
    WORD len  
);
```

### Returns

None

### Description

Adds data to a SHA-1 hash calculation.

### Preconditions

The hash context has already been initialized.

### Parameters

Parameters	Description
theSum	a pointer to the hash context structure
data	the data to add to the hash

len	the length of the data to add
-----	-------------------------------

## 10.6.2.4 MD5AddData Function

### File

Hashes.h

### C

```
void MD5AddData(
    HASH_SUM* theSum,
    BYTE* data,
    WORD len
);
```

### Returns

None

### Description

Adds data to an MD5 hash calculation.

### Preconditions

The hash context has already been initialized.

### Parameters

Parameters	Description
theSum	a pointer to the hash context structure
data	the data to add to the hash
len	the length of the data to add

## 10.6.3 Hashes Internal Members

### Enumerations

	Name	Description
	HASH_TYPE ( <a href="#">see page 204</a> )	Type of hash being calculated

### Functions

	Name	Description
	SHA1HashBlock ( <a href="#">see page 205</a> )	Calculates the SHA-1 hash sum of a block.
	MD5HashBlock ( <a href="#">see page 206</a> )	Calculates the MD5 hash sum of a block.

### Module

Hashes ([see page 196](#))

### Variables

	Name	Description
	_MD5_k ( <a href="#">see page 204</a> )	Array of pre-defined K values for MD5
	_MD5_r ( <a href="#">see page 204</a> )	Array of pre-defined R vales for MD5
	lastBlock ( <a href="#">see page 204</a> )	Stores a copy of the last block with the required padding

**Description**

The following functions and variables are designated as internal to the Hashes (see page 196) module.

**10.6.3.1 \_MD5\_k Variable****File**

Hashes.c

**C**

```
ROM DWORD _MD5_k[64] = { 0xD76AA478, 0xE8C7B756, 0x242070DB, 0xC1BDCEEE, 0xF57C0FAF,
0x4787C62A, 0xA8304613, 0xFD469501, 0x698098D8, 0x8B44F7AF, 0xFFFF5BB1, 0x895CD7BE,
0x6B901122, 0xFD987193, 0xA679438E, 0x49B40821, 0xF61E2562, 0xC040B340, 0x265E5A51,
0xE9B6C7AA, 0xD62F105D, 0x02441453, 0xD8A1E681, 0xE7D3FBC8, 0x21E1CDE6, 0xC33707D6,
0xF4D50D87, 0x455A14ED, 0xA9E3E905, 0xFCEFA3F8, 0x676F02D9, 0x8D2A4C8A, 0xFFFA3942,
0x8771F681, 0x6D9D6122, 0xFDE5380C, 0xA4BEEA44, 0x4BDECFA9, 0xF6BB4B60, 0xBEBFBC70,
0x289B7EC6, 0xEAA127FA, 0xD4EF3085, 0x04881D05, 0xD9D4D039, 0xE6DB99E5, 0x1FA27CF8,
0xC4AC5665, 0xF4292244, 0x432AFF97, 0xAB9423A7, 0xFC93A039, 0x655B59C3, 0x8F0CCC92,
0xFFEF47D, 0x85845DD1, 0x6FA87E4F, 0xFE2CE6E0, 0xA3014314, 0x4E0811A1, 0xF7537E82,
0xBD3AF235, 0x2AD7D2BB, 0xEB86D391 };
```

**Description**

Array of pre-defined K values for MD5

**10.6.3.2 \_MD5\_r Variable****File**

Hashes.c

**C**

```
ROM BYTE _MD5_r[64] = {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 5, 9,
14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16,
23, 4, 11, 16, 23, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21};
```

**Description**

Array of pre-defined R values for MD5

**10.6.3.3 lastBlock Variable****File**

Hashes.c

**C**

```
BYTE lastBlock[64];
```

**Description**

Stores a copy of the last block with the required padding

**10.6.3.4 HASH\_TYPE Enumeration****File**

Hashes.h

**C**

```
typedef enum {
    HASH_MD5 = 0u,
    HASH_SHA1
} HASH_TYPE;
```

**Members**

Members	Description
HASH_MD5 = 0u	MD5 is being calculated
HASH_SHA1	SHA-1 is being calculated

**Description**

Type of hash being calculated

## 10.6.3.5 SHA1HashBlock Function

**File**

Hashes.c

**C**

```
static void SHA1HashBlock(
    BYTE* data,
    DWORD* h0,
    DWORD* h1,
    DWORD* h2,
    DWORD* h3,
    DWORD* h4
);
```

**Returns**

None

**Description**

This function calculates the SHA-1 hash sum over a block and updates the values of h0-h3 with the next context.

**Internal**

TODO convert data to a DWORD\* or read from the pointer using byte accesses only to avoid any accidental alignment errors

**Preconditions**

The data pointer must be WORD aligned on 16-bit parts and DWORD aligned on 32-bit PICs. If alignment is not correct, a memory alignment exception will occur.

**Parameters**

Parameters	Description
data	The block of 64 bytes to hash
h0	the current hash context h0 value
h1	the current hash context h1 value
h2	the current hash context h2 value
h3	the current hash context h3 value
h4	the current hash context h4 value

**Section**

Functions and variables required for SHA-1

## 10.6.3.6 MD5HashBlock Function

### File

Hashes.c

### C

```
static void MD5HashBlock(
    BYTE* data,
    DWORD* h0,
    DWORD* h1,
    DWORD* h2,
    DWORD* h3
);
```

### Returns

None

### Description

This function calculates the MD5 hash sum over a block and updates the values of h0-h3 with the next context.

### Internal

TODO convert data to a DWORD\* or read from the pointer using byte accesses only to avoid any accidental alignment errors

### Preconditions

The data pointer must be WORD aligned on 16-bit parts and DWORD aligned on 32-bit PICs. If alignment is not correct, a memory alignment exception will occur.

### Parameters

Parameters	Description
data	The block of 64 bytes to hash
h0	the current hash context h0 value
h1	the current hash context h1 value
h2	the current hash context h2 value
h3	the current hash context h3 value

## 10.7 Helpers

### Functions

	Name	Description
	LFSRRand ( <a href="#">see page 222</a> )	Returns a pseudo-random 16-bit unsigned integer in the range from 0 to 65535 (0x0000 to 0xFFFF).
	LFSRSeedRand ( <a href="#">see page 223</a> )	Seeds the LFSR random number generator invoked by the LFSRRand ( <a href="#">see page 222</a> ()) function. The prior seed is returned.

### Variables

	Name	Description
	dwLFSRRandSeed ( <a href="#">see page 224</a> )	Default Random Number Generator seed. 0x41FE9F9E corresponds to calling LFSRSeedRand ( <a href="#">see page 223</a> )(1)

### Description

This module contains several helper functions used throughout the TCP/IP Stack. Some of these duplicate functionality already implemented in the compiler's default libraries. In those cases, the compiler's version is used and the stack's version

is omitted.

## 10.7.1 Helpers Public Members

### Functions

	Name	Description
⇒	Base64Decode (see page 208)	Decodes a Base-64 array to its literal representation.
⇒	Base64Encode (see page 208)	Encodes a binary array to Base-64.
⇒	btohexa_high (see page 209)	Converts the upper nibble of a binary value to a hexadecimal ASCII byte.
⇒	btohexa_low (see page 209)	Converts the lower nibble of a binary value to a hexadecimal ASCII byte.
⇒	CalcIPBufferChecksum (see page 210)	Calculates an IP checksum in the MAC buffer itself.
⇒	CalcIPChecksum (see page 210)	Calculates an IP checksum value.
⇒	ExtractURLFields (see page 211)	Extracts all parameters from an URL string (ex: "http://admin:passwd@www.microchip.com:8080/myfile.gif" is split into {PROTOCOL_HTTP, "admin", "passwd", "www.microchip.com", 8080, "/myfile.gif"}).
⇒	FormatNetBIOSName (see page 214)	Formats a string to a valid NetBIOS name.
⇒	GenerateRandomDWORD (see page 214)	Generates a random DWORD.
⇒	hexatob (see page 215)	Converts a hex string to a single byte.
⇒	leftRotateDWORD (see page 215)	Left-rotates a DWORD.
⇒	Replace (see page 216)	Replaces all instances of a particular substring with a new string
⇒	ROMStringToIPAddress (see page 217)	Converts a string to an IP address
⇒	stricmpgm2ram (see page 218)	Case-insensitive comparison of a string in RAM to a string in ROM.
⇒	StringToIPAddress (see page 218)	Converts a string to an IP address
⇒	strupr (see page 219)	Converts a string to uppercase.
⇒	strnchr (see page 219)	Searches a string up to a specified number of characters for a specific character.
⇒	swapl (see page 220)	Swaps the endian-ness of a DWORD.
⇒	swaps (see page 220)	Swaps the endian-ness of a WORD.
⇒	uitoa (see page 221)	Converts an unsigned integer to a decimal string.
⇒	ultoa (see page 221)	Converts an unsigned integer to a decimal string.
⇒	UnencodeURL (see page 222)	Decodes a URL-encoded string.

### Macros

	Name	Description
⇒	leftRotateDWORD (see page 216)	Rotations are more efficient in C30 and C32
⇒	ROMStringToIPAddress (see page 218)	Non-ROM variant for C30 and C32

**Module**

Helpers (see page 206)

**Description**

The following functions and variables are available to the stack application.

## 10.7.1.1 Base64Decode Function

**File**

Helpers.h

**C**

```
WORD Base64Decode(
    BYTE* cSourceData,
    WORD wSourceLen,
    BYTE* cDestData,
    WORD wDestLen
);
```

**Returns**

Number of decoded bytes written to cDestData.

**Description**

Decodes a Base-64 array to its literal representation.

**Remarks**

This function is binary safe and will ignore invalid characters (CR, LF, etc). If cSourceData is equal to cDestData, the data will be converted in-place. If cSourceData is not equal to cDestData, but the regions overlap, the behavior is undefined.

Decoded data is always at least 1/4 smaller than the source data.

**Preconditions**

None

**Parameters**

Parameters	Description
cSourceData	Pointer to a string of Base-64 encoded data
wSourceLen	Length of the Base-64 source data Maximum length that can be written to cDestData
cDestData	Pointer to write the decoded data

## 10.7.1.2 Base64Encode Function

**File**

Helpers.h

**C**

```
WORD Base64Encode(
    BYTE* cSourceData,
    WORD wSourceLen,
    BYTE* cDestData,
    WORD wDestLen
);
```

**Returns**

Number of encoded bytes written to cDestData. This will always be a multiple of 4.

**Description**

Encodes a binary array to Base-64.

**Remarks**

Encoding cannot be performed in-place. If cSourceData overlaps with cDestData, the behavior is undefined.

Encoded data is always at least 1/3 larger than the source data. It may be 1 or 2 bytes larger than that.

**Preconditions**

None

**Parameters**

Parameters	Description
cSourceData	Pointer to a string of binary data
wSourceLen	Length of the binary source data Maximum length that can be written to cDestData
cDestData	Pointer to write the Base-64 encoded data

### 10.7.1.3 btohexa\_high Function

**File**

Helpers.h

**C**

```
BYTE btohexa_high(
    BYTE b
);
```

**Returns**

The upper hexadecimal ASCII byte '0'-'9' or 'A'-'F'.

**Description**

Converts the upper nibble of a binary value to a hexadecimal ASCII byte. For example, btohexa\_high(0xAE) will return 'A'.

**Preconditions**

None

**Parameters**

Parameters	Description
b	the byte to convert

### 10.7.1.4 btohexa\_low Function

**File**

Helpers.h

**C**

```
BYTE btohexa_low(
    BYTE b
);
```

**Returns**

The lower hexadecimal ASCII byte '0'-'9' or 'A'-'F'.

**Description**

Converts the lower nibble of a binary value to a hexadecimal ASCII byte. For example, `btohexa_high` (see page 209) (0xAE) will return 'E'.

**Preconditions**

None

**Parameters**

Parameters	Description
b	the byte to convert

## 10.7.1.5 CalcIPBufferChecksum Function

**File**

Helpers.h

**C**

```
WORD CalcIPBufferChecksum(
    WORD len
);
```

**Returns**

The calculated checksum.

**Description**

This function calculates an IP checksum over an array of input data existing in the MAC buffer. The checksum is the 16-bit one's complement of one's complement sum of all words in the data (with zero-padding if an odd number of bytes are summed). This checksum is defined in RFC 793.

**Remarks**

All Microchip MACs should perform this function in hardware.

**Preconditions**

TCP is initialized and the MAC buffer pointer is set to the start of the buffer.

**Parameters**

Parameters	Description
len	number of bytes to be checksummed

## 10.7.1.6 CalcIPChecksum Function

**File**

Helpers.h

**C**

```
WORD CalcIPChecksum(
    BYTE* buffer,
    WORD len
);
```

**Returns**

The calculated checksum.

**Description**

This function calculates an IP checksum over an array of input data. The checksum is the 16-bit one's complement of one's

complement sum of all words in the data (with zero-padding if an odd number of bytes are summed). This checksum is defined in RFC 793.

#### Internal

This function could be improved to do 32-bit sums on PIC32 platforms.

#### Preconditions

buffer is WORD aligned (even memory address) on 16- and 32-bit PICs.

#### Parameters

Parameters	Description
buffer	pointer to the data to be checksummed
count	number of bytes to be checksummed

## 10.7.1.7 ExtractURLFields Function

#### File

Helpers.h

#### C

```

BYTE ExtractURLFields(
    BYTE * vURL,
    PROTOCOLS * protocol,
    BYTE * vUsername,
    WORD * wUsernameLen,
    BYTE * vPassword,
    WORD * wPasswordLen,
    BYTE * vHostname,
    WORD * wHostnameLen,
    WORD * wPort,
    BYTE * vFilePath,
    WORD * wFilePathLen
);

```

#### Returns

Zero on success. Nonzero indicates an error code. If a nonzero error code is returned, none of the returned buffers or pointer values should be treated as valid, but some of them may have been written to. The following are all possible return values.

0	No error
1	Protocol unknown (additional code needs to be added to ExtractURLFields()) and the PROTOCOLS enum needs to be updated if you want to decode URLs of this protocol type.
2	URL malformed. Illegal or unknown URL format encountered.
3	Buffer too small. One of the input buffer sizes is too small to contain the URL parameter.

#### Description

Extracts all parameters from an URL string (ex: "http://admin:passwd@www.microchip.com:8080/myfile.gif" is split into {PROTOCOL\_HTTP, "admin", "passwd", "www.microchip.com", 8080, "/myfile.gif"}).

The URL string can be null terminated, or alternatively could be terminated by a carriage return or line feed.

If the protocol is unrecognized or the protocol is recognized but the URL is malformed, than an error is safely returned. For more information on URL/URI interpretation see RFC 2396.

#### Preconditions

This function is commented out by default to save code space because it is not used by any current stack features. However, if you want to use it, go ahead and uncomment it. It has been tested, so it (should) work correctly.

**Parameters**

Parameters	Description
vURL	<p>Pointer to null terminated URL to decode and extract from. This parameter is required and needs to have the minimum RFC 1738 components in it (protocol and hostname).</p>
protocol	<p>Optional pointer to a PROTOCOLS enum to retrieve the decoded protocol type. If this parameter is unneeded, specify a NULL pointer. The protocol is a required part of the URL, so it must always be present. The protocol also determines what scheme all other parameters are decoded using, so the function will fail if an unrecognized protocol is provided. The PROTOCOLS enum members show all of the currently supported protocols for this function.</p> <p>For the example URL provided in the function description, PROTOCOL_HTTP would be returned for this field.</p>
vUsername	<p>Optional pointer to a buffer to write the decoded username portion of the URL. If the URL does not contain a username or a NULL pointer is supplied, then this field is ignored.</p> <p>For the example URL provided in the function description, "admin" would be returned for this field.</p>
wUsernameLen	<p>On call: Optional pointer to a WORD specifying the maximum length of the vUsername buffer, including the null terminator character.</p> <p>Upon return: If wUsernameLen and vUsername are non-NULL, the *wUsernameLen WORD is updated with the actual number of characters written to the vUsername buffer, including the null terminator character. If vUsername is NULL but wUsernameLen is non-NULL, then no characters are copied, but *wUsernameLen will return the number of characters required to fit the full username string. If wUsernameLen is NULL, then the username field in the URL, if present, is ignored and the vUsername pointer is not used.</p> <p>If zero characters were written, this indicates that the URL did not contain a username field. If one character was written, this indicates that a username field was present, but was a zero character string (ex: "").</p> <p>For the example URL provided in the function description, 6 (0x0006) would be returned for this field.</p>
vPassword	<p>Optional pointer to a buffer to write the decoded password portion of the URL. If the URL does not contain a password or a NULL pointer is supplied, then this field is ignored.</p> <p>For the example URL provided in the function description, "passwd" would be returned for this field.</p>

<p>wPasswordLen</p>	<p>On call: Optional pointer to a WORD specifying the maximum length of the vPassword buffer, including the null terminator character.</p> <p>Upon return: If wPasswordLen and vPassword are non-NULL, the *wPasswordLen WORD is updated with the actual number of characters written to the vPassword buffer, including the null terminator character. If vPassword is NULL but wPasswordLen is non-NULL, then no characters are copied, but *wPasswordLen will return the number of characters required to fit the full password string. If wPasswordLen is NULL, then the password field in the URL, if present, is ignored and the vPassword pointer is not used.</p> <p>If zero characters were written, this indicates that the URL did not contain a password field. If one character was written, this indicates that a password field was present, but was a zero character string (ex: "").</p> <p>For the example URL provided in the function description, 7 (0x0007) would be returned for this field.</p>
<p>vHostname</p>	<p>Optional pointer to a buffer to write the decoded hostname portion of the URL. All Internet URLs must contain a hostname or IP address, however, if a NULL pointer is supplied, then this field is ignored.</p> <p>For the example URL provided in the function description, "www.microchip.com" would be returned for this field. If the URL was "http://192.168.0.1", then this field would be returned as "192.168.0.1". The IP address would not be decoded to a DWORD (use the StringToIPAddress (see page 218)() helper function to do this).</p>
<p>wHostnameLen</p>	<p>On call: Optional pointer to a WORD specifying the maximum length of the vHostname buffer, including the null terminator character.</p> <p>Upon return: If wHostnameLen and vHostname are non-NULL, the *wHostnameLen WORD is updated with the actual number of characters written to the vHostname buffer, including the null terminator character. If vHostname is NULL but wHostnameLen is non-NULL, then no characters are copied, but *wHostnameLen will return the number of characters required to fit the full hostname string. If wHostnameLen is NULL, then the hostname field in the URL, is ignored and the vHostname pointer is not used.</p> <p>For the example URL provided in the function description, 18 (0x0012) would be returned for this field. If the URL was "http://192.168.0.1", then this field would be returned as 12 (0x000C).</p>
<p>wPort</p>	<p>Optional pointer to a WORD specifying the TCP or UDP port that the server is listening on. If the port field is absent from the URL, then this parameter will specify the default port for the protocol. For example, "http://www.microchip.com" would result in 80 being return as the specified port.</p> <p>If the wPort pointer is NULL, then the port field in the URL is ignored, if present.</p>
<p>vFilePath</p>	<p>Optional pointer to a buffer to write the decoded file path portion of the URL. If a NULL pointer is supplied, then this field is ignored. If a file path is not present in the URL, then "/" will be returned in this field.</p> <p>For the example URL provided in the function description, "/myfile.gif" would be returned for this field.</p>

wFilePathLen	<p>On call: Optional pointer to a WORD specifying the maximum length of the vFilePath buffer, including the null terminator character.</p> <p>Upon return: If wFilePathLen and vFilePath are non-NULL, the *wFilePathLen WORD is updated with the actual number of characters written to the vFilePath buffer, including the null terminator character. If vFilePath is NULL but wFilePathLen is non-NULL, then no characters are copied, but *wFilePathLen will return the number of characters required to fit the full file path string. If wFilePathLen is NULL, then the file path field in the URL, if present, is ignored and the vFilePath pointer is not used.</p> <p>This function always returns "/" if no file path is present, so *wFilePathLen will also be at least 2 characters ('/' and null terminator) if the pointer is non-NULL.</p> <p>For the example URL provided in the function description, 12 (0x000C) would be returned for this field.</p>
--------------	--

### 10.7.1.8 FormatNetBIOSName Function

**File**

Helpers.h

**C**

```
void FormatNetBIOSName(
    BYTE Name[16]
);
```

**Returns**

None

**Description**

This function formats a string to a valid NetBIOS name. Names will be exactly 16 characters, as defined by the NetBIOS spec. The 16th character will be a 0x00 byte, while the other 15 will be the provided string, padded with spaces as necessary.

**Preconditions**

None

**Parameters**

Parameters	Description
Name	the string to format as a NetBIOS name. This parameter must have at least 16 bytes allocated.

### 10.7.1.9 GenerateRandomDWORD Function

**File**

Helpers.h

**C**

```
DWORD GenerateRandomDWORD();
```

**Side Effects**

This function uses the A/D converter (and so you must disable interrupts if you use the A/D converted in your ISR). The LFSRRand (see page 222)() function will be reseeded, and Timer0 (PIC18) and Timer1 (PIC24, dsPIC, and PIC32) will be used. TMR#H:TMR#L will have a new value. Note that this is the same timer used by the Tick module.

**Returns**

Random 32-bit number.

**Description**

This function generates a random 32-bit integer. It collects randomness by comparing the A/D converter's internal R/C oscillator clock with our main system clock. By passing collected entropy to the LFSRSeedRand (see page 223)/LFSRRand (see page 222)() functions, the output is normalized (deskewed) in the hopes of meeting statistical randomness tests.

**Remarks**

This function times out after 1 second of attempting to generate the random DWORD. In such a case, the output may not be truly random. Typically, this function executes in around 500,000 instruction cycles.

The intent of this function is to produce statistically random and cryptographically secure random number. Whether or not this is true on all (or any) devices/voltages/temperatures is not tested.

**Preconditions**

None

## 10.7.1.10 hexatob Function

**File**

Helpers.h

**C**

```
BYTE hexatob(
    WORD_VAL AsciiChars
);
```

**Returns**

Resulting packed byte 0x00 - 0xFF.

**Description**

Converts a two-character ASCII hex string to a single packed byte.

**Preconditions**

None

**Parameters**

Parameters	Description
AsciiChars	WORD_VAL where .v[0] is the ASCII value for the lower nibble and .v[1] is the ASCII value for the upper nibble. Each must range from '0'-'9', 'A'-'F', or 'a'-'f'.

## 10.7.1.11 leftRotateDWORD Function

**File**

Helpers.h

**C**

```
DWORD leftRotateDWORD(
    DWORD val,
    BYTE bits
);
```

**Returns**

Rotated DWORD value.

**Description**

This function rotates the bits in a 32-bit DWORD left by a specific number of bits.

**Remarks**

This function is only implemented on 8-bit platforms for now. The 8-bit compilers generate excessive code for this function, while C30 and C32 already generate compact code. Those compilers are served by a macro defined in Helpers.h.

**Preconditions**

None

**Parameters**

Parameters	Description
val	the DWORD to be rotated
bits	the number of bits by which to shift

## 10.7.1.12 leftRotateDWORD Macro

**File**

Helpers.h

**C**

```
#define leftRotateDWORD(x, n) (((x) << (n)) | ((x) >> (32-(n))))
```

**Description**

Rotations are more efficient in C30 and C32

## 10.7.1.13 Replace Function

**File**

Helpers.h

**C**

```
SHORT Replace(
    BYTE * vExpression,
    ROM BYTE * vFind,
    ROM BYTE * vReplacement,
    WORD wMaxLen,
    BOOL bSearchCaseInsensitive
);
```

**Returns**

If zero or greater, indicates the count of how many replacements were made. If less than zero (negative result), indicates that wMaxLen was too small to make the necessary replacements. In this case, no replacements were made.

**Description**

Searches a string (vExpression) and replaces all instances of a particular substring (vFind) with a new string (vReplacement). The start offset to being searching and a maximum number of replacements can be specified. The search can be performed in a case sensitive or case insensitive manner.

**Remarks**

If the replacement string length is shorter than or equal to the search string length and the search string occurs in multiple overlapping locations (ex: expression is "aaa", find is "aa", and replacement is "bb") then the first find match occurring when searching from left to right will be replaced. (ex: output expression will be "bba").

However, if the replacement string length is longer than the search string length, the search will occur starting from the end

of the string and proceed to the beginning (right to left searching). In this case if the expression was "aaa", find was "aa", and replacement was "bbb", then the final output expression will be "abbb".

**Preconditions**

This function is commented out by default to save code space because it is not used by any current stack features. However, if you want to use it, go ahead and uncomment it. It has been tested, so it (should) work correctly.

**Parameters**

Parameters	Description
vExpression	Null terminated string to search and make replacements within.
vFind	Null terminated string to search for.
vReplacement	Null terminated string to replace all instances of vFind with.
wMaxLen	Maximum length of the output vExpression string if string expansion is going to occur (replacement length is longer than find length). If the replacements will cause this maximum string length to be exceeded, then no replacements will be made and a negative result will be returned, indicating failure. If the replacement length is shorter or equal to the search length, then this parameter is ignored.
bSearchCaseInsensitive	Boolean indicating if the search should be performed in a case insensitive manner. Specify TRUE for case insensitive searches (slower) or FALSE for case sensitive searching (faster).

## 10.7.1.14 ROMStringToIPAddress Function

**File**

Helpers.h

**C**

```

BOOL ROMStringToIPAddress (
    ROM BYTE* str,
    IP_ADDR* IPAddress
);
    
```

**Description**

This function parses a dotted-quad decimal IP address string into an IP\_ADDR struct. The output result is big-endian.

**Remarks**

This function is aliased to StringToIPAddress (see page 218) on non-PIC18 platforms.

**Preconditions**

None

**Parameters**

Parameters	Description
str	Pointer to a dotted-quad IP address string
IPAddress	Pointer to IP_ADDR in which to store the result

**Return Values**

Return Values	Description
TRUE	an IP address was successfully decoded
FALSE	no IP address could be found, or the format was incorrect

## 10.7.1.15 ROMStringToIPAddress Macro

### File

Helpers.h

### C

```
#define ROMStringToIPAddress(a,b) StringToIPAddress((BYTE*)a,b)
```

### Description

Non-ROM variant for C30 and C32

## 10.7.1.16 stricmpgm2ram Function

### File

Helpers.h

### C

```
signed char stricmpgm2ram(
    BYTE* a,
    ROM BYTE* b
);
```

### Description

Performs a case-insensitive comparison of a string in RAM to a string in ROM. This function performs identically to strcmpgm2ram, except that the comparison is not case-sensitive.

### Preconditions

None

### Parameters

Parameters	Description
a	Pointer to string in RAM
b	Pointer to string in ROM

### Return Values

Return Values	Description
-1	a < b
0	a = b
1	a > b

## 10.7.1.17 StringToIPAddress Function

### File

Helpers.h

### C

```
BOOL StringToIPAddress(
    BYTE* str,
    IP_ADDR* IPAddress
);
```

### Description

This function parses a dotted-quad decimal IP address string into an IP\_ADDR struct. The output result is big-endian.

**Preconditions**

None

**Parameters**

Parameters	Description
str	Pointer to a dotted-quad IP address string
IPAddress	Pointer to IP_ADDR in which to store the result

**Return Values**

Return Values	Description
TRUE	an IP address was successfully decoded
FALSE	no IP address could be found, or the format was incorrect

## 10.7.1.18 strupr Function

**File**

Helpers.h

**C**

```
char * strupr(
    char* s
);
```

**Returns**

Pointer to the initial string.

**Description**

This function converts strings to uppercase on platforms that do not already have this function defined. All lower-case characters are converted, an characters not included in 'a'-'z' are left as-is.

**Preconditions**

None

**Parameters**

Parameters	Description
s	the null-terminated string to be converted.

## 10.7.1.19 strnchr Function

**File**

Helpers.h

**C**

```
char * strnchr(
    const char * searchString,
    size_t count,
    char c
);
```

**Returns**

Pointer to the first occurrence of the character c in the string searchString. If the character is not found or the maximum count is reached, a NULL pointer is returned.

**Description**

Searches a string up to a specified number of characters for a specific character. The string is searched forward and the first

occurrence location is returned. If the search character is not present in the string, or if the maximum character count is reached first, then a NULL pointer is returned.

#### Preconditions

None

#### Parameters

Parameters	Description
searchString	Pointer to a null terminated string to search. If count is less than the string size, then the string need not be null terminated.
count	Maximum number of characters to search before aborting.
c	Character to search for

## 10.7.1.20 swapl Function

#### File

Helpers.h

#### C

```
DWORD swapl(
    DWORD v
);
```

#### Returns

The swapped version of v.

#### Description

Swaps the endian-ness of a DWORD.

#### Preconditions

None

#### Parameters

Parameters	Description
v	the DWORD to swap

## 10.7.1.21 swaps Function

#### File

Helpers.h

#### C

```
WORD swaps(
    WORD v
);
```

#### Returns

The swapped version of v.

#### Description

Swaps the endian-ness of a WORD.

#### Preconditions

None

**Parameters**

Parameters	Description
v	the WORD to swap

## 10.7.1.22 uitoa Function

**File**

Helpers.h

**C**

```
void uitoa(
    WORD Value,
    BYTE* Buffer
);
```

**Returns**

None

**Description**

Converts a 16-bit unsigned integer to a null-terminated decimal string.

**Preconditions**

None

**Parameters**

Parameters	Description
Value	The number to be converted
Buffer	Pointer in which to store the converted string

## 10.7.1.23 ultoa Function

**File**

Helpers.h

**C**

```
void ultoa(
    DWORD Value,
    BYTE* Buffer
);
```

**Returns**

None

\*\*\*\*\*

HI-TECH PICC-18 PRO 9.63, C30 v3.25, and C32 v1.12 already have a ultoa() library function C18 already has a ultoa() function that more-or-less matches this one C32 < 1.12 and C30 < v3.25 need this function

**Description**

C32 < 1.12 and C30 < v3.25 need this 2 parameter stack implemented function

Converts a 32-bit unsigned integer to a null-terminated decimal string.

**Preconditions**

None

**Parameters**

Parameters	Description
Value	The number to be converted
Buffer	Pointer in which to store the converted string

## 10.7.1.24 UnencodeURL Function

**File**

Helpers.h

**C**

```
void UnencodeURL(
    BYTE* URL
);
```

**Returns**

None

**Description**

This function is deprecated except for use with HTTP Classic. It attempts to decode a URL encoded string, converting all hex escape sequences into a literal byte. However, it is inefficient over long strings and does not handle URL-encoded data strings ('&' and '=').

**Preconditions**

None

**Parameters**

Parameters	Description
URL	the null-terminated string to decode

## 10.7.2 Functions

**Functions**

	Name	Description
	LFSRRand ( <a href="#">see page 222</a> )	Returns a pseudo-random 16-bit unsigned integer in the range from 0 to 65535 (0x0000 to 0xFFFF).
	LFSRSeedRand ( <a href="#">see page 223</a> )	Seeds the LFSR random number generator invoked by the LFSRRand ( <a href="#">see page 222</a> )() function. The prior seed is returned.

**Module**Helpers ([see page 206](#))

### 10.7.2.1 LFSRRand Function

**File**

Helpers.h

**C**

```
WORD LFSRRand( );
```

**Side Effects**

The internal LFSR seed is updated so that the next call to LFSRRand() will return a different random number.

**Returns**

Random 16-bit unsigned integer.

**Description**

Returns a pseudo-random 16-bit unsigned integer in the range from 0 to 65535 (0x0000 to 0xFFFF). The random number is generated using a Linear Feedback Shift Register (LFSR) type pseudo-random number generator algorithm. The LFSR can be seeded by calling the LFSRSeedRand (see page 223)() function to generate the same sequence of random numbers as a prior string of calls.

The internal LFSR will repeat after  $2^{32}-1$  iterations.

**Remarks**

None

**Preconditions**

None

## 10.7.2.2 LFSRSeedRand Function

**File**

Helpers.h

**C**

```
DWORD LFSRSeedRand(
    DWORD dwSeed
);
```

**Side Effects**

None

**Returns**

The last seed in use. This can be saved and restored by a subsequent call to LFSRSeedRand() if you wish to use LFSRRand (see page 222)() in multiple contexts without disrupting the random number sequence from the alternative context. For example, if App 1 needs a given sequence of random numbers to perform a test, if you save and restore the seed in App 2, it is possible for App 2 to not disrupt the random number sequence provided to App 1, even if the number of times App 2 calls LFSRRand (see page 222)() varies.

**Description**

Seeds the LFSR random number generator invoked by the LFSRRand (see page 222)() function. The prior seed is returned.

**Remarks**

Upon initial power up, the internal seed is initialized to 0x1. Using a dwSeed value of 0x0 will return the same sequence of random numbers as using the seed of 0x1.

**Preconditions**

None

**Parameters**

Parameters	Description
wSeed	The new 32-bit seed value to assign to the LFSR.

## 10.7.3 Variables

### Module

Helpers (see page 206)

### Variables

	Name	Description
	dwLFSRRandSeed (see page 224)	Default Random Number Generator seed. 0x41FE9F9E corresponds to calling LFSRSeedRand (see page 223)(1)

### 10.7.3.1 dwLFSRRandSeed Variable

#### File

Helpers.c

#### C

```
DWORD dwLFSRRandSeed = 0x41FE9F9E;
```

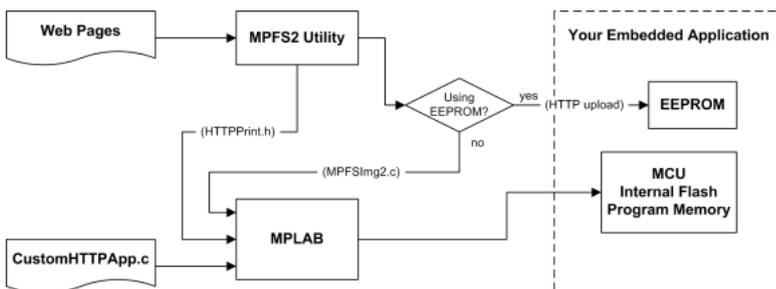
#### Description

Default Random Number Generator seed. 0x41FE9F9E corresponds to calling LFSRSeedRand (see page 223)(1)

## 10.8 HTTP2 Server

The HTTP2 web server module and its associated MPFS2 file system module allow the board to act as a web server. This facilitates an easy method to view status information and control applications using any standard web browser.

Three main components are necessary to understand how the HTTP2 web server works: the web pages, the MPFS2 Utility, and the source files CustomHTTPApp.c and HTTPPrint.h. An overview of the entire process is shown below.



### Web Pages

This includes all the HTML and associated images, CSS stylesheets, and JavaScript files necessary to display the website. A sample application including all these components is located in the WebPages2 folder.

### MPFS2 Utility

This program, supplied by Microchip, packages the web pages into a format that can be efficiently stored in either external non-volatile storage, or internal flash program memory. This program also indexes dynamic variables found in the web pages and updates HTTPPrint.h with these indices.

If external storage is being used, the MPFS2 Utility outputs a BIN file and can upload that file directly to the board. If the data is being stored in Flash program memory, the MPFS2 Utility will generate a C source file image to be included in the project.

When dynamic variables are added or removed from your application, the MPFS2 Utility will update `HTTPPrint.h`. When this happens, the project must be recompiled in the MPLAB IDE to ensure that all the new variable indices get added into the application.

### CustomHTTPApp.c

This file implements the web application. It describes the output for dynamic variables (via `HTTPPrint_varname` (see page 240) callbacks), parses data submitted through forms (in `HTTPExecuteGet` (see page 236) and `HTTPExecutePost` (see page 237)) and validates authorization credentials (in `HTTPAuthenticate`). The exact functionality of these callbacks is described within the demo application's web pages, and is also documented within the `CustomHTTPApp.c` example that is distributed with the stack.

### HTTPPrint.h

This file is generated automatically by the MPFS2 Utility. It indexes all the dynamic variables and provides the "glue" between the variables located in the web pages and their associated `HTTPPrint_varname` (see page 240) callback functions defined in `CustomHTTPApp.c`. This file does not require modification by the programmer.

---

## 10.8.1 HTTP2 Features

### Module

HTTP2 Server (see page 224)

### Description

The HTTP2 web server module has many capabilities. The following topics will introduce these features and provide examples.

### 10.8.1.1 HTTP2 Dynamic Variables

One of the most basic needs is to provide status information back to the user of your web application. The HTTP server provides for this using dynamic variable substitution callbacks. These commands in your HTML code will alert the server to execute a callback function at that point, which the developer creates to write data into the web page. Dynamic Variables should be considered the output of your application.

#### Basic Use

To create a dynamic variable, simply enclose the name of the variable inside a pair of tilde (~) characters within the web pages' HTML source code. (ex: `~myVariable~`) When you run the MPFS2 Utility to generate the web pages, it will automatically index these variables in `HTTPPrint.h`. This index will instruct your application to invoke the function `HTTPPrint_myVariable` when this string is encountered.

Here is an example of using a dynamic variable to insert the build date of your application into the web pages:

```
<div class="examplebox code">~builddate~</div>
```

The associated callback will print the value into the web page:

```
void HTTPPrint_builddate(void)
{
    TCPPutROMString(sktHTTP, (ROM void*)__DATE__);
}
```

#### Passing Parameters

You can also pass parameters to dynamic variables by placing numeric values inside of parenthesis after the variable name. For example, `~led(2)~` will print the value of the second LED. The numeric values are passed as WORD values to your callback function. You can pass as many parameters as you wish to these functions, and if your C code has constants defined, those will be parsed as well. (ex: `~pair(3,TRUE)~`)

The following code inserts the value of the push buttons into the web page, all using the same callback function:

```
<div class="examplebox code">btn(3)~ btn(2)~ btn(1)~ btn(0)~</div>
```

This associated callback will print the value of the requested button to the web page:

```
void HTTPPrint_btn(WORD num)
{
    // Determine which button
    switch(num)
    {
        case 0:
            num = BUTTON0_IO;
            break;
        case 1:
            num = BUTTON1_IO;
            break;
        case 2:
            num = BUTTON2_IO;
            break;
        case 3:
            num = BUTTON3_IO;
            break;
        default:
            num = 0;
    }

    // Print the output
    if(num == 1)
        TCPPutROMString(sktHTTP, "up");
    else
        TCPPutROMString(sktHTTP, "down");
}
```

### Longer Outputs

The HTTP protocol operates in a fixed memory buffer for transmission, so not all data can be sent at once. Care must be taken inside of your callback function to avoid overrunning this buffer.

The HTTP2 web server verifies that at least 16 bytes are free in this buffer before invoking a callback. For short outputs (less than 16 bytes), callbacks need only to call the appropriate TCPput (see page 454) function and return. For longer outputs, callback functions must check how much space is available, write up to that many bytes, then return. The callback will be invoked again when more space is free.

To manage the output state, callbacks should make use of `curHTTP.callbackPos`. This DWORD value is set to zero when a callback is first invoked. If a callback is only writing part of its output, it should set this field to a non-zero value to indicate that it should be called again when more space is available. This value will be available to the callback during the next call, which allows the function to resume output where it left off. A common use is to store the number of bytes written, or remaining to be written, in this field. Once the callback is finished writing its output, it must set `curHTTP.callbackPos` back to zero in order to indicate completion.

As an example, this code outputs the current value of the LCD display, which is 32 bytes on many Microchip development boards:

```
<div class="examplebox code">~lcdtext~</div>
```

The following callback function handles the output, and manages its state for multiple calls:

```
void HTTPPrint_lcdtext(void)
{
    WORD len;

    // Determine how many bytes we can write
    len = TCPIsPutReady(sktHTTP);

    // If just starting, set callbackPos
    if(curHTTP.callbackPos == 0)
        curHTTP.callbackPos = 32;

    // Write a byte at a time while we still can
```

```

// It may take up to 12 bytes to write a character
// (spaces and newlines are longer)
while(len > 12 && curHTTP.callbackPos)
{
    // After 16 bytes write a newline
    if(curHTTP.callbackPos == 16)
        len -= TCPPutROMString(sktHTTP, (ROM BYTE*)"<br />");

    if(LCDText[32-curHTTP.callbackPos] == ' ' || LCDText[32-curHTTP.callbackPos] ==
'\0')
        len -= TCPPutROMString(sktHTTP, (ROM BYTE*)"&nbsp;");
    else
        len -= TCPPut(sktHTTP, LCDText[32-curHTTP.callbackPos]);

    curHTTP.callbackPos--;
}
}

```

The initial call to `TCPisPutReady` (see page 450) determines how many bytes can be written to the buffer right now. The `TCPput` (see page 454) functions all return the number of bytes written, so we can subtract that value from `len` to track how much buffer space is left. When buffer space is exhausted, the function exits and waits to be called again. For subsequent calls, the value of `curHTTP.callbackPos` is exactly as we left it. The function resumes its output at that point.

### Including Files

Often it is useful to include the entire contents of another file in your output. Most web pages have at least some portion that does not change, such as the header, menu of links, and footer. These sections can be abstracted out into separate files which makes them easier to manage and conserves storage space.

To include the entire contents of another file, use a dynamic variable that starts with `"inc:"`, such as `~inc:header.inc~`. This sequence will cause the file `header.inc` to be read from the file system and inserted at this location.

The following example indicates how to include a standard menu bar section into every page:

```
<div id="menu">~inc:menu.inc~/div>
```

At this time, dynamic variables are not recursive, so any variables located inside files included in this manner are not parsed.

## 10.8.1.2 HTTP2 Form Processing

Many applications need to accept (see page 163) data from a user. A common solution is to present a form to the user in a web page, then have the device process the values submitted via this form. Web forms are usually submitted using one of two methods (**GET** and **POST**), and the HTTP2 web server supports both.

### The GET Method

The **GET** method appends the data to the end of the URI. This data follows the question mark (?) in the browser's address bar. (ex: `http://mchpboard/form.htm?led1=0&led2=1&led3=0`) Data sent via **GET** is automatically decoded and stored in the `curHTTP.data` array. Since it is to be stored in memory, this data is limited to the size of `curHTTP.data`, which by default is 100 bytes. However, it is generally easier to process data received in this manner.

The callback function `HTTPExecuteGet` (see page 236) is implemented by the application developer to process this data and perform any necessary actions. The functions `HTTPGetArg` (see page 238) and `HTTPGetROMArg` (see page 239) provide an easy method to retrieve submitted values for processing.

The following example demonstrates a form to control several LEDs.

```

<form method="get" action="leds.htm">
  LED 1: <input type="checkbox" name="led1" value="1" /><br />
  LED 2: <input type="checkbox" name="led2" value="1" /><br />
  LED 3: <input type="checkbox" name="led3" value="1" /><br />
  <input type="submit" value="Set LEDs" />
</form>

```

Suppose a user selects the checkboxes for LED 1 and LED3. The following string will be submitted to the server:

```
GET /leds.htm?led1=1&led3=1 HTTP/1.1
```

The HTTP2 web server will parse this request and store the following string in `curHTTP.data`:

```
"led1\01\0led3\01\0\0"
```

It will then call `HTTPExecuteGet` (see page 236) to process this input. To process this data, that callback needs to do several things. First, it should call `MPFSGetFilename` (see page 271) to verify which form was submitted. (This step may be omitted if only one form is provided by the application.) Next, since a checkbox control was used a default state of unchecked must be assumed. Finally, the callback should search for each argument it expects, compare the value, and set the LED pins accordingly. The following example satisfies all these requirements:

```
HTTP_IO_RESULT HTTPExecuteGet(void)
{
    BYTE *ptr, filename[20];

    // Load the file name (filename[] must be large enough to hold
    // the longest file name expected)
    MPFSGetFilename(curHTTP.file, filename, 20);

    // Verify the file name
    if(!strcmppgm2ram(filename, (ROM char*)"leds.htm"))
    {
        // Assume a default state of off
        LED1_IO = 0;
        LED2_IO = 0;
        LED3_IO = 0;

        // Search for each LED parameter and process
        ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE*)"led1");
        if(ptr)
            LED1_IO = (*ptr == '1');

        ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE*)"led2");
        if(ptr)
            LED2_IO = (*ptr == '1');

        ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE*)"led3");
        if(ptr)
            LED3_IO = (*ptr == '1');
    }

    // Indicate completion
    return HTTP_IO_DONE;
}
```

### The POST Method

The POST method transmits data after all the request headers have been sent. This data is not visible in the browser's address bar, and can only be seen with a packet capture tool. It does however use the same URL encoding method.

The HTTP2 server does not perform any pre-parsing of this data. All POST data is left in the TCP buffer, so the custom application will need to access the TCP buffer directly to retrieve and decode it. The functions `HTTPReadPostName` (see page 241) and `HTTPReadPostValue` (see page 242) have been provided to assist with these requirements. However, these functions can only be used when at least entire variables are expected to fit in the TCP buffer at once.

Most POST processing functions will be implemented as state machines in order to use these functions. The variable `curHTTP.smPost` is available to store the current state. This state machine variable is reset to zero with each new request. Functions should generally implement a state to read a variable name, and another to read an expected value. Additional states may be helpful depending on the application.

The following example form accepts an e-mail address, a subject, and a message body. Since this data will likely total over 100 bytes, it should be submitted via POST.

```
<form method="post" action="/email.htm">
  To: <input type="text" name="to" maxlength="50" /><br />
  Subject: <input type="text" name="subject" maxlength="50" /><br />
  Message:<br />
  <textarea name="msg" rows="6"></textarea><br />
```

```
<input type="submit" value="Send Message" /></div>
</form>
```

Suppose a user enters the following data into this form:

```
To: joe@picsaregood.com
Subject: Sent by a PIC
Message: I sent this message using my development board!
```

The HTTPExecutePost (see page 237) function will be called with the following data still in the TCP buffer:

```
to=joe%40picsaregood.com&subject=Sent+by+a+PIC
&msg=I+sent+this+message+using+my+development+board%21
```

To use the e-mail module, the application needs to read in the address and the subject, store those in RAM, then send the message. However, since the message is not guaranteed to fit in RAM all at once, it must be read as space is available and passed to the e-mail module. A state machine, coupled with the HTTPReadPostName (see page 241) and HTTPReadPostValue (see page 242) functions can simplify this greatly.

The following example callback function will properly parse this input. For this example, it is assumed that this is the only form the board accepts, so file name checking is not performed. The address will be stored at curHTTP.data[0:49], and the subject will be stored at curHTTP.data[50:99]. This is not the most optimal solution, but serves as a simple example.

```
HTTP_IO_RESULT HTTPExecutePost(void)
{
    BYTE *dest, temp[16];

    // Define state machine values
    #define SM_READ_NAME      (0u)
    #define SM_READ_VALUE    (1u)
    #define SM_START_MESSAGE (2u)
    #define SM_SEND_MESSAGE  (3u)

    switch(curHTTP.smPost)
    {
        case SM_READ_NAME:
            // Read the next variable name. If a complete name is
            // not found, request more data. This function will
            // automatically truncate invalid data to prevent
            // buffer overflows.
            if(HTTPReadPostName(temp,16) == HTTP_READ_INCOMPLETE)
                return HTTP_IO_NEED_DATA;

            // Save "to" values to curHTTP.data[0:49]
            if(!strcpgm2ram((char*)temp, (ROM char*)"to"))
                dest = curHTTP.data;

            // Save "subject" values to curHTTP.data[50:99]
            else if(!strcpgm2ram((char*)temp, (ROM char*)"subject"))
                dest = curHTTP.data + 50;

            // When a "msg" is encountered, start sending
            else if(!strcpgm2ram((char*)temp, (ROM char*)"to"))
            {
                curHTTP.smPost = SM_START_MESSAGE;
                break;
            }

            // Ignore unexpected values
            else
                dest = NULL;

            // Move to the next state, but do not break yet
            curHTTP.smPost = SM_READ_VALUE;

        case SM_READ_VALUE:
            // Read the next value. If a complete value is
            // not found, request more data. This function will
            // automatically truncate invalid data to prevent
            // buffer overflows.
            if(HTTPReadPostValue(dest,50) == HTTP_READ_INCOMPLETE)
                return HTTP_IO_NEED_DATA;
```

```

    // Return to read a new name
    curHTTP.smPost = SM_READ_NAME;
    break;

case SM_START_MESSAGE:
    // TODO: Perform necessary tasks to start sending the message.

    // Move on to sending the message body
    curHTTP.smPost = SM_SEND_MESSAGE;
    break;

case SM_SEND_MESSAGE:
    // The message may be longer than the TCP buffer can hold
    // at once. To avoid errors, read the data piece by
    // piece and send it to the e-mail module. This requires
    // using TCP functions directly.

    // Send all remaining data
    while(curHTTP.byteCount > 0)
    {
        // First check if data is ready
        if(TCPIsGetReady(sktHTTP) == 0)
            return HTTP_IO_NEED_DATA;

        // TODO: Read data with TCPGetArray and send
        // it to the e-mail module.
    }

    // Process is complete
    return HTTP_IO_DONE;
}

// Assume return for state machine convenience.
// Do not return HTTP_IO_NEED_DATA here by default, because
// doing so when more data will not arrive is cause for
// the HTTP2 server to return an error to the user.
return HTTP_IO_WAITING;
}

```

The previous example uses the `HTTPReadPostName` (see page 241) and `HTTPReadPostValue` (see page 242) functions, and also demonstrates using the need to use `TCPIsGetReady` (see page 450), `TCPGet` (see page 446), and `TCPGetArray` (see page 447) when longer values are expected. For applications that will receive and react to parameters immediately and have no need for a state machine, a simple while loop can be written around `HTTPReadPostPair` (see page 242) to accomplish the callback. The `HTTPPostLCD` (see page 89) function in the TCPIP Demo App provides a simple example of this.

For more examples, refer to `CustomHTTPApp.c` in the TCPIP Demo App project.

### 10.8.1.3 HTTP2 Authentication

The HTTP protocol provides a method for servers to request a user name and password from a client before granting access to a page. The HTTP2 server supports this authentication mechanism, allowing developers to require valid credentials for access to some or all pages.

Authentication (see page 84) functionality is supported by two user-provided callback functions. The first, `HTTPNeedsAuth` (see page 239), determines if the requested page requires valid credentials to proceed. The second, `HTTPVerifyAuth`, checks the user name and password against an accepted list and determines whether to grant or deny access. This split between two callback functions is necessitated by the nature of the HTTP protocol and the low-memory architecture of the HTTP2 server. In cases where different credentials or sets of credentials may be accepted for different pages, the two functions communicate with each other through a single byte stored in `curHTTP.isAuthorized`.

#### Requiring Authentication

When a request is first made, the function `HTTPNeedsAuth` (see page 239) is called to determine if that page needs

password protection. This function returns a value to instruct the HTTP2 server how to proceed. The most significant bit indicates whether or not access is granted. That is, values 0x80 and higher allow access unconditionally, while values 0x79 and lower will require a user name and password at a later point. The value returned is stored as `curHTTP.isAuthorized` so that it can be accessed by future callback functions.

The following example is the simplest case, in which all files require a password for access:

```
BYTE HTTPNeedsAuth(BYTE* cFile)
{
    return 0x00;
}
```

In some cases, only certain files will need to be protected. The second example requires a password for any file located in the `/treasure` folder:

```
BYTE HTTPNeedsAuth(BYTE* cFile)
{
    // Compare to "/treasure" folder. Don't use strcmp here, because
    // cFile has additional path info such as "/treasure/gold.htm"
    if(!memcmppgm2ram((void*)cFile, (ROM void*)"treasure", 5))
        return 0x00;

    return 0x80;
}
```

More complex uses could require an administrative user to access the `/admin` folder, while any authenticated user can access the rest of the site. The third example requires a different set of user name and password combinations for the `/admin` folder versus the rest of the site:

```
#define HTTP_AUTH_ADMIN      (0x00)
#define HTTP_AUTH_OTHER     (0x01)

BYTE HTTPNeedsAuth(BYTE* cFile)
{
    // Return a specific code for admin users
    if(!memcmppgm2ram((void*)cFile, (ROM void*)"admin", 5))
        return HTTP_AUTH_ADMIN;

    return HTTP_AUTH_OTHER;
}
```

### Validating Credentials

The `HTTPCheckAuth` (see page 235) function determines if the supplied user name and password are valid to access this resource. Again, the most significant bit indicates whether or not access is granted. The value returned is also stored as `curHTTP.isAuthorized` so that it can be accessed by future callback functions.

The following example is the simplest case, in which one user/password pair is accepted for all pages:

```
BYTE HTTPCheckAuth(BYTE* cUser, BYTE* cPass)
{
    if(!strcmppgm2ram((char*)cUser, (ROM char*)"AliBaba") &&
        !strcmppgm2ram((char*)cPass, (ROM char*)"Open Sesame!"))
        return 0x80;

    return 0x00;
}
```

In some cases, you may have multiple users with various levels of access. The following example satisfies the needs used in the third example of `HTTPNeedsAuth` (see page 239) above:

```
BYTE HTTPCheckAuth(BYTE* cUser, BYTE* cPass)
{
    // Check for admin users first
    if(curHTTP.isAuthorized == HTTP_AUTH_ADMIN &&
        !strcmppgm2ram((char*)cUser, (ROM char*)"admin") &&
        !strcmppgm2ram((char*)cPass, (ROM char*)"s3cREt"))
        return 0x80;

    if(!strcmppgm2ram((char*)cUser, (ROM char*)"kate") &&
        !strcmppgm2ram((char*)cPass, (ROM char*)"puppies!"))

```

```

        return 0x80;
    }
    return 0x00;
}

```

More complex uses are certainly feasible. Many applications may choose to store the user names and passwords in EEPROM or other non-volatile storage so that they may be updated by the end-user. Some applications may wish to return various values above 0x80 in HTTPCheckAuth (see page 235) so that later callback functions can determine which user logged in. The flexibility of these functions provides for many more possibilities that are not documented here but can be developed in just a few hours.

## 10.8.1.4 HTTP2 Cookies

By design, HTTP is a session-less and state-less protocol; every connection is an independent session with no relation to another. Cookies (see page 86) were added to the protocol description to solve this problem. This feature allows a web server to store small bits of text in a user's browser. These values will be returned to the server with every request, allowing the server to associate session variables with a request. Cookies (see page 86) are typically used for more advanced authentication systems.

Best practice is generally to store the bulk of the data on the server, and store only a unique identifier with the browser. This cuts down on data overhead and ensures that the user cannot modify the values stored with the session. However, logic must be implemented in the server to expire old sessions and allocate memory for new ones. If sensitive data is being stored, it is also important that the identifier be random enough so as to prevent stealing or spoofing another user's cookies.

### Retrieving Cookies

In the HTTP2 server, cookies are retrieved automatically. They are stored in `curHTTP.data`, just as any other GET form argument or URL parameter would be. The proper place to parse these values is therefore in the HTTPExecuteGet (see page 236) callback using the HTTPGetArg (see page 238) or HTTPGetROMArg (see page 239) functions to locate the values.

This model consumes some of the limited space available for URL parameters. Ensure that cookies do not consume more space than is available (as defined by `HTTP_MAX_DATA_LEN` (see page 248)) and that they will fit after any data that may be submitted via a GET form. If enough space is not available, the cookies will be truncated.

### Setting Cookies

Cookies (see page 86) can be set in HTTPExecuteGet (see page 236) or HTTPExecutePost (see page 237). To set a cookie, store the name/value pairs in `curHTTP.data` as a series of null-terminated strings. Then set, `curHTTP.hasArgs` equal to the number of name/value pairs to be set. For example, the following code sets a cookie indicating a user's preference for a type of cookie:

```

void HTTPExecuteGet(void)
{
    ...

    // Set a cookie
    strcpypgm2ram((char*)curHTTP.data, (ROM char*)"flavor\0oatmeal raisin");
    curHTTP.hasArgs = 1;

    ...
}

```

After this, all future requests from this browser will include the parameter "flavor" in `curHTTP.data`, along with the associated value of "oatmeal raisin".

## 10.8.1.5 HTTP2 Compression

All modern web browsers can receive files encoded with GZIP compression. For static files (those without dynamic variables), this can decrease the amount of data transmitted by as much as 60%.

The MPFS2 Utility will automatically determine which files can benefit from GZIP compression, and will store the compressed file in the MPFS2 image when possible. This generally includes all JavaScript and CSS files. (Images are typically already compressed, so the MPFS2 Utility will generally decide it is better to store them uncompressed.) This HTTP server will then seamlessly return this compressed file to the browser. Less non-volatile storage space will be required for the MPFS2 image, and faster transfers back to the client will result. No special configuration is required for this feature.

To prevent certain extensions from being compressed, use the Advanced Settings dialog in the MPFS2 Utility.

## 10.8.2 HTTP2 Public Members

### Enumerations

	Name	Description
	HTTP_IO_RESULT ( <a href="#">see page 235</a> )	Result states for execution callbacks
	HTTP_READ_STATUS ( <a href="#">see page 235</a> )	Result states for HTTPPostReadName and HTTPPostReadValue

### Functions

	Name	Description
	HTTPCheckAuth ( <a href="#">see page 235</a> )	Performs validation on a specific user name and password.
	HTTPExecuteGet ( <a href="#">see page 236</a> )	Processes GET form field variables and cookies.
	HTTPExecutePost ( <a href="#">see page 237</a> )	Processes POST form variables and data.
	HTTPGetArg ( <a href="#">see page 238</a> )	Locates a form field value in a given data array.
	HTTPGetROMArg ( <a href="#">see page 239</a> )	Locates a form field value in a given data array.
	HTTPNeedsAuth ( <a href="#">see page 239</a> )	Determines if a given file name requires authentication
	HTTPPrint_varname ( <a href="#">see page 240</a> )	Inserts dynamic content into a web page
	HTTPReadPostName ( <a href="#">see page 241</a> )	Reads a name from a URL encoded string in the TCP buffer.
	HTTPReadPostValue ( <a href="#">see page 242</a> )	Reads a value from a URL encoded string in the TCP buffer.
	HTTPURLDecode ( <a href="#">see page 243</a> )	Parses a string from URL encoding to plain-text.

### Macros

	Name	Description
	HTTPReadPostPair ( <a href="#">see page 242</a> )	Reads a name and value pair from a URL encoded string in the TCP buffer.
	sktHTTP ( <a href="#">see page 244</a> )	Access the current socket

### Module

HTTP2 Server ([see page 224](#))

### Structures

	Name	Description
	HTTP_CONN ( <a href="#">see page 234</a> )	Stores extended state data for each connection

**Variables**

	Name	Description
	curHTTP (see page 234)	Current HTTP connection state

**Description**

The following functions and variables are accessible or implemented by the stack application.

## 10.8.2.1 curHTTP Variable

**File**

HTTP2.c

**C**

```
HTTP_CONN curHTTP;
```

**Description**

Current HTTP connection state

## 10.8.2.2 HTTP\_CONN Structure

**File**

HTTP2.h

**C**

```
typedef struct {
    DWORD byteCount;
    DWORD nextCallback;
    DWORD callbackID;
    DWORD callbackPos;
    BYTE * ptrData;
    BYTE * ptrRead;
    MPFS_HANDLE file;
    MPFS_HANDLE offsets;
    BYTE hasArgs;
    BYTE isAuthorized;
    HTTP_STATUS httpStatus;
    HTTP_FILE_TYPE fileType;
    BYTE data[HTTP_MAX_DATA_LEN];
    BYTE smPost;
} HTTP_CONN;
```

**Members**

Members	Description
DWORD byteCount;	How many bytes have been read so far
DWORD nextCallback;	Byte index of the next callback
DWORD callbackID;	Callback ID to execute, also used as watchdog timer
DWORD callbackPos;	Callback position indicator
BYTE * ptrData;	Points to first free byte in data
BYTE * ptrRead;	Points to current read location
MPFS_HANDLE file;	File pointer for the file being served
MPFS_HANDLE offsets;	File pointer for any offset info being used
BYTE hasArgs;	True if there were get or cookie arguments
BYTE isAuthorized;	0x00-0x79 on fail, 0x80-0xff on pass
HTTP_STATUS httpStatus;	Request method/status

HTTP_FILE_TYPE fileType;	File type to return with Content-Type
BYTE data[HTTP_MAX_DATA_LEN];	General purpose data buffer
BYTE smPost;	POST state machine variable

**Description**

Stores extended state data for each connection

### 10.8.2.3 HTTP\_IO\_RESULT Enumeration

**File**

HTTP2.h

**C**

```
typedef enum {
    HTTP_IO_DONE = 0u,
    HTTP_IO_NEED_DATA,
    HTTP_IO_WAITING
} HTTP_IO_RESULT;
```

**Members**

Members	Description
HTTP_IO_DONE = 0u	Finished with procedure
HTTP_IO_NEED_DATA	More data needed to continue, call again later
HTTP_IO_WAITING	Waiting for asynchronous process to complete, call again later

**Description**

Result states for execution callbacks

### 10.8.2.4 HTTP\_READ\_STATUS Enumeration

**File**

HTTP2.h

**C**

```
typedef enum {
    HTTP_READ_OK = 0u,
    HTTP_READ_TRUNCATED,
    HTTP_READ_INCOMPLETE
} HTTP_READ_STATUS;
```

**Members**

Members	Description
HTTP_READ_OK = 0u	Read was successful
HTTP_READ_TRUNCATED	Buffer overflow prevented by truncating value
HTTP_READ_INCOMPLETE	Entire object is not yet in the buffer. Try again later.

**Description**

Result states for HTTPPostReadName and HTTPPostReadValue

### 10.8.2.5 HTTPCheckAuth Function

**File**

HTTP2.h

**C**

```

BYTE HTTPCheckAuth(
    BYTE* cUser,
    BYTE* cPass
);

```

**Description**

This function is implemented by the application developer in CustomHTTPApp.c. Its function is to determine if the user name and password supplied by the client are acceptable for this resource.

The value of curHTTP.isAuthorized will be set to the previous return value of HTTPRequiresAuthorization. This callback function can check this value to determine if only specific user names or passwords will be accepted for this resource.

Return values 0x80 - 0xff indicate that the credentials were accepted, while values from 0x00 to 0x79 indicate that authorization failed. While most applications will only use a single value to grant access, flexibility is provided to store multiple values in order to indicate which user (or user's group) logged in.

The return value of this function is saved as curHTTP.isAuthorized, and will be available to future callbacks, including any of the HTTPExecuteGet (see page 236), HTTPExecutePost (see page 237), or HTTPPrint\_varname (see page 240) callbacks.

**Remarks**

This function is only called when an Authorization header is encountered.

This function may NOT write to the TCP buffer.

**Internal**

See documentation in the TCP/IP Stack API or HTTP2.h for details.

**Preconditions**

None

**Parameters**

Parameters	Description
cUser	the user name supplied by the client
cPass	the password supplied by the client

**Return Values**

Return Values	Description
<= 0x79	the credentials were rejected
>= 0x80	access is granted for this connection

## 10.8.2.6 HTTPExecuteGet Function

**File**

HTTP2.h

**C**

```

HTTP_IO_RESULT HTTPExecuteGet();

```

**Description**

This function is implemented by the application developer in CustomHTTPApp.c. Its purpose is to parse the data received from URL parameters (GET method forms) and cookies and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated.

When this function is called, curHTTP.data contains sequential name/value pairs of strings representing the data received. In this format, HTTPGetArg (see page 238) and HTTPGetROMArg (see page 239) can be used to search for specific variables in the input. If data buffer space associated with this connection is required, curHTTP.data may be overwritten here

once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including HTTPExecutePost (see page 237) and any HTTPPrint\_varname (see page 240) dynamic substitutions.

This function may also issue redirections by setting curHTTP.data to the destination file name or URL, and curHTTP.httpStatus to HTTP\_REDIRECT.

Finally, this function may set cookies. Set curHTTP.data to a series of name/value string pairs (in the same format in which parameters arrive) and then set curHTTP.hasArgs equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in curHTTP.data.

### Remarks

This function is only called if variables are received via URL parameters or Cookie arguments. This function may NOT write to the TCP buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use curHTTP.callbackPos or curHTTP.data for storage associated with individual requests.

### Internal

See documentation in the TCP/IP Stack API or HTTP2.h for details.

### Preconditions

None

### Return Values

Return Values	Description
HTTP_IO_DONE	application is done processing
HTTP_IO_NEED_DATA	this value may not be returned because more data will not become available
HTTP_IO_WAITING	the application is waiting for an asynchronous process to complete, and this function should be called again later

## 10.8.2.7 HTTPExecutePost Function

### File

HTTP2.h

### C

```
HTTP_IO_RESULT HTTPExecutePost ( );
```

### Description

This function is implemented by the application developer in CustomHTTPApp.c. Its purpose is to parse the data received from POST forms and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated before this function is called.

When this function is called, POST data will be waiting in the TCP buffer. curHTTP.byteCount will indicate the number of bytes remaining to be received before the browser request is complete.

Since data is still in the TCP buffer, the application must call TCPGet (see page 446) or TCPGetArray (see page 447) in order to retrieve bytes. When this is done, curHTTP.byteCount MUST be updated to reflect how many bytes now remain. The functions TCPFind (see page 443), TCPFindString, TCPFindROMString, TCPFindArray (see page 443), and TCPFindROMArray (see page 445) may be helpful to locate data in the TCP buffer.

In general, data submitted from web forms via POST is URL encoded. The HTTPURLDecode (see page 243) function can be used to decode this information back to a standard string if required. If data buffer space associated with this connection is required, curHTTP.data may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including HTTPExecutePost and any HTTPPrint\_varname (see page 240) dynamic substitutions.

Whenever a POST form is processed it is recommended to issue a redirect back to the browser, either to a status page or to the same form page that was posted. This prevents accidental duplicate submissions (by clicking refresh or back/forward)

and avoids browser warnings about "resubmitting form data". Redirects may be issued to the browser by setting `curHTTP.data` to the destination file or URL, and `curHTTP.httpStatus` to `HTTP_REDIRECT`.

Finally, this function may set cookies. Set `curHTTP.data` to a series of name/value string pairs (in the same format in which parameters arrive) and then set `curHTTP.hasArgs` equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in `curHTTP.data`.

### Remarks

This function is only called when the request method is POST, and is only used when `HTTP_USE_POST` is defined. This method may NOT write to the TCP buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use `curHTTP.callbackPos` or `curHTTP.data` for storage associated with individual requests.

### Internal

See documentation in the TCP/IP Stack API or `HTTP2.h` for details.

### Preconditions

None

### Return Values

Return Values	Description
<code>HTTP_IO_DONE</code>	application is done processing
<code>HTTP_IO_NEED_DATA</code>	more data is needed to continue, and this function should be called again later
<code>HTTP_IO_WAITING</code>	the application is waiting for an asynchronous process to complete, and this function should be called again later

## 10.8.2.8 HTTPGetArg Function

### File

`HTTP2.h`

### C

```
BYTE* HTTPGetArg(
    BYTE* cData,
    BYTE* cArg
);
```

### Returns

A pointer to the argument value, or NULL if not found.

### Description

Searches through a data array to find the value associated with a given argument. It can be used to find form field values in data received over GET or POST.

The end of data is assumed to be reached when a null name parameter is encountered. This requires the string to have an even number of null-terminated strings, followed by an additional null terminator.

### Preconditions

The data array has a valid series of null terminated name/value pairs.

### Parameters

Parameters	Description
<code>data</code>	the buffer to search
<code>arg</code>	the name of the argument to find

## 10.8.2.9 HTTPGetROMArg Function

### File

HTTP2.h

### C

```
BYTE* HTTPGetROMArg(
    BYTE* cData,
    ROM BYTE* cArg
);
```

### Returns

A pointer to the argument value, or NULL if not found.

### Description

Searches through a data array to find the value associated with a given argument. It can be used to find form field values in data received over GET or POST.

The end of data is assumed to be reached when a null name parameter is encountered. This requires the string to have an even number of null-terminated strings, followed by an additional null terminator.

### Remarks

This function is aliased to HTTPGetArg (see page 238) on non-PIC18 platforms.

### Preconditions

The data array has a valid series of null terminated name/value pairs.

### Parameters

Parameters	Description
data	the buffer to search
arg	the name of the argument to find

## 10.8.2.10 HTTPNeedsAuth Function

### File

HTTP2.h

### C

```
BYTE HTTPNeedsAuth(
    BYTE* cFile
);
```

### Description

This function is implemented by the application developer in CustomHTTPApp.c. Its function is to determine if a file being requested requires authentication to view. The user name and password, if supplied, will arrive later with the request headers, and will be processed at that time.

Return values 0x80 - 0xff indicate that authentication is not required, while values from 0x00 to 0x79 indicate that a user name and password are required before proceeding. While most applications will only use a single value to grant access and another to require authorization, the range allows multiple "realms" or sets of pages to be protected, with different credential requirements for each.

The return value of this function is saved as curHTTP.isAuthorized, and will be available to future callbacks, including HTTPCheckAuth (see page 235) and any of the HTTPExecuteGet (see page 236), HTTPExecutePost (see page 237), or HTTPPrint\_varname (see page 240) callbacks.

**Remarks**

This function may NOT write to the TCP buffer.

**Internal**

See documentation in the TCP/IP Stack API or HTTP2.h for details.

**Preconditions**

None

**Parameters**

Parameters	Description
cFile	the name of the file being requested

**Return Values**

Return Values	Description
<= 0x79	valid authentication is required
>= 0x80	access is granted for this connection

## 10.8.2.11 HTTPPrint\_varname Function

**File**

HTTP2.h

**C**

```
void HTTPPrint_varname(
    WORD wParam1,
    WORD wParam2,
    ...
);
```

**Returns**

None

**Description**

Functions in this style are implemented by the application developer in CustomHTTPApp.c. These functions generate dynamic content to be inserted into web pages and other files returned by the HTTP2 server.

Functions of this type are called when a dynamic variable is located in a web page. (ie, ~varname~ ) The name between the tilde '~' characters is appended to the base function name. In this example, the callback would be named HTTPPrint\_varname.

The function prototype is located in your project's HTTPPrint.h, which is automatically generated by the MPFS2 Utility. The prototype will have WORD parameters included for each parameter passed in the dynamic variable. For example, the variable "~myArray(2,6)~" will generate the prototype "void HTTPPrint\_varname(WORD, WORD);".

When called, this function should write its output directly to the TCP socket using any combination of TCPIsPutReady (see page 450), TCPput (see page 454), TCPputArray (see page 454), TCPputString (see page 456), TCPputROMArray (see page 455), and TCPputROMString (see page 455).

Before calling, the HTTP2 server guarantees that at least HTTP\_MIN\_CALLBACK\_FREE (see page 248) bytes (defaults to 16 bytes) are free in the output buffer. If the function is writing less than this amount, it should simply write the data to the socket and return.

In situations where a function needs to write more this amount, it must manage its output state using curHTTP.callbackPos. This value will be set to zero before the function is called. If the function is managing its output state, it must set this to a non-zero value before returning. Typically this is used to track how many bytes have been written, or how many remain to be written. If curHTTP.callbackPos is non-zero, the function will be called again when more buffer space is available. Once the callback completes, set this value back to zero to resume normal servicing of the request.

**Remarks**

This function may service multiple HTTP requests simultaneously, especially when managing its output state. Exercise caution when using global or static variables inside this routine. Use curHTTP.callbackPos or curHTTP.data for storage associated with individual requests.

**Preconditions**

None

**Parameters**

Parameters	Description
wParam1	first parameter passed in the dynamic variable (if any)
wParam2	second parameter passed in the dynamic variable (if any)
...	additional parameters as necessary

## 10.8.2.12 HTTPReadPostName Function

**File**

HTTP2.h

**C**

```

HTTP_READ_STATUS HTTPReadPostName (
    BYTE* cData,
    WORD wLen
);
    
```

**Description**

Reads a name from a URL encoded string in the TCP buffer. This function is meant to be called from an HTTPExecutePost (see page 237) callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '=' character, which indicates the end of a name parameter. It assumes that the front of the buffer is the beginning of the name parameter to be read.

This function properly updates curHTTP.byteCount by decrementing it by the number of bytes read. It also removes the delimiting '=' from the buffer.

**Preconditions**

Front of TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character.

**Parameters**

Parameters	Description
cData	where to store the name once it is read
wLen	how many bytes can be written to cData

**Return Values**

Return Values	Description
HTTP_READ_OK	name was successfully read
HTTP_READ_TRUNCATED	entire name could not fit in the buffer, so the value was truncated and data has been lost
HTTP_READ_INCOMPLETE	entire name was not yet in the buffer, so call this function again later to retrieve

## 10.8.2.13 HTTPReadPostPair Macro

**File**

HTTP2.h

**C**

```
#define HTTPReadPostPair(cData, wLen) HTTPReadPostValue(cData, wLen)
```

**Description**

Reads a name and value pair from a URL encoded string in the TCP buffer. This function is meant to be called from an HTTPExecutePost (see page 237) callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the beginning of the name parameter to be read.

This function properly updates curHTTP.byteCount by decrementing it by the number of bytes read. It also removes the delimiting '&' from the buffer.

Once complete, two strings will exist in the cData buffer. The first is the parameter name that was read, while the second is the associated value.

**Remarks**

This function is aliased to HTTPReadPostValue (see page 242), since they effectively perform the same task. The name is provided only for completeness.

**Preconditions**

Front of TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character and a value parameter terminated by a '&'.

**Parameters**

Parameters	Description
cData	where to store the name and value strings once they are read
wLen	how many bytes can be written to cData

**Return Values**

Return Values	Description
HTTP_READ_OK	name and value were successfully read
HTTP_READ_TRUNCATED	entire name and value could not fit in the buffer, so input was truncated and data has been lost
HTTP_READ_INCOMPLETE	entire name and value was not yet in the buffer, so call this function again later to retrieve

## 10.8.2.14 HTTPReadPostValue Function

**File**

HTTP2.h

**C**

```
HTTP_READ_STATUS HTTPReadPostValue(
    BYTE* cData,
    WORD wLen
);
```

**Description**

Reads a value from a URL encoded string in the TCP buffer. This function is meant to be called from an HTTPExecutePost (see page 237) callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData above the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the beginning of the value parameter to be read. If curHTTP.byteCount indicates that all expected bytes are in the buffer, it assumes that all remaining data is the value and acts accordingly.

This function properly updates curHTTP.byteCount by decrementing it by the number of bytes read. The terminating '&' character is also removed from the buffer.

**Preconditions**

Front of TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character.

**Parameters**

Parameters	Description
cData	where to store the value once it is read
wLen	how many bytes can be written to cData

**Return Values**

Return Values	Description
HTTP_READ_OK	value was successfully read
HTTP_READ_TRUNCATED	entire value could not fit in the buffer, so the value was truncated and data has been lost
HTTP_READ_INCOMPLETE	entire value was not yet in the buffer, so call this function again later to retrieve

## 10.8.2.15 HTTPURLDecode Function

**File**

HTTP2.h

**C**

```
BYTE* HTTPURLDecode(
    BYTE* cData
);
```

**Returns**

A pointer to the last null terminator in data, which is also the first free byte for new data.

**Description**

Parses a string from URL encoding to plain-text. The following conversions are made: '=' to '0', '&' to '0', '+' to ' ', and "%xx" to a single hex byte.

After completion, the data has been decoded and a null terminator signifies the end of a name or value. A second null terminator (or a null name parameter) indicates the end of all the data.

**Remarks**

This function is called by the stack to parse GET arguments and cookie data. User applications can use this function to decode POST data, but first need to verify that the string is null-terminated.

**Preconditions**

The data parameter is null terminated and has at least one extra byte free.

**Parameters**

Parameters	Description
cData	The string which is to be decoded in place.

## 10.8.2.16 sktHTTP Macro

**File**

HTTP2.h

**C**

```
#define sktHTTP httpStubs[curHTTPID].socket // Access the current socket
```

**Description**

Access the current socket

## 10.8.3 HTTP2 Stack Members

**Functions**

	Name	Description
≡	HTTPInit (see page 244)	Initializes the HTTP server module.
≡	HTTPServer (see page 245)	Performs periodic tasks for the HTTP2 module.

**Module**

HTTP2 Server (see page 224)

**Description**

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.8.3.1 HTTPInit Function

**File**

HTTP2.h

**C**

```
void HTTPInit();
```

**Returns**

None

**Description**

Sets all HTTP sockets to the listening state, and initializes the state machine and file handles for each connection. If SSL is enabled, opens a socket on that port as well.

**Remarks**

This function is called only one during lifetime of the application.

**Preconditions**

TCP must already be initialized.

**Section**

Function Prototypes

### 10.8.3.2 HTTPServer Function

**File**

HTTP2.h

**C**

```
void HTTPServer ( ) ;
```

**Returns**

None

**Description**

Browses through each open connection and attempts to process any pending operations.

**Remarks**

This function acts as a task (similar to one in an RTOS). It performs its task in a co-operative manner, and the main application must call this function repeatedly to ensure that all open or new connections are served in a timely fashion.

**Preconditions**

HTTPInit (see page 244)() must already be called.

## 10.8.4 HTTP2 Internal Members

**Enumerations**

	Name	Description
	HTTP_FILE_TYPE (see page 247)	File type definitions
	HTTP_STATUS (see page 248)	Supported Commands and Server Response Codes
	SM_HTTP2 (see page 256)	Basic HTTP Connection State Machine

**Functions**

	Name	Description
	HTTPHeaderParseAuthorization (see page 250)	Parses the "Authorization:" header for a request and verifies the credentials.
	HTTPHeaderParseContentLength (see page 251)	Parses the "Content-Length:" header for a request.
	HTTPHeaderParseCookie (see page 251)	Parses the "Cookie:" headers for a request and stores them as GET variables.
	HTTPHeaderParseLookup (see page 252)	Calls the appropriate header parser based on the index of the header that was read from the request.
	HTTPIncFile (see page 252)	Writes a file byte-for-byte to the currently loaded TCP socket.
	HTTPLoadConn (see page 253)	Switches the currently loaded connection for the HTTP2 module.
	HTTPMPFUpload (see page 253)	Saves a file uploaded via POST as the new MPFS image in EEPROM or external Flash.
	HTTPProcess (see page 254)	Performs any pending operations for the currently loaded HTTP connection.
	HTTPReadTo (see page 254)	Reads to a buffer until a specified delimiter character.

	HTTPSendFile ( <a href="#">see page 256</a> )	Serves up the next chunk of curHTTP ( <a href="#">see page 234</a> )'s file, up to a) available TX FIFO space or b) the next callback index, whichever comes first.
---	---	---

**Macros**

	Name	Description
	HTTP_CACHE_LEN ( <a href="#">see page 247</a> )	Max lifetime (sec) of static responses as string
	HTTP_MAX_DATA_LEN ( <a href="#">see page 248</a> )	Define the maximum data length for reading cookie and GET/POST arguments (bytes)
	HTTP_MAX_HEADER_LEN ( <a href="#">see page 248</a> )	Set to length of longest string above
	HTTP_MIN_CALLBACK_FREE ( <a href="#">see page 248</a> )	Define the minimum number of bytes free in the TX FIFO before executing callbacks
	HTTP_PORT ( <a href="#">see page 248</a> )	Define the listening port for the HTTP server
	HTTP_TIMEOUT ( <a href="#">see page 250</a> )	Max time (sec) to await more data before
	HTTPS_PORT ( <a href="#">see page 255</a> )	Define the listening port for the HTTPS server (if STACK_USE_SSL_SERVER is enabled)
	smHTTP ( <a href="#">see page 257</a> )	Access the current state machine
	RESERVED_HTTP_MEMORY ( <a href="#">see page 257</a> )	Macro indicating how much RAM to allocate on an ethernet controller to store HTTP state data.

**Module**

HTTP2 Server ([see page 224](#))

**Structures**

	Name	Description
	HTTP_STUB ( <a href="#">see page 249</a> )	HTTP Connection Struct Stores partial state data for each connection Meant for storage in fast access RAM

**Variables**

	Name	Description
	curHTTPID ( <a href="#">see page 246</a> )	ID of the currently loaded HTTP_CONN ( <a href="#">see page 234</a> )
	httpContentTypes ( <a href="#">see page 250</a> )	Content-type strings corresponding to HTTP_FILE_TYPE ( <a href="#">see page 247</a> )
	httpFileExtensions ( <a href="#">see page 250</a> )	
	HTTPRequestHeaders ( <a href="#">see page 255</a> )	Header strings for which we'd like to parse
	HTTPResponseHeaders ( <a href="#">see page 255</a> )	Initial response strings (Corresponding to HTTP_STATUS ( <a href="#">see page 248</a> ))
	httpStubs ( <a href="#">see page 256</a> )	HTTP stubs with state machine and socket

**Description**

The following functions and variables are designated as internal to the HTTP2 module.

### 10.8.4.1 curHTTPID Variable

**File**

HTTP2.c

**C**

```
BYTE curHTTPID;
```

**Description**

ID of the currently loaded HTTP\_CONN (see page 234)

### 10.8.4.2 HTTP\_CACHE\_LEN Macro

**File**

HTTP2.h

**C**

```
#define HTTP_CACHE_LEN ("600") // Max lifetime (sec) of static responses as string
```

**Description**

Max lifetime (sec) of static responses as string

### 10.8.4.3 HTTP\_FILE\_TYPE Enumeration

**File**

HTTP2.h

**C**

```
typedef enum {
    HTTP_TXT = 0u,
    HTTP_HTM,
    HTTP_HTML,
    HTTP_CGI,
    HTTP_XML,
    HTTP_CSS,
    HTTP_GIF,
    HTTP_PNG,
    HTTP_JPG,
    HTTP_JAVA,
    HTTP_WAV,
    HTTP_UNKNOWN
} HTTP_FILE_TYPE;
```

**Members**

Members	Description
HTTP_TXT = 0u	File is a text document
HTTP_HTM	File is HTML (extension .htm)
HTTP_HTML	File is HTML (extension .html)
HTTP_CGI	File is HTML (extension .cgi)
HTTP_XML	File is XML (extension .xml)
HTTP_CSS	File is stylesheet (extension .css)
HTTP_GIF	File is GIF image (extension .gif)
HTTP_PNG	File is PNG image (extension .png)
HTTP_JPG	File is JPG image (extension .jpg)
HTTP_JAVA	File is java (extension .java)
HTTP_WAV	File is audio (extension .wav)
HTTP_UNKNOWN	File type is unknown

**Description**

File type definitions

## 10.8.4.4 HTTP\_MAX\_DATA\_LEN Macro

### File

TCPIP MRF24WB.h

### C

```
#define HTTP_MAX_DATA_LEN (100u)
```

### Description

Define the maximum data length for reading cookie and GET/POST arguments (bytes)

## 10.8.4.5 HTTP\_MAX\_HEADER\_LEN Macro

### File

HTTP2.c

### C

```
#define HTTP_MAX_HEADER_LEN (15u)
```

### Description

Set to length of longest string above

## 10.8.4.6 HTTP\_MIN\_CALLBACK\_FREE Macro

### File

TCPIP MRF24WB.h

### C

```
#define HTTP_MIN_CALLBACK_FREE (16u)
```

### Description

Define the minimum number of bytes free in the TX FIFO before executing callbacks

## 10.8.4.7 HTTP\_PORT Macro

### File

TCPIP MRF24WB.h

### C

```
#define HTTP_PORT (80u)
```

### Description

Define the listening port for the HTTP server

## 10.8.4.8 HTTP\_STATUS Enumeration

### File

HTTP2.h

**C**

```
typedef enum {
    HTTP_GET = 0u,
    HTTP_POST,
    HTTP_BAD_REQUEST,
    HTTP_UNAUTHORIZED,
    HTTP_NOT_FOUND,
    HTTP_OVERFLOW,
    HTTP_INTERNAL_SERVER_ERROR,
    HTTP_NOT_IMPLEMENTED,
    HTTP_MPFS_FORM,
    HTTP_MPFS_UP,
    HTTP_MPFS_OK,
    HTTP_MPFS_ERROR,
    HTTP_REDIRECT,
    HTTP_SSL_REQUIRED
} HTTP_STATUS;
```

**Members**

Members	Description
HTTP_GET = 0u	GET command is being processed
HTTP_POST	POST command is being processed
HTTP_BAD_REQUEST	400 Bad Request will be returned
HTTP_UNAUTHORIZED	401 Unauthorized will be returned
HTTP_NOT_FOUND	404 Not Found will be returned
HTTP_OVERFLOW	414 Request-URI Too Long will be returned
HTTP_INTERNAL_SERVER_ERROR	500 Internal Server Error will be returned
HTTP_NOT_IMPLEMENTED	501 Not Implemented (not a GET or POST command)
HTTP_MPFS_FORM	Show the MPFS Upload form
HTTP_MPFS_UP	An MPFS Upload is being processed
HTTP_MPFS_OK	An MPFS Upload was successful
HTTP_MPFS_ERROR	An MPFS Upload was not a valid image
HTTP_REDIRECT	302 Redirect will be returned
HTTP_SSL_REQUIRED	403 Forbidden is returned, indicating SSL is required

**Description**

Supported Commands and Server Response Codes

### 10.8.4.9 HTTP\_STUB Structure

**File**

HTTP2.h

**C**

```
typedef struct {
    SM_HTTP2 sm;
    TCP_SOCKET socket;
} HTTP_STUB;
```

**Members**

Members	Description
SM_HTTP2 sm;	Current connection state
TCP_SOCKET socket;	Socket being served

**Description**

HTTP Connection Struct Stores partial state data for each connection Meant for storage in fast access RAM

## 10.8.4.10 HTTP\_TIMEOUT Macro

### File

HTTP2.h

### C

```
#define HTTP_TIMEOUT (45u) // Max time (sec) to await more data before
```

### Description

Max time (sec) to await more data before

## 10.8.4.11 httpContentTypes Variable

### File

HTTP2.c

### C

```
ROM char * ROM httpContentTypes[HTTP_UNKNOWN+1] = { "text/plain", "text/html", "text/html",
"text/html", "text/xml", "text/css", "image/gif", "image/png", "image/jpeg",
"application/java-vm", "audio/x-wave", "" };
```

### Description

Content-type strings corresponding to HTTP\_FILE\_TYPE (see page 247)

## 10.8.4.12 httpFileExtensions Variable

### File

HTTP2.c

### C

```
ROM char * ROM httpFileExtensions[HTTP_UNKNOWN+1] = { "txt", "htm", "html", "cgi", "xml",
"css", "gif", "png", "jpg", "cla", "wav", "\0\0\0" };
```

### Section

File and Content Type Settings

\*\*\*\*\*

File type extensions corresponding to HTTP\_FILE\_TYPE

## 10.8.4.13 HTTPHeaderParseAuthorization Function

### File

HTTP2.c

### C

```
static void HTTPHeaderParseAuthorization();
```

### Returns

None

**Description**

Parses the "Authorization:" header for a request. For example, "BASIC YWRtaW46cGFzc3dvcmQ=" is decoded to a user name of "admin" and a password of "password". Once read, HTTPCheckAuth (see page 235) is called from CustomHTTPApp.c to determine if the credentials are acceptable.

The return value of HTTPCheckAuth (see page 235) is saved in curHTTP.isAuthorized for later use by the application.

**Remarks**

This function is only available when HTTP\_USE\_AUTHENTICATION is defined.

**Preconditions**

None

## 10.8.4.14 HTTPHeaderParseContentLength Function

**File**

HTTP2.c

**C**

```
static void HTTPHeaderParseContentLength();
```

**Returns**

None

**Description**

Parses the "Content-Length:" header to determine how many bytes of POST data to expect after the request. This value is stored in curHTTP.byteCount.

**Remarks**

This function is only available when HTTP\_USE\_POST is defined.

**Preconditions**

None

## 10.8.4.15 HTTPHeaderParseCookie Function

**File**

HTTP2.c

**C**

```
static void HTTPHeaderParseCookie();
```

**Returns**

None

**Description**

Parses the "Cookie:" headers for a request. For example, "Cookie: name=Wile+E.+Coyote; order=ROCKET\_LAUNCHER" is decoded to "name=Wile+E.+Coyote&order=ROCKET\_LAUNCHER&" and stored as any other GET variable in curHTTP.data.

The user application can easily access these values later using the HTTPGetArg (see page 238)() and HTTPGetROMArg (see page 239)() functions.

**Remarks**

This function is only available when HTTP\_USE\_COOKIES is defined.

**Preconditions**

None

## 10.8.4.16 HTTPHeaderParseLookup Function

**File**

HTTP2.c

**C**

```
static void HTTPHeaderParseLookup(
    BYTE i
);
```

**Description**

Calls the appropriate header parser based on the index of the header that was read from the request.

**Preconditions**

None

**Parameters**

Parameters	Description
i	the index of the string found in HTTPRequestHeaders ( <a href="#">see page 255</a> )

**Return Values**

Return Values	Description
TRUE	the end of the file was reached and reading is done
FALSE	more data remains to be read

**Section**

Function Prototypes

## 10.8.4.17 HTTPIncFile Function

**File**

HTTP2.h

**C**

```
void HTTPIncFile(
    ROM BYTE* cFile
);
```

**Returns**

None

**Description**

Allows an entire file to be included as a dynamic variable, providing a basic templating system for HTML web pages. This reduces unneeded duplication of visual elements such as headers, menus, etc.

When curHTTP.callbackPos is 0, the file is opened and as many bytes as possible are written. The current position is then saved to curHTTP.callbackPos and the file is closed. On subsequent calls, reading begins at the saved location and continues. Once the end of the input file is reached, curHTTP.callbackPos is set back to 0 to indicate completion.

**Remarks**

Users should not call this function directly, but should instead add dynamic variables in the form of ~inc:filename.ext~ in their HTML code to include (for example) the file "filename.ext" at that specified location. The MPFS2 Generator utility will handle

the rest.

#### Preconditions

None

#### Parameters

Parameters	Description
cFile	the name of the file to be sent

## 10.8.4.18 HTTPLoadConn Function

#### File

HTTP2.c

#### C

```
static void HTTPLoadConn(
    BYTE hHTTP
);
```

#### Returns

None

#### Description

Saves the currently loaded HTTP connection back to Ethernet buffer RAM, then loads the selected connection into curHTTP (see page 234) in local RAM for processing.

#### Preconditions

None

#### Parameters

Parameters	Description
hHTTP	the connection ID to load

## 10.8.4.19 HTTPMPFSUpload Function

#### File

HTTP2.c

#### C

```
static HTTP_IO_RESULT HTTPMPFSUpload();
```

#### Description

Allows the MPFS image in EEPROM or external Flash to be updated via a web page by accepting a file upload and storing it to the external memory.

#### Remarks

This function is only available when MPFS uploads are enabled and the MPFS image is stored in EEPROM.

#### Internal

After the headers, the first line from the form will be the MIME separator. Following that is more headers about the file, which are discarded. After another CRLF pair the file data begins, which is read 16 bytes at a time and written to external memory.

#### Preconditions

MPFSFormat (see page 268)() has been called.

**Return Values**

Return Values	Description
HTTP_IO_DONE	on success
HTTP_IO_NEED_DATA	if more data is still expected

## 10.8.4.20 HTTPProcess Function

**File**

HTTP2.c

**C**

```
static void HTTPProcess();
```

**Returns**

None

**Description**

Performs any pending operations for the currently loaded HTTP connection.

**Preconditions**

HTTPInit (see page 244)() and HTTPLoadConn (see page 253)() have been called.

## 10.8.4.21 HTTPReadTo Function

**File**

HTTP2.c

**C**

```
static HTTP_READ_STATUS HTTPReadTo(
    BYTE delim,
    BYTE* buf,
    WORD len
);
```

**Description**

Reads from the TCP buffer to cData until either cDelim is reached, or until wLen - 2 bytes have been read. The value read is saved to cData and null terminated. (wLen - 2 is used so that the value can be passed to HTTPURLDecode (see page 243) later, which requires a null terminator plus one extra free byte.)

The delimiter character is removed from the buffer, but not saved to cData. If all data cannot fit into cData, it will still be removed from the buffer but will not be saved anywhere.

This function properly updates curHTTP.byteCount by decrementing it by the number of bytes read.

**Preconditions**

None

**Parameters**

Parameters	Description
cDelim	the character at which to stop reading, or NULL to read to the end of the buffer
cData	where to store the data being read
wLen	how many bytes can be written to cData

**Return Values**

Return Values	Description
HTTP_READ_OK	data was successfully read
HTTP_READ_TRUNCATED	entire data could not fit in the buffer, so the data was truncated and data has been lost
HTTP_READ_INCOMPLETE	delimiter character was not found

### 10.8.4.22 HTTPRequestHeaders Variable

**File**

HTTP2.c

**C**

```
ROM char * ROM HTTPRequestHeaders[] = { "Cookie:", "Authorization:", "Content-Length:" };
```

**Description**

Header strings for which we'd like to parse

### 10.8.4.23 HTTPResponseHeaders Variable

**File**

HTTP2.c

**C**

```
ROM char * ROM HTTPResponseHeaders[] = { "HTTP/1.1 200 OK\r\nConnection: close\r\n",
"HTTP/1.1 200 OK\r\nConnection: close\r\n", "HTTP/1.1 400 Bad Request\r\nConnection:
close\r\n\r\n400 Bad Request: can't handle Content-Length\r\n", "HTTP/1.1 401
Unauthorized\r\nWWW-Authenticate: Basic realm=\"Protected\"\r\nConnection: close\r\n\r\n401
Unauthorized: Password required\r\n", "HTTP/1.1 404 Not found\r\nConnection:
close\r\nContent-Type: text/html\r\n\r\n404: File not found<br>Use <a href=\"/"
HTTP_MPFS_UPLOAD "\>MPFS Upload</a> to program web pages\r\n", "HTTP/1.1 404 Not
found\r\nConnection: close\r\n\r\n404: File not found\r\n", "HTTP/1.1 414 Request-URI Too
Long\r\nConnection: close\r\n\r\n414 Request-URI Too Long: Buffer overflow detected\r\n",
"HTTP/1.1 500 Internal Server Error\r\nConnection: close\r\n\r\n500 Internal Server Error:
Expected data not present\r\n", "HTTP/1.1 501 Not Implemented\r\nConnection:
close\r\n\r\n501 Not Implemented: Only GET and POST supported\r\n", "HTTP/1.1 200
OK\r\nConnection: close\r\nContent-Type: text/html\r\n\r\n<html><body
style=\"margin:100px\"><form method=post action=\\\" HTTP_MPFS_UPLOAD \"\
enctype=\\\"multipart/form-data\"><b>MPFS Image Upload</b><p><input type=file name=i size=40>
&nbsp; <input type=submit value=\\\"Upload\\\"</form></body></html>", "", "HTTP/1.1 200
OK\r\nConnection: close\r\nContent-Type: text/html\r\n\r\n<html><body
style=\\\"margin:100px\"><b>MPFS Update Successful</b><p><a href=\\\"/\">Site main
page</a></body></html>", "HTTP/1.1 500 Internal Server Error\r\nConnection:
close\r\nContent-Type: text/html\r\n\r\n<html><body style=\\\"margin:100px\"><b>MPFS Image
Corrupt or Wrong Version</b><p><a href=\\\"/\" HTTP_MPFS_UPLOAD "\>Try
again?</a></body></html>", "HTTP/1.1 302 Found\r\nConnection: close\r\nLocation: ",
"HTTP/1.1 403 Forbidden\r\nConnection: close\r\n\r\n403 Forbidden: SSL Required - use
HTTPS\r\n" };
```

**Description**

Initial response strings (Corresponding to HTTP\_STATUS (see page 248))

### 10.8.4.24 HTTPS\_PORT Macro

**File**

TCPIP MRF24WB.h

**C**

```
#define HTTPS_PORT (443u)
```

**Description**

Define the listening port for the HTTPS server (if STACK\_USE\_SSL\_SERVER is enabled)

## 10.8.4.25 HTTPSendFile Function

**File**

HTTP2.c

**C**

```
static BOOL HTTPSendFile();
```

**Description**

Serves up the next chunk of curHTTP (see page 234)'s file, up to a) available TX FIFO space or b) the next callback index, whichever comes first.

**Preconditions**

curHTTP.file and curHTTP.offsets have both been opened for reading.

**Return Values**

Return Values	Description
TRUE	the end of the file was reached and reading is done
FALSE	more data remains to be read

## 10.8.4.26 httpStubs Variable

**File**

HTTP2.c

**C**

```
HTTP_STUB httpStubs[MAX_HTTP_CONNECTIONS];
```

**Description**

HTTP stubs with state machine and socket

## 10.8.4.27 SM\_HTTP2 Enumeration

**File**

HTTP2.h

**C**

```
typedef enum {
    SM_HTTP_IDLE = 0u,
    SM_HTTP_PARSE_REQUEST,
    SM_HTTP_PARSE_HEADERS,
    SM_HTTP_AUTHENTICATE,
    SM_HTTP_PROCESS_GET,
    SM_HTTP_PROCESS_POST,
    SM_HTTP_PROCESS_REQUEST,
    SM_HTTP_SERVE_HEADERS,
    SM_HTTP_SERVE_COOKIES,
    SM_HTTP_SERVE_BODY,
    SM_HTTP_SEND_FROM_CALLBACK,
}
```

```

    SM_HTTP_DISCONNECT
} SM_HTTP2;

```

**Members**

Members	Description
SM_HTTP_IDLE = 0u	Socket is idle
SM_HTTP_PARSE_REQUEST	Parses the first line for a file name and GET args
SM_HTTP_PARSE_HEADERS	Reads and parses headers one at a time
SM_HTTP_AUTHENTICATE	Validates the current authorization state
SM_HTTP_PROCESS_GET	Invokes user callback for GET args or cookies
SM_HTTP_PROCESS_POST	Invokes user callback for POSTed data
SM_HTTP_PROCESS_REQUEST	Begins the process of returning data
SM_HTTP_SERVE_HEADERS	Sends any required headers for the response
SM_HTTP_SERVE_COOKIES	Adds any cookies to the response
SM_HTTP_SERVE_BODY	Serves the actual content
SM_HTTP_SEND_FROM_CALLBACK	Invokes a dynamic variable callback
SM_HTTP_DISCONNECT	Disconnects the server and closes all files

**Description**

Basic HTTP Connection State Machine

## 10.8.4.28 smHTTP Macro

**File**

HTTP2.c

**C**

```
#define smHTTP httpStubs[curHTTPID].sm // Access the current state machine
```

**Description**

Access the current state machine

## 10.8.4.29 RESERVED\_HTTP\_MEMORY Macro

**File**

HTTP2.h

**C**

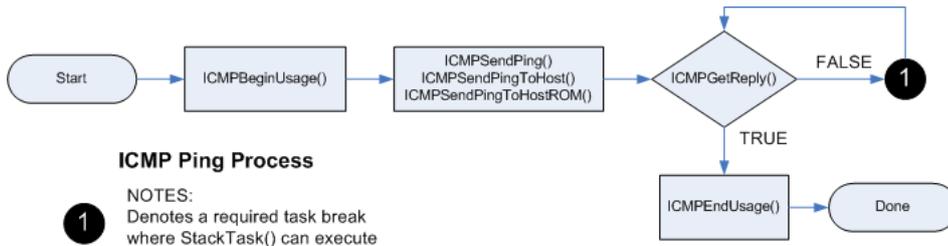
```
#define RESERVED_HTTP_MEMORY ((DWORD)MAX_HTTP_CONNECTIONS * (DWORD)sizeof(HTTP_CONN))
```

**Description**

Macro indicating how much RAM to allocate on an ethernet controller to store HTTP state data.

# 10.9 ICMP

The Internet Control Message Protocol is used to send error and status messages and requests. The ICMP module implements the Echo Reply message type (commonly referred to as a ping) which can be used to determine if a specified host is reachable across an IP network from a device running the TCP/IP stack. An ICMP server is also supported to respond to pings from other devices.



## 10.9.1 ICMP Public Members

### Functions

	Name	Description
≡	ICMPBeginUsage (see page 258)	Claims ownership of the ICMP module.
≡	ICMPSendPing (see page 259)	None
≡	ICMPSendPingToHost (see page 259)	None
≡	ICMPSendPingToHostROM (see page 260)	Begins the process of transmitting an ICMP echo request. This normally involves an ARP resolution procedure first.
≡	ICMPGetReply (see page 260)	None
≡	ICMPEndUsage (see page 261)	Gives up ownership of the ICMP module.

### Macros

	Name	Description
↔	ICMPSendPingToHostROM (see page 261)	This is macro ICMPSendPingToHostROM.

### Module

ICMP (see page 257)

### Description

The following functions and variables are accessible or implemented by the stack application.

### 10.9.1.1 ICMPBeginUsage Function

#### File

ICMP.h

#### C

```

    BOOL ICMPBeginUsage ( ) ;
  
```

#### Side Effects

None

#### Returns

TRUE: You have successfully gained ownership of the ICMP client module and can now use the ICMPSendPing (see page 259)() and ICMPGetReply (see page 260)() functions. FALSE: Some other application is using the ICMP client module. Calling ICMPSendPing (see page 259)() will corrupt the other application's ping result.

**Description**

Claims ownership of the ICMP module.

**Remarks**

None

**Preconditions**

None

## 10.9.1.2 ICMPSendPing Function

**File**

ICMP.h

**C**

```
void ICMPSendPing(
    DWORD dwRemoteIP
);
```

**Side Effects**

None

**Returns**

Begins the process of transmitting an ICMP echo request. This normally involves an ARP resolution procedure first.

**Description**

None

**Remarks**

None

**Preconditions**

ICMPBeginUsage (see page 258)() returned TRUE

**Parameters**

Parameters	Description
dwRemoteIP	IP Address (see page 141) to ping. Must be stored big endian. Ex. 192.168.0.1 should be passed as 0x0100A8C0.

## 10.9.1.3 ICMPSendPingToHost Function

**File**

ICMP.h

**C**

```
void ICMPSendPingToHost(
    BYTE * szRemoteHost
);
```

**Side Effects**

None

**Returns**

Begins the process of transmitting an ICMP echo request. This normally involves an ARP resolution procedure first.

**Description**

None

**Remarks**

None

**Preconditions**

ICMPBeginUsage (see page 258)() returned TRUE

**Parameters**

Parameters	Description
szRemoteHost	Host name to ping. Must be stored in RAM if being called by PIC18. Ex. www.microchip.com

## 10.9.1.4 ICMPSendPingToHostROM Function

**File**

ICMP.h

**C**

```
void ICMPSendPingToHostROM(
    ROM BYTE * szRemoteHost
);
```

**Side Effects**

None

**Returns**

None

**Description**

Begins the process of transmitting an ICMP echo request. This normally involves an ARP resolution procedure first.

**Remarks**

None

**Preconditions**

ICMPBeginUsage (see page 258)() returned TRUE

**Parameters**

Parameters	Description
szRemoteHost	Host name to ping. Must be stored in ROM. Should only be called by PIC18. Ex. www.microchip.com

## 10.9.1.5 ICMPGetReply Function

**File**

ICMP.h

**C**

```
LONG ICMPGetReply();
```

**Side Effects**

None

**Returns**

-3: Could not resolve hostname (DNS timeout or hostname invalid) -2: No response received yet -1: Operation timed out (longer than ICMP\_TIMEOUT (see page 264)) has elapsed. >=0: Number of TICKs that elapsed between initial ICMP transmission and reception of a valid echo.

**Description**

None

**Remarks**

None

**Preconditions**

ICMPBeginUsage (see page 258)() returned TRUE and ICMPSendPing (see page 259)() was called

## 10.9.1.6 ICMPEndUsage Function

**File**

ICMP.h

**C**

```
void ICMPEndUsage();
```

**Side Effects**

None

**Returns**

Your ownership of the ICMP module is released. You can no longer use ICMPSendPing (see page 259)().

**Description**

Gives up ownership of the ICMP module.

**Remarks**

None

**Preconditions**

ICMPBeginUsage (see page 258)() was called by you and it returned TRUE.

## 10.9.1.7 ICMPSendPingToHostROM Macro

**File**

ICMP.h

**C**

```
#define ICMPSendPingToHostROM(a) ICMPSendPingToHost((BYTE*)(a))
```

**Description**

This is macro ICMPSendPingToHostROM.

## 10.9.2 ICMP Internal Members

### Functions

	Name	Description
	ICMPProcess ( <a href="#">see page 262</a> )	None

### Macros

	Name	Description
	ICMP_TIMEOUT ( <a href="#">see page 264</a> )	ICMP Timeout Value

### Module

ICMP ([see page 257](#))

### Structures

	Name	Description
	ICMP_PACKET ( <a href="#">see page 263</a> )	ICMP Packet Structure

### Variables

	Name	Description
	ICMPFlags ( <a href="#">see page 263</a> )	ICMP Flag structure
	ICMPState ( <a href="#">see page 263</a> )	ICMP State Machine Enumeration
	ICMPTimer ( <a href="#">see page 264</a> )	ICMP tick timer variable
	StaticVars ( <a href="#">see page 264</a> )	ICMP Static Variables
	wICMPSequenceNumber ( <a href="#">see page 265</a> )	ICMP Sequence Number

### Description

The following functions and variables are designated as internal to the ICMP module.

### 10.9.2.1 ICMPProcess Function

#### File

ICMP.h

#### C

```
void ICMPProcess(
    NODE_INFO * remote,
    WORD len
);
```

#### Side Effects

None

#### Returns

Generates an echo reply, if requested Validates and sets ICMPFlags.bReplyValid if a correct ping response to one of ours is received.

#### Description

None

**Remarks**

None

**Preconditions**

MAC buffer contains ICMP type packet.

**Parameters**

Parameters	Description
*remote	Pointer to a NODE_INFO structure of the ping requester
len	Count of how many bytes the ping header and payload are in this IP packet

### 10.9.2.2 ICMPFlags Variable

**File**

ICMP.c

**C**

```

struct {
    unsigned char bICMPInUse : 1;
    unsigned char bReplyValid : 1;
    unsigned char bRemoteHostIsROM : 1;
} ICMPFlags;
    
```

**Members**

Members	Description
unsigned char bICMPInUse : 1;	Indicates that the ICMP Client is in use
unsigned char bReplyValid : 1;	Indicates that a correct Ping response to one of our pings was received
unsigned char bRemoteHostIsROM : 1;	Indicates that a remote host name was passed as a ROM pointer argument

**Description**

ICMP Flag structure

### 10.9.2.3 ICMP\_PACKET Structure

**File**

ICMP.c

**C**

```

typedef struct {
    BYTE vType;
    BYTE vCode;
    WORD wChecksum;
    WORD wIdentifier;
    WORD wSequenceNumber;
    WORD wData;
} ICMP_PACKET;
    
```

**Description**

ICMP Packet Structure

### 10.9.2.4 ICMPState Variable

**File**

ICMP.c

**C**

```
enum {  
    SM_IDLE = 0,  
    SM_DNS_SEND_QUERY,  
    SM_DNS_GET_RESPONSE,  
    SM_ARP_SEND_QUERY,  
    SM_ARP_GET_RESPONSE,  
    SM_ICMP_SEND_ECHO_REQUEST,  
    SM_ICMP_GET_ECHO_RESPONSE  
} ICMPState;
```

**Description**

ICMP State Machine Enumeration

## 10.9.2.5 ICMP\_TIMEOUT Macro

**File**

ICMP.c

**C**

```
#define ICMP_TIMEOUT (4ul*TICK_SECOND)
```

**Description**

ICMP Timeout Value

## 10.9.2.6 ICMPTimer Variable

**File**

ICMP.c

**C**

```
DWORD ICMPTimer;
```

**Description**

ICMP tick timer variable

## 10.9.2.7 StaticVars Variable

**File**

ICMP.c

**C**

```
union {  
    union {  
        ROM BYTE * szROM;  
        BYTE * szRAM;  
    } RemoteHost;  
    NODE_INFO ICMPRemote;  
} StaticVars;
```

**Description**

ICMP Static Variables

## 10.9.2.8 wICMPSequenceNumber Variable

### File

ICMP.c

### C

WORD **wICMPSequenceNumber** ;

### Description

ICMP Sequence Number

---

## 10.10 MPFS2

The MPFS2 file system module provides a light-weight read-only file system that can be stored in external EEPROM, external serial Flash, or internal Flash program memory. This file system serves as the basis for the HTTP2 web server module, but is also used by the SNMP module and is available to other applications that require basic read-only storage capabilities.

The MPFS2 module includes an MPFS2 Utility that runs on your PC. This program builds MPFS2 images in various formats for use in the supported storage mediums. More information is available in the MPFS2 Utility ([see page 57](#)) section.

### Using External Storage

For external storage, the MPFS2 file system supports Microchip 25LCxxx EEPROM parts for densities up to 1Mbit. SST 25VFxxxB serial Flash parts are also supported for densities up to 32Mbit.

To use external EEPROM storage, ensure that the configuration macro `MPFS_USE_EEPROM` is defined in `TCPIPConfig.h`. If you are using a 1Mbit part (25LC1024), also be sure to define `USE_EEPROM_25LC1024` to enable the 24-bit device addressing used by that part. For external serial Flash, define `MPFS_USE_SPI_FLASH` instead of the EEPROM macros.

Images stored externally are uploaded via HTTP. This can be accomplished using the MPFS2 Utility, or can be accessed directly from a browser. Uploading files directly ([see page 58](#)) is described in the MPFS2 Utility documentation. Uploading images via HTTP can be accomplished as described in the Getting Started ([see page 72](#)) section.

When storing images externally, space can be reserved for separate application use. The configuration macro `MPFS_RESERVE_BLOCK` controls the size of this space. The specified number of bytes will be reserved at the beginning address of the storage device (0x000000). When using serial Flash, this address must be a multiple of the flash sector size (4096 bytes).

### Using Internal Flash Storage

When storing images in internal Flash program memory, new images cannot be uploaded at run time. Instead, the image is compiled in as part of your project in the MPLAB IDE. To select this storage option comment out the configuration macro `MPFS_USE_EEPROM` in `TCPIPConfig.h`, then ensure that the image file generated by the MPFS2 Utility is included in the MPLAB project.

### Other Considerations

MPFS2 defines a fixed number of files that can be opened simultaneously. The configuration parameter `MAX_MPFS_HANDLES` controls how many files can be opened at once. If this resource is depleted, no new files can be opened until `MPFSClose` ([see page 268](#)) is called for an existing handle. The HTTP2 web server expects to be able to use at least two handles for each connection, plus one extra. Additional handles should be allocated if other modules will be accessing the file system as well.

## 10.10.1 MPFS2 Public Members

### Enumerations

	Name	Description
	MPFS_SEEK_MODE ( <a href="#">see page 267</a> )	Indicates the method for MPFSSeek ( <a href="#">see page 277</a> )

### Functions

	Name	Description
	MPFSClose ( <a href="#">see page 268</a> )	Closes a file.
	MPFSFormat ( <a href="#">see page 268</a> )	Prepares the MPFS image for writing.
	MPFSGet ( <a href="#">see page 269</a> )	Reads a byte from a file.
	MPFSGetArray ( <a href="#">see page 269</a> )	Reads a series of bytes from a file.
	MPFSGetBytesRem ( <a href="#">see page 270</a> )	Determines how many bytes remain to be read.
	MPFSGetEndAddr ( <a href="#">see page 270</a> )	Determines the ending address of a file.
	MPFSGetFilename ( <a href="#">see page 271</a> )	Reads the file name of a file that is already open.
	MPFSGetFlags ( <a href="#">see page 271</a> )	Reads a file's flags.
	MPFSGetID ( <a href="#">see page 272</a> )	Determines the ID in the FAT for a file.
	MPFSGetLong ( <a href="#">see page 272</a> )	Reads a DWORD or Long value from the MPFS.
	MPFSGetMicrotime ( <a href="#">see page 273</a> )	Reads the microtime portion of a file's timestamp.
	MPFSGetPosition ( <a href="#">see page 273</a> )	Determines the current position in the file
	MPFSGetSize ( <a href="#">see page 273</a> )	Reads the size of a file.
	MPFSGetStartAddr ( <a href="#">see page 274</a> )	Reads the starting address of a file.
	MPFSGetTimestamp ( <a href="#">see page 274</a> )	Reads the timestamp for the specified file.
	MPFSOpen ( <a href="#">see page 275</a> )	Opens a file in the MPFS2 file system.
	MPFSOpenID ( <a href="#">see page 275</a> )	Quickly re-opens a file.
	MPFSOpenROM ( <a href="#">see page 276</a> )	Opens a file in the MPFS2 file system.
	MPFSPutArray ( <a href="#">see page 276</a> )	Writes an array of data to the MPFS image.
	MPFSSeek ( <a href="#">see page 277</a> )	Moves the current read pointer to a new location.
	MPFSPutEnd ( <a href="#">see page 277</a> )	Finalizes an MPFS writing operation.

### Macros

	Name	Description
	MPFS_INVALID ( <a href="#">see page 267</a> )	Indicates a position pointer is invalid

	MPFS_INVALID_HANDLE ( <a href="#">see page 267</a> )	Indicates that a handle is not valid
---	---	--------------------------------------

**Module**

MPFS2 ([see page 265](#))

**Types**

	Name	Description
	MPFS_HANDLE ( <a href="#">see page 267</a> )	MPFS Handles are currently stored as BYTES

**Description**

The following functions and variables are accessible by the stack application.

### 10.10.1.1 MPFS\_HANDLE Type

**File**

MPFS2.h

**C**

```
typedef BYTE MPFS_HANDLE;
```

**Description**

MPFS Handles are currently stored as BYTES

### 10.10.1.2 MPFS\_INVALID Macro

**File**

MPFS2.h

**C**

```
#define MPFS_INVALID (0xfffffffffu) // Indicates a position pointer is invalid
```

**Description**

Indicates a position pointer is invalid

### 10.10.1.3 MPFS\_INVALID\_HANDLE Macro

**File**

MPFS2.h

**C**

```
#define MPFS_INVALID_HANDLE (0xffu) // Indicates that a handle is not valid
```

**Description**

Indicates that a handle is not valid

### 10.10.1.4 MPFS\_SEEK\_MODE Enumeration

**File**

MPFS2.h

**C**

```
typedef enum {
    MPFS_SEEK_START = 0u,
    MPFS_SEEK_END,
    MPFS_SEEK_FORWARD,
    MPFS_SEEK_REWIND
} MPFS_SEEK_MODE;
```

**Members**

Members	Description
MPFS_SEEK_START = 0u	Seek forwards from the front of the file
MPFS_SEEK_END	Seek backwards from the end of the file
MPFS_SEEK_FORWARD	Seek forward from the current position
MPFS_SEEK_REWIND	See backwards from the current position

**Description**

Indicates the method for MPFSSeek (see page 277)

### 10.10.1.5 MPFSClose Function

**File**

MPFS2.h

**C**

```
void MPFSClose(
    MPFS_HANDLE hMPFS
);
```

**Returns**

None

**Description**

Closes a file and releases its stub back to the pool of available handles.

**Preconditions**

None

**Parameters**

Parameters	Description
hMPFS (see page 335)	the file handle to be closed

### 10.10.1.6 MPFSFormat Function

**File**

MPFS2.h

**C**

```
MPFS_HANDLE MPFSFormat();
```

**Returns**

An MPFS handle that can be used for MPFSPut commands, or MPFS\_INVALID\_HANDLE (see page 267) when the EEPROM failed to initialize for writing.

**Description**

Prepares the MPFS image for writing and locks the image so that other processes may not access it.

**Remarks**

In order to prevent misreads, the MPFS will be inaccessible until MPFSClose (see page 268) is called. This function is not available when the MPFS is stored in internal Flash program memory.

**Preconditions**

None

### 10.10.1.7 MPFSGet Function

**File**

MPFS2.h

**C**

```

BOOL MPFSGet(
    MPFS_HANDLE hMPFS,
    BYTE* c
);
    
```

**Description**

Reads a byte from a file.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

Parameters	Description
hMPFS (see page 335)	the file handle from which to read
c	Where to store the byte that was read

**Return Values**

Return Values	Description
TRUE	The byte was successfully read
FALSE	No byte was read because either the handle was invalid or the end of the file has been reached.

### 10.10.1.8 MPFSGetArray Function

**File**

MPFS2.h

**C**

```

WORD MPFSGetArray(
    MPFS_HANDLE hMPFS,
    BYTE* cData,
    WORD wLen
);
    
```

**Returns**

The number of bytes successfully read. If this is less than wLen, an EOF occurred while attempting to read.

**Description**

Reads a series of bytes from a file.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

Parameters	Description
hMPFS (see page 335)	the file handle from which to read
cData	where to store the bytes that were read
wLen	how many bytes to read

## 10.10.1.9 MPFSGetBytesRem Function

**File**

MPFS2.h

**C**

```
DWORD MPFSGetBytesRem(
    MPFS_HANDLE hMPFS
);
```

**Returns**

The number of bytes remaining in the file as a DWORD

**Description**

Determines how many bytes remain to be read.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

Parameters	Description
hMPFS (see page 335)	the file handle from which to read the metadata

## 10.10.1.10 MPFSGetEndAddr Function

**File**

MPFS2.h

**C**

```
MPFS_PTR MPFSGetEndAddr(
    MPFS_HANDLE hMPFS
);
```

**Returns**

The address just after the file ends (start address of next file)

**Description**

Determines the ending address of a file.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

Parameters	Description
hMPFS (see page 335)	the file handle from which to read the metadata

## 10.10.1.11 MPFSGetFilename Function

### File

MPFS2.h

### C

```

BOOL MPFSGetFilename(
    MPFS_HANDLE hMPFS,
    BYTE* cName,
    WORD wLen
);

```

### Description

Reads the file name of a file that is already open.

### Preconditions

The file handle referenced by hMPFS is already open.

### Parameters

Parameters	Description
hMPFS (see page 335)	the file handle from which to determine the file name
cName	where to store the name of the file
wLen	the maximum length of data to store in cName

### Return Values

Return Values	Description
TRUE	the file name was successfully located
FALSE	the file handle provided is not currently open

## 10.10.1.12 MPFSGetFlags Function

### File

MPFS2.h

### C

```

WORD MPFSGetFlags(
    MPFS_HANDLE hMPFS
);

```

### Returns

The flags that were associated with the file

### Description

Reads a file's flags.

### Preconditions

The file handle referenced by hMPFS is already open.

### Parameters

Parameters	Description
hMPFS (see page 335)	the file handle from which to read the metadata

### 10.10.1.13 MPFSGetID Function

**File**

MPFS2.h

**C**

```
WORD MPFSGetID(
    MPFS_HANDLE hMPFS
);
```

**Returns**

The ID in the FAT for this file

**Description**

Determines the ID in the FAT for a file.

**Remarks**

Use this function in association with MPFSOpenID (see page 275) to quickly access file without permanently reserving a file handle.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

Parameters	Description
hMPFS (see page 335)	the file handle from which to read the metadata

### 10.10.1.14 MPFSGetLong Function

**File**

MPFS2.h

**C**

```
BOOL MPFSGetLong(
    MPFS_HANDLE hMPFS,
    DWORD* ul
);
```

**Returns**

TRUE - The byte was successfully read FALSE - No byte was read because either the handle was invalid or the end of the file has been reached.

**Description**

Reads a DWORD or Long value from the MPFS.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

Parameters	Description
hMPFS (see page 335)	the file handle from which to read
ul	where to store the DWORD or long value that was read

## 10.10.1.15 MPFSGetMicrotime Function

### File

MPFS2.h

### C

```
DWORD MPFSGetMicrotime(
    MPFS_HANDLE hMPFS
);
```

### Returns

The microtime that was read as a DWORD

### Description

Reads the microtime portion of a file's timestamp.

### Preconditions

The file handle referenced by hMPFS is already open.

### Parameters

Parameters	Description
hMPFS ( <a href="#">↗</a> see page 335)	the file handle from which to read the metadata

## 10.10.1.16 MPFSGetPosition Function

### File

MPFS2.h

### C

```
DWORD MPFSGetPosition(
    MPFS_HANDLE hMPFS
);
```

### Returns

The position in the file as a DWORD (or MPFS\_PTR ([↗](#) see page 280))

### Description

Determines the current position in the file

### Remarks

Calling MPFSSeek ([↗](#) see page 277)(hMPFS, pos, MPFS\_SEEK\_START) will return the pointer to this position at a later time. (Where pos is the value returned by this function.)

### Preconditions

The file handle referenced by hMPFS is already open.

### Parameters

Parameters	Description
hMPFS ( <a href="#">↗</a> see page 335)	the file handle for which to determine position

## 10.10.1.17 MPFSGetSize Function

### File

MPFS2.h

**C**

```
DWORD MPFSGetSize(
    MPFS_HANDLE hMPFS
);
```

**Returns**

The size that was read as a DWORD

**Description**

Reads the size of a file.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

Parameters	Description
hMPFS (see page 335)	the file handle from which to read the metadata

## 10.10.1.18 MPFSGetStartAddr Function

**File**

MPFS2.h

**C**

```
MPFS_PTR MPFSGetStartAddr(
    MPFS_HANDLE hMPFS
);
```

**Returns**

The starting address of the file in the MPFS image

**Description**

Reads the starting address of a file.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

Parameters	Description
hMPFS (see page 335)	the file handle from which to read the metadata

## 10.10.1.19 MPFSGetTimestamp Function

**File**

MPFS2.h

**C**

```
DWORD MPFSGetTimestamp(
    MPFS_HANDLE hMPFS
);
```

**Returns**

The timestamp that was read as a DWORD

**Description**

Reads the timestamp for the specified file.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

Parameters	Description
hMPFS (see page 335)	the file handle from which to read the metadata

## 10.10.1.20 MPFSOpen Function

**File**

MPFS2.h

**C**

```
MPFS_HANDLE MPFSOpen(
    BYTE* cFile
);
```

**Returns**

An MPFS\_HANDLE (see page 267) to the opened file if found, or MPFS\_INVALID\_HANDLE (see page 267) if the file could not be found or no free handles exist.

**Description**

Opens a file in the MPFS2 file system.

**Preconditions**

None

**Parameters**

Parameters	Description
cFile	a null terminated file name to open

## 10.10.1.21 MPFSOpenID Function

**File**

MPFS2.h

**C**

```
MPFS_HANDLE MPFSOpenID(
    WORD hFatID
);
```

**Returns**

An MPFS\_HANDLE (see page 267) to the opened file if found, or MPFS\_INVALID\_HANDLE (see page 267) if the file could not be found or no free handles exist.

**Description**

Quickly re-opens a file in the MPFS2 file system. Use this function along with MPFSGetID (see page 272)() to quickly re-open a file without tying up a permanent MPFSStub.

**Preconditions**

None

**Parameters**

Parameters	Description
hFatID	the ID of a previous opened file in the FAT

## 10.10.1.22 MPFSOpenROM Function

**File**

MPFS2.h

**C**

```
MPFS_HANDLE MPFSOpenROM(
    ROM BYTE* cFile
);
```

**Returns**

An MPFS\_HANDLE (see page 267) to the opened file if found, or MPFS\_INVALID\_HANDLE (see page 267) if the file could not be found or no free handles exist.

**Description**

Opens a file in the MPFS2 file system.

**Remarks**

This function is aliased to MPFSOpen (see page 275) on non-PIC18 platforms.

**Preconditions**

None

**Parameters**

Parameters	Description
cFile	a null terminated file name to open

## 10.10.1.23 MPFSPutArray Function

**File**

MPFS2.h

**C**

```
WORD MPFSPutArray(
    MPFS_HANDLE hMPFS,
    BYTE* cData,
    WORD wLen
);
```

**Returns**

The number of bytes successfully written.

**Description**

Writes an array of data to the MPFS image.

**Remarks**

For EEPROM, the actual write may not initialize until the internal write page is full. To ensure that previously written data gets stored, MPFSPutEnd (see page 277) must be called after the last call to MPFSPutArray.

**Preconditions**

MPFSFormat (see page 268) was successfully called.

**Parameters**

Parameters	Description
hMPFS (see page 335)	the file handle for writing

cData	the array of bytes to write
wLen	how many bytes to write

### 10.10.1.24 MPFSSeek Function

**File**

MPFS2.h

**C**

```

BOOL MPFSSeek(
    MPFS_HANDLE hMPFS,
    DWORD dwOffset,
    MPFS_SEEK_MODE tMode
);
    
```

**Returns**

TRUE - the seek was successful FALSE - either the new location or the handle itself was invalid

**Description**

Moves the current read pointer to a new location.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

Parameters	Description
hMPFS (see page 335)	the file handle to seek with
dwOffset	offset from the specified position in the specified direction
tMode	one of the MPFS_SEEK_MODE (see page 267) constants

### 10.10.1.25 MPFSPutEnd Function

**File**

MPFS2.h

**C**

```

void MPFSPutEnd(
    BOOL final
);
    
```

**Returns**

None

**Description**

Finalizes an MPFS writing operation.

**Preconditions**

MPFSFormat (see page 268) and MPFSPutArray (see page 276) were successfully called.

**Parameters**

Parameters	Description
final	TRUE if the application is done writing, FALSE if MPFS2 called this function locally.

## 10.10.2 MPFS2 Stack Members

### Functions

	Name	Description
≡◆	MPFSInit (🔗 see page 278)	Initializes the MPFS module.

### Module

MPFS2 (🔗 see page 265)

### Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.10.2.1 MPFSInit Function

#### File

MPFS2.h

#### C

```
void MPFSInit();
```

#### Returns

None

#### Description

Sets all MPFS handles to closed, and initializes access to the EEPROM if necessary.

#### Remarks

This function is called only one during lifetime of the application.

#### Preconditions

None

#### Section

Function Definitions

## 10.10.3 MPFS2 Internal Members

### Functions

	Name	Description
≡◆	ReadProgramMemory (🔗 see page 282)	Assembly function to read all three bytes of program memory for 16-bit parts
≡◆	_LoadFATRecord (🔗 see page 282)	Loads the FAT record for a specified handle.
≡◆	_Validate (🔗 see page 283)	Validates the MPFS Image

**Macros**

	<b>Name</b>	<b>Description</b>
↔	MAX_FILE_NAME_LEN (see page 280)	Supports long file names to 64 characters
↔	MPFS_WRITE_PAGE_SIZE (see page 281)	Defines the size of a page in EEPROM
↔	MPFS2_FLAG_HASINDEX (see page 281)	Indicates a file has an associated index of dynamic variables
↔	MPFS2_FLAG_ISZIPPED (see page 281)	Indicates a file is compressed with GZIP compression
↔	MPFSTell (see page 282)	Alias of MPFSGetPosition (see page 273)
↔	MPFS_INVALID_FAT (see page 284)	Indicates an invalid FAT cache

**Module**

MPFS2 (see page 265)

**Structures**

	<b>Name</b>	<b>Description</b>
🔗	MPFS_STUB (see page 280)	Stores each file handle's information Handles are free when addr = MPFS_INVALID (see page 267)
🔗	MPFS_FAT_RECORD (see page 283)	Stores the data for an MPFS2 FAT record

**Types**

	<b>Name</b>	<b>Description</b>
🔗	MPFS_PTR (see page 280)	MPFS Pointers are currently DWORDs

**Variables**

	<b>Name</b>	<b>Description</b>
🔗	isMPFSLocked (see page 279)	Allows the MPFS to be locked, preventing access during updates
🔗	lastRead (see page 280)	Track the last read address to prevent unnecessary data overhead to switch locations.
🔗	MPFSStubs (see page 281)	Track the MPFS File Handles MPFSStubs[0] is reserved for internal use (FAT access)
🔗	fatCache (see page 283)	FAT record cache
🔗	fatCacheID (see page 284)	ID of currently loaded fatCache (see page 283)
🔗	numFiles (see page 284)	Number of files in this MPFS image

**Description**

The following functions and variables are designated as internal to the MPFS2 module.

### 10.10.3.1 isMPFSLocked Variable

**File**

MPFS2.c

**C**

BOOL **isMPFSLocked**;

**Description**

Allows the MPFS to be locked, preventing access during updates

## 10.10.3.2 lastRead Variable

### File

MPFS2.c

### C

```
MPFS_PTR lastRead;
```

### Description

Track the last read address to prevent unnecessary data overhead to switch locations.

## 10.10.3.3 MAX\_FILE\_NAME\_LEN Macro

### File

MPFS2.c

### C

```
#define MAX_FILE_NAME_LEN (64u)
```

### Description

Supports long file names to 64 characters

## 10.10.3.4 MPFS\_PTR Type

### File

MPFS2.h

### C

```
typedef DWORD MPFS_PTR;
```

### Description

MPFS Pointers are currently DWORDs

## 10.10.3.5 MPFS\_STUB Structure

### File

MPFS2.h

### C

```
typedef struct {
    MPFS_PTR addr;
    DWORD bytesRem;
    WORD fatID;
} MPFS_STUB;
```

### Members

Members	Description
MPFS_PTR addr;	Current address in the file system
DWORD bytesRem;	How many bytes remain in this file
WORD fatID;	ID of which file in the FAT was accessed

**Description**

Stores each file handle's information. Handles are free when `addr = MPFS_INVALID` (see page 267)

### 10.10.3.6 MPFS\_WRITE\_PAGE\_SIZE Macro

**File**

MPFS2.h

**C**

```
#define MPFS_WRITE_PAGE_SIZE (64u) // Defines the size of a page in EEPROM
```

**Description**

Defines the size of a page in EEPROM

### 10.10.3.7 MPFS2\_FLAG\_HASINDEX Macro

**File**

MPFS2.h

**C**

```
#define MPFS2_FLAG_HASINDEX ((WORD)0x0002) // Indicates a file has an associated index of dynamic variables
```

**Description**

Indicates a file has an associated index of dynamic variables

### 10.10.3.8 MPFS2\_FLAG\_ISZIPPED Macro

**File**

MPFS2.h

**C**

```
#define MPFS2_FLAG_ISZIPPED ((WORD)0x0001) // Indicates a file is compressed with GZIP compression
```

**Description**

Indicates a file is compressed with GZIP compression

### 10.10.3.9 MPFSStubs Variable

**File**

MPFS2.c

**C**

```
MPFS_STUB MPFSStubs[MAX_MPFS_HANDLES+1];
```

**Description**

Track the MPFS File Handles. `MPFSStubs[0]` is reserved for internal use (FAT access)

### 10.10.3.10 MPFSTell Macro

#### File

MPFS2.h

#### C

```
#define MPFSTell(a) MPFSGetPosition(a)
```

#### Description

Alias of MPFSGetPosition (see page 273)

### 10.10.3.11 ReadProgramMemory Function

#### File

MPFS2.h

#### C

```
DWORD ReadProgramMemory(  
    DWORD address  
);
```

#### Description

Assembly function to read all three bytes of program memory for 16-bit parts

### 10.10.3.12 \_LoadFATRecord Function

#### File

MPFS2.c

#### C

```
static void _LoadFATRecord(  
    WORD fatID  
);
```

#### Returns

None

#### Description

Loads the FAT record for a specified handle.

#### Remarks

The FAT record will be stored in fatCache (see page 283).

#### Preconditions

None

#### Parameters

Parameters	Description
fatID	the ID of the file whose FAT is to be loaded

### 10.10.3.13 `_Validate` Function

**File**

MPFS2.c

**C**

```
static void _Validate();
```

**Returns**

None

**Description**

Verifies that the MPFS image is valid, and reads the number of available files from the image header. This function is called on boot, and again after any image is written.

**Preconditions**

None

### 10.10.3.14 `MPFS_FAT_RECORD` Structure

**File**

MPFS2.h

**C**

```
typedef struct {
    DWORD string;
    DWORD data;
    DWORD len;
    DWORD timestamp;
    DWORD microtime;
    WORD flags;
} MPFS_FAT_RECORD;
```

**Members**

Members	Description
DWORD string;	Pointer to the file name
DWORD data;	Address (see page 141) of the file data
DWORD len;	Length of file data
DWORD timestamp;	Timestamp of file
DWORD microtime;	Microtime stamp of file
WORD flags;	Flags for this file

**Description**

Stores the data for an MPFS2 FAT record

### 10.10.3.15 `fatCache` Variable

**File**

MPFS2.c

**C**

```
MPFS_FAT_RECORD fatCache;
```

**Description**

FAT record cache

### 10.10.3.16 fatCacheID Variable

**File**

MPFS2.c

**C**

```
WORD fatCacheID;
```

**Description**

ID of currently loaded fatCache (see page 283)

### 10.10.3.17 numFiles Variable

**File**

MPFS2.c

**C**

```
WORD numFiles;
```

**Description**

Number of files in this MPFS image

### 10.10.3.18 MPFS\_INVALID\_FAT Macro

**File**

MPFS2.h

**C**

```
#define MPFS_INVALID_FAT (0xffffu) // Indicates an invalid FAT cache
```

**Description**

Indicates an invalid FAT cache

---

## 10.11 NBNS

The NetBIOS Name Service protocol associates host names with IP addresses, similarly to DNS, but on the same IP subnet. Practically, this allows the assignment of human-name hostnames to access boards on the same subnet. For example, in the "TCPIP Demo App" demonstration project, the demo board is programmed with the human name 'mchpboard' so it can be accessed directly instead of with its IP address.

## 10.11.1 NBNS Stack Members

### Functions

	Name	Description
	NBNSGetName ( <a href="#">see page 285</a> )	Reads the NetBIOS name from a UDP socket and copies it into a user-specified buffer.
	NBNSPutName ( <a href="#">see page 286</a> )	Transmits the NetBIOS name across an open UDP socket.
	NBNSTask ( <a href="#">see page 286</a> )	Sends responses to NetBIOS name requests

### Macros

	Name	Description
	NBNS_PORT ( <a href="#">see page 287</a> )	NetBIOS Name Service port

### Module

NBNS ([see page 284](#))

### Structures

	Name	Description
	NBNS_HEADER ( <a href="#">see page 287</a> )	NBNS Header structure
	_NBNS_HEADER ( <a href="#">see page 287</a> )	NBNS Header structure

### Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.11.1.1 NBNSGetName Function

#### File

NBNS.c

#### C

```
static void NBNSGetName(
    BYTE * String
);
```

#### Side Effects

None

#### Returns

None

#### Description

Reads the NetBIOS name from a UDP socket and copies it into a user-specified buffer.

#### Remarks

None

#### Preconditions

None

**Parameters**

Parameters	Description
String	Pointer to an array into which a received NetBIOS name should be copied.

**10.11.1.2 NBNSPutName Function****File**

NBNS.c

**C**

```
static void NBNSPutName(
    BYTE * String
);
```

**Side Effects**

None

**Returns**

None

**Description**

Transmits the NetBIOS name across an open UDP socket.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
String	The name to transmit

**10.11.1.3 NBNSTask Function****File**

NBNS.h

**C**

```
void NBNSTask();
```

**Side Effects**

None

**Returns**

None

**Description**

Sends responses to NetBIOS name requests

**Remarks**

None

**Preconditions**

None

### 10.11.1.4 NBNS\_HEADER Structure

#### File

NBNS.c

#### C

```
typedef struct _NBNS_HEADER {
    WORD_VAL TransactionID;
    WORD_VAL Flags;
    WORD_VAL Questions;
    WORD_VAL Answers;
    WORD_VAL AuthoritativeRecords;
    WORD_VAL AdditionalRecords;
} NBNS_HEADER;
```

#### Description

NBNS Header structure

### 10.11.1.5 NBNS\_PORT Macro

#### File

NBNS.c

#### C

```
#define NBNS_PORT (137u)
```

#### Description

NetBIOS Name Service port

## 10.12 Performance Tests

The TCP and UDP Performance Test modules provide a method for obtaining metrics about the stack's performance. They can be used to benchmark the stack on various hardware platforms, and are also useful to determine how your custom application has affected stack performance.

The stack automatically calls these modules when they are included, so you never need to call any functions directly. Instructions for use of the modules can be found in the documentation for UDPPerformanceTask (see page 288), TCPTXPerformanceTask (see page 290), and TCPRXPerformanceTask (see page 289).

### 10.12.1 Performance Test Stack Members

#### Functions

	Name	Description
◆	TCPPerformanceTask (see page 288)	Tests the performance of the TCP module.
◆	UDPPerformanceTask (see page 288)	Tests the transmit performance of the UDP module.

**Module**

Performance Tests (see page 287)

**Description**

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

## 10.12.1.1 TCPPerformanceTask Function

**File**

TCPPerformanceTest.h

**C**

```
void TCPPerformanceTask( );
```

**Returns**

None

**Description**

This function calls both TCPTXPerformanceTask (see page 290) and TCPRXPerformanceTask (see page 289) to perform the performance task functions. Refer to the documentation for each of those functions for details.

**Preconditions**

TCP is initialized.

## 10.12.1.2 UDPPerformanceTask Function

**File**

UDPPerformanceTest.h

**C**

```
void UDPPerformanceTask( );
```

**Returns**

None

**Description**

This function tests the transmit performance of the UDP module. At boot, this module will transmit 1024 large UDP broadcast packets of 1024 bytes each. Using a packet sniffer, one can determine how long this process takes and calculate the transmit rate of the stack. This function tests true UDP performance in that it will open a socket, transmit one packet, and close the socket for each loop. After this initial transmission, the module can be re-enabled by holding button 3.

This function is particularly useful after development to determine the impact of your application code on the stack's performance. A before and after comparison will indicate if your application is unacceptably blocking the processor or taking too long to execute.

**Preconditions**

UDP is initialized.

## 10.12.2 Performance Test Internal Members

### Functions

	Name	Description
	TCPRXPerformanceTask ( <a href="#">see page 289</a> )	Tests the receive performance of the TCP module.
	TCPTXPerformanceTask ( <a href="#">see page 290</a> )	Tests the transmit performance of the TCP module.

### Macros

	Name	Description
	PERFORMANCE_PORT ( <a href="#">see page 290</a> )	Which UDP port to broadcast from for the UDP tests
	RX_PERFORMANCE_PORT ( <a href="#">see page 290</a> )	The TCP port to listen ( <a href="#">see page 169</a> ) on for TCP receive tests
	TX_PERFORMANCE_PORT ( <a href="#">see page 291</a> )	The TCP port to listen ( <a href="#">see page 169</a> ) on for TCP transmit tests

### Module

Performance Tests ([see page 287](#))

### Description

The following functions and variables are designated as internal to the module.

### 10.12.2.1 TCPRXPerformanceTask Function

#### File

TCPerformanceTest.c

#### C

```
void TCPRXPerformanceTask ( );
```

#### Returns

None

#### Description

This function tests the receive performance of the TCP module. To use, open a telnet connection to the device on RX\_PERFORMANCE\_PORT ([see page 290](#)) (9763 by default). Then use your telnet utility to upload a large file to the device. Each second the board will report back how many bytes were received in the previous second.

TCP performance is affected by many factors, including round-trip time and the TCP buffer size. For faster results, increase the size of the RX buffer size for the TCP\_PURPOSE\_TCP\_PERFORMANCE\_RX socket in TCPIPConfig.h. Round-trip time is affected by the distance to the device, so across the desk will be orders of magnitude faster than across the Internet.

This function is particularly useful after development to determine the impact of your application code on the stack's performance. A before and after comparison will indicate if your application is unacceptably blocking the processor or taking too long to execute.

#### Preconditions

TCP is initialized.

## 10.12.2.2 TCPTXPerformanceTask Function

### File

TCPPerformanceTest.c

### C

```
void TCPTXPerformanceTask( );
```

### Returns

None

### Description

This function tests the transmit performance of the TCP module. To use, open a telnet connection to the device on TX\_PERFORMANCE\_PORT (see page 291) (9762 by default). The board will rapidly transmit data and report its performance to the telnet client.

TCP performance is affected by many factors, including round-trip time and the TCP buffer size. For faster results, increase the size of the TX buffer size for the TCP\_PURPOSE\_TCP\_PERFORMANCE\_TX socket in TCPIPConfig.h. Round-trip time is affected by the distance to the device, so across the desk will be orders of magnitude faster than across the Internet.

This function is particularly useful after development to determine the impact of your application code on the stack's performance. A before and after comparison will indicate if your application is unacceptably blocking the processor or taking too long to execute.

### Preconditions

TCP is initialized.

## 10.12.2.3 PERFORMANCE\_PORT Macro

### File

UDPPerformanceTest.c

### C

```
#define PERFORMANCE_PORT 9
```

### Description

Which UDP port to broadcast from for the UDP tests

## 10.12.2.4 RX\_PERFORMANCE\_PORT Macro

### File

TCPPerformanceTest.c

### C

```
#define RX_PERFORMANCE_PORT 9763
```

### Description

The TCP port to listen (see page 169) on for TCP receive tests

## 10.12.2.5 TX\_PERFORMANCE\_PORT Macro

### File

TCPPerformanceTest.c

### C

```
#define TX_PERFORMANCE_PORT 9762
```

### Description

The TCP port to listen (see page 169) on for TCP transmit tests

---

## 10.13 SMTP Client

The SMTP client module in the TCP/IP Stack lets applications send e-mails to any recipient worldwide. These message could include status information or important alerts. Using the e-mail to SMS gateways provided by most cell phone carriers, these messages can also be delivered directly to cell phone handsets.

Using the SMTP client requires access to a local mail server (such as mail.yourdomain.com) for reliable operation. Your ISP or network administrator can provide the correct address, but end-user applications will need an interface to provide this data.

---

### 10.13.1 SMTP Client Examples

#### Module

SMTP Client (see page 291)

#### Description

The following two examples demonstrate the use of the SMTP client in different scenarios. The first, and simpler example, sends a short message whose contents are all located in RAM at once.

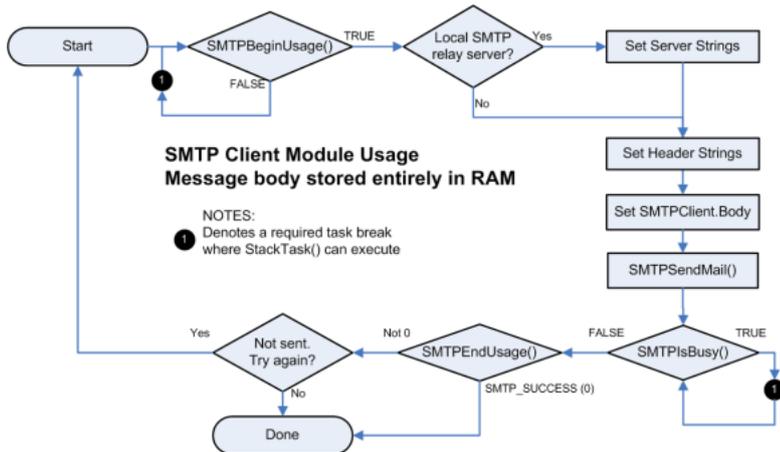
The second example is more involved and demonstrates generating a message on the fly in the case where the entire message cannot fit into RAM at once. In this case, the message is started by the stack, but the delivery of the contents happens in pieces and is handled by the application.

#### 10.13.1.1 SMTP Client Short Message Example

The SMTP client API is simplified when messages can be buffered entirely in RAM. The SMTPDemo (see page 92) example provided in MainDemo.c sends a brief e-mail message indicating the current status of the board's buttons. This document will walk through that example.

Make sure STACK\_USE\_SMTP\_CLIENT is uncommented in TCPConfig.h before continuing.

The diagram below provides an overview of the process:



First, call SMTPBeginUsage (see page 297) to verify that the SMTP client is available and to begin a new message. If FALSE is returned, the SMTP client is busy and the application must return to the main loop to allow StackTask to execute again.

Next, set the local relay server to use as SMTPClient.Server. If the local relay server requires a user name and password, set SMTPClient.Username and SMTPClient.Password to the appropriate credentials.

If server parameters are not set, the stack will attempt to deliver the message directly to its destination host. This will likely fail due to spam prevention measures put in place by most ISPs and network administrators.

Continue on to set the header strings as necessary for the message. This includes the subject line, from address, and any recipients you need to add. Finally, set SMTPClient.Body to the message to be sent.

At this point, verify that SMTPClient.ROMPointers is correctly configured for any strings that are stored in program memory. Once the message is ready to send, call SMTPSendMail (see page 303) to instruct the SMTP client to begin transmission.

The application must now call SMTPIsBusy (see page 299) until it returns FALSE. Each time TRUE is returned, return to the main loop and wait for StackTask to execute again. This allows the SMTP server to continue its work in a cooperative multitasking manner. Once FALSE is returned, call SMTPEndUsage (see page 298) to release the SMTP client. Check the return value of this function to determine if the message was successfully sent.

The example in MainDemo.c needs minor modifications to use your e-mail address. The Server and To fields must be set in SMTPDemo (see page 92) in order for the message to be properly delivered. Once this is done, holding down BUTTON2 and BUTTON3 simultaneously (the left-most two buttons) will begin sending the message. LED1 will light as the message is being processed, and will extinguish when the SMTP state machine completes. If the transmission was successful LED2 will light, otherwise it will remain dark.

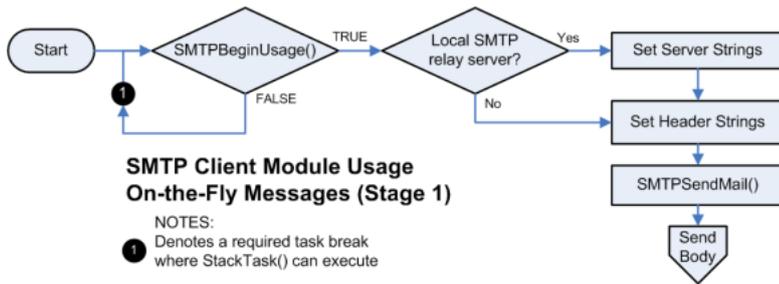
### 10.13.1.2 SMTP Client Long Message Example

The SMTP client API is capable of sending messages that do not fit entirely in RAM. To do so, the application must manage its output state and only write as many bytes as are available in the buffer at a time. The second SMTPDemo (see page 92) example provided in MainDemo.c sends a message that is a dump of all contents of the PIC's RAM. This example is currently commented out. Comment out the previous Short Message Example and uncomment the Long Message Example. This document will walk through sending a longer message.

Make sure STACK\_USE\_SMTP\_CLIENT is uncommented in TCPIPConfig.h before continuing.

Sending longer messages is divided into three stages. The first stage configures the SMTP client to send the message. The second stage sends the message in small chunks as buffer space is available. The final stage finishes the transmission and determines whether or not the message was successful.

The diagram below illustrates the first stage:



The first stage is largely similar to the first few steps in sending a short message. First, call SMTPBeginUsage (see page 297) to verify that the SMTP client is available and to begin a new message. If FALSE is returned, the SMTP client is busy and the application must return to the main loop to allow StackTask to execute again.

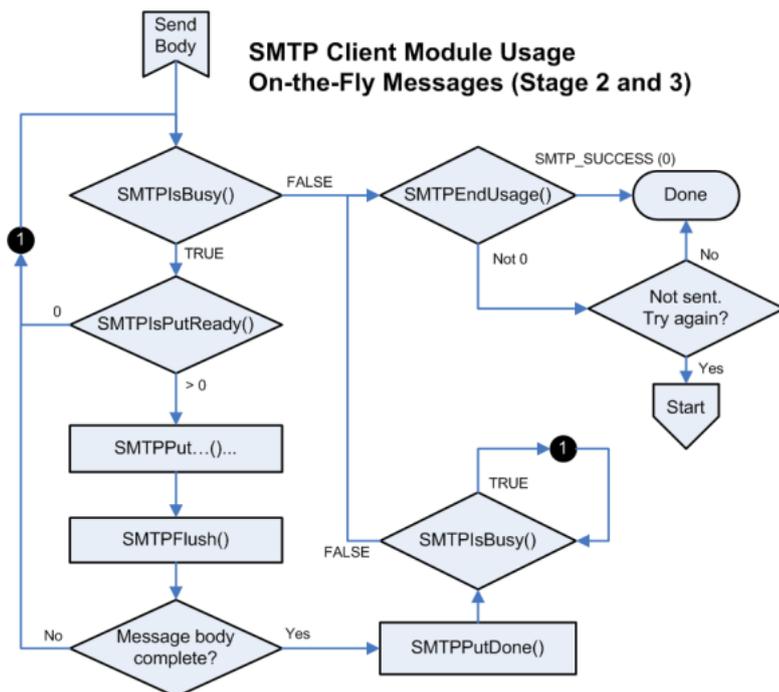
Next, set the local relay server to use as SMTPClient.Server. If the local relay server requires a user name and password, set SMTPClient.Username and SMTPClient.Password to the appropriate credentials.

If server parameters are not set, the stack will attempt to deliver the message directly to its destination host. This will likely fail due to spam prevention measures put in place by most ISPs and network administrators.

Continue on to set the header strings as necessary for the message. This includes the subject line, from address, and any recipients you need to add.

The next portion of the process differs. Ensure that SMTPClient.Body remains set to its default (NULL). At this point, call SMTPSendMail (see page 303) to open a connection to the remote server and transmit the headers. The application is now ready to proceed to the second stage and send the message body.

The following diagram provides an overview of stage two and three:



Upon entering stage two, the application should call SMTPIsBusy (see page 299) to verify that the connection to the remote server is active and has not been lost. If the call succeeds, call SMTPIsPutReady (see page 299) to determine how many bytes are available in the TX buffer. If no bytes are available, return to the main loop so that StackTask can transmit the data to the remote node and free up the buffer.

If space is available, any combination of the SMTPPut (see page 300), SMTPPutArray (see page 300), SMTPPutROMArray (see page 301), SMTPPutString (see page 302), and SMTPPutROMString (see page 302) functions may be called to transmit the message. These functions return the number of bytes successfully written. Use this value, along with the value originally returned from SMTPIsPutReady (see page 299) to track how much free space

remains in the TX buffer. Once the buffer is depleted, call `SMTPFlush` (see page 298) to force the data written to be sent.

The SMTP client module can accept (see page 163) as much data as the TCP TX FIFO can hold. This is determined by the socket initializer for `TCP_PURPOSE_DEFAULT` type sockets in `TCPConfig.h`, which defaults to 200 bytes.

If the TX buffer is exhausted before the message is complete, return to the main loop so that `StackTask` may transmit the data to the remote node and free up the buffer. Upon return, go to the beginning of the second stage to transmit the next portion of the message.

Once the message is complete, the application will move to the third stage. Call `SMTPPutDone` (see page 301) to inform the SMTP client that no more data remains. Then call `SMTPIsBusy` (see page 299) repeatedly. Each time `TRUE` is returned, return to the main loop and wait for `StackTask` to execute again. Once `FALSE` is returned, the message transmission has completed and the application must call `SMTPEndUsage` (see page 298) to release the SMTP client. Check the return value of this function to determine if the message was successfully sent.

The example in `MainDemo.c` needs minor modifications to use your e-mail address. Set the `Server` and `To` fields in `SMTPDemo` (see page 92), and ensure that these fields are being properly assigned to `SMTPClient` (see page 298) struct. The demo works exactly the same way as the previous one, with `BUTTON2` and `BUTTON3` held down simultaneously (the left-most two buttons) kicking off the state machine. `LED1` will light as the message is being processed, and will extinguish when the SMTP state machine completes. If the transmission was successful `LED2` will light, otherwise it will remain dark.

## 10.13.2 SMTP Client Public Members

### Functions

	Name	Description
⇒	<code>SMTPBeginUsage</code> (see page 297)	Requests control of the SMTP client module.
⇒	<code>SMTPEndUsage</code> (see page 298)	Releases control of the SMTP client module.
⇒	<code>SMTPFlush</code> (see page 298)	Flushes the SMTP socket and forces all data to be sent.
⇒	<code>SMTPIsBusy</code> (see page 299)	Determines if the SMTP client is busy.
⇒	<code>SMTPIsPutReady</code> (see page 299)	Determines how much data can be written to the SMTP client.
⇒	<code>SMTPPut</code> (see page 300)	Writes a single byte to the SMTP client.
⇒	<code>SMTPPutArray</code> (see page 300)	Writes a series of bytes to the SMTP client.
⇒	<code>SMTPPutDone</code> (see page 301)	Indicates that the on-the-fly message is complete.
⇒	<code>SMTPPutROMArray</code> (see page 301)	Writes a series of bytes from ROM to the SMTP client.
⇒	<code>SMTPPutROMString</code> (see page 302)	Writes a string from ROM to the SMTP client.
⇒	<code>SMTPPutString</code> (see page 302)	Writes a string to the SMTP client.
⇒	<code>SMTPSendMail</code> (see page 303)	Initializes the message sending process.

### Macros

	Name	Description
⇒	<code>SMTP_CONNECT_ERROR</code> (see page 295)	Connection to SMTP server failed

	SMTP_RESOLVE_ERROR ( <a href="#">see page 297</a> )	DNS lookup for SMTP server failed
	SMTP_SUCCESS ( <a href="#">see page 297</a> )	Message was successfully sent

**Module**

SMTP Client ([see page 291](#))

**Structures**

	Name	Description
	SMTP_POINTERS ( <a href="#">see page 295</a> )	Configures the SMTP client to send a message

**Variables**

	Name	Description
	SMTPClient ( <a href="#">see page 298</a> )	

**Description**

The following functions and variables are available to the stack application.

## 10.13.2.1 SMTP\_CONNECT\_ERROR Macro

**File**

SMTP.h

**C**

```
#define SMTP_CONNECT_ERROR (0x8001u) // Connection to SMTP server failed
```

**Description**

Connection to SMTP server failed

## 10.13.2.2 SMTP\_POINTERS Structure

**File**

SMTP.h

**C**

```
typedef struct {
    union {
        BYTE * szRAM;
        ROM BYTE * szROM;
    } Server;
    union {
        BYTE * szRAM;
        ROM BYTE * szROM;
    } Username;
    union {
        BYTE * szRAM;
        ROM BYTE * szROM;
    } Password;
    union {
        BYTE * szRAM;
        ROM BYTE * szROM;
    } To;
    union {
        BYTE * szRAM;
        ROM BYTE * szROM;
    } CC;
}
```

```

union {
    BYTE * szRAM;
    ROM BYTE * szROM;
} BCC;
union {
    BYTE * szRAM;
    ROM BYTE * szROM;
} From;
union {
    BYTE * szRAM;
    ROM BYTE * szROM;
} Subject;
union {
    BYTE * szRAM;
    ROM BYTE * szROM;
} OtherHeaders;
union {
    BYTE * szRAM;
    ROM BYTE * szROM;
} Body;
struct {
    unsigned char Server : 1;
    unsigned char Username : 1;
    unsigned char Password : 1;
    unsigned char To : 1;
    unsigned char CC : 1;
    unsigned char BCC : 1;
    unsigned char From : 1;
    unsigned char Subject : 1;
    unsigned char OtherHeaders : 1;
    unsigned char Body : 1;
} ROMPointers;
BOOL UseSSL;
WORD ServerPort;
} SMTP_POINTERS;
    
```

**Description**

This structure of pointers configures the SMTP Client to send an e-mail message. Initially, all pointers will be null. Set SMTPClient (see page 298).[field name].szRAM to use a string stored in RAM, or SMTPClient (see page 298).[field name].szROM to use a string stored in ROM. (Where [field name] is one of the parameters below.)

If a ROM string is specified, SMTPClient.ROMPointers.[field name] must also be set to 1 to indicate that this field should be retrieved from ROM instead of RAM.

**Remarks**

When formatting an e-mail address, the SMTP standard format for associating a printable name may be used. This format places the printable name in quotation marks, with the address following in pointed brackets, such as "John Smith" <john.smith@domain.com>

**Parameters**

Parameters	Description
Server	the SMTP server to relay the message through
Username	the user name to use when logging into the SMTP server, if any is required
Password	the password to supply when logging in, if any is required
To	the destination address for this message. May be a comma-separated list of addresss, and/or formatted.
CC	The CC addresses for this message, if any. May be a comma-separated list of addresss, and/or formatted.
BCC	The BCC addresses for this message, if any. May be a comma-separated list of addresss, and/or formatted.
From	The From address for this message. May be formatted.
Subject	The Subject header for this message.

OtherHeaders	Any additional headers for this message. Each additional header, including the last one, must be terminated with a CRLF pair.
Body	When sending a message from memory, the location of the body of this message in memory. Leave as NULL to build a message on-the-fly.
ROMPointers	Indicates which parameters to read from ROM instead of RAM.
UseSSL	When STACK_USE_SSL_CLIENT is enabled, this flag causes the SMTP client to make an SSL connection to the server.
ServerPort (see page 94)	(WORD value) Indicates the port on which to connect (see page 165) to the remote SMTP server.

### 10.13.2.3 SMTP\_RESOLVE\_ERROR Macro

**File**

SMTP.h

**C**

```
#define SMTP_RESOLVE_ERROR (0x8000u) // DNS lookup for SMTP server failed
```

**Description**

DNS lookup for SMTP server failed

### 10.13.2.4 SMTP\_SUCCESS Macro

**File**

SMTP.h

**C**

```
#define SMTP_SUCCESS (0x0000u) // Message was successfully sent
```

**Description**

Message was successfully sent

### 10.13.2.5 SMTPBeginUsage Function

**File**

SMTP.h

**C**

```
BOOL SMTPBeginUsage();
```

**Description**

Call this function before calling any other SMTP Client APIs. This function obtains a lock on the SMTP Client, which can only be used by one stack application at a time. Once the application is finished with the SMTP client, it must call SMTPEndUsage (see page 298) to release control of the module to any other waiting applications.

This function initializes all the SMTP state machines and variables back to their default state.

**Preconditions**

None

**Return Values**

Return Values	Description
TRUE	The application has successfully obtained control of the module

FALSE	The SMTP module is in use by another application. Call SMTPBeginUsage again later, after returning to the main program loop
-------	---

**Section**

SMTP Function Prototypes

### 10.13.2.6 SMTPClient Variable

**File**

SMTP.c

**C**

```
SMTP_POINTERS SMTPClient;
```

**Section**

SMTP Client Public Variables

\*\*\*\*\*

The global set of SMTP\_POINTERS.

Set these parameters after calling SMTPBeginUsage successfully.

### 10.13.2.7 SMTPEndUsage Function

**File**

SMTP.h

**C**

```
WORD SMTPEndUsage ( );
```

**Description**

Call this function to release control of the SMTP client module once an application is finished using it. This function releases the lock obtained by SMTPBeginUsage (see page 297), and frees the SMTP client to be used by another application.

**Preconditions**

SMTPBeginUsage (see page 297) returned TRUE on a previous call.

**Return Values**

Return Values	Description
SMTP_SUCCESS (see page 297)	A message was successfully sent
SMTP_RESOLVE_ERROR (see page 297)	The SMTP server could not be resolved
SMTP_CONNECT_ERROR (see page 295)	The connection to the SMTP server failed or was prematurely terminated
1-199 and 300-399	The last SMTP server response code

### 10.13.2.8 SMTPFlush Function

**File**

SMTP.h

**C**

```
void SMTPFlush( );
```

**Returns**

None

**Description**

Flushes the SMTP socket and forces all data to be sent.

**Remarks**

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, SMTPSendMail (see page 303) was called with SMTPClient.Body set to NULL.)

**Preconditions**

SMTPBeginUsage (see page 297) returned TRUE on a previous call.

## 10.13.2.9 SMTPIsBusy Function

**File**

SMTP.h

**C**

```
BOOL SMTPIsBusy( );
```

**Description**

Call this function to determine if the SMTP client is busy performing background tasks. This function should be called after any call to SMTPSendMail (see page 303), SMTPPutDone (see page 301) to determine if the stack has finished performing its internal tasks. It should also be called prior to any call to SMTPIsPutReady (see page 299) to verify that the SMTP client has not prematurely disconnected. When this function returns FALSE, the next call should be to SMTPEndUsage (see page 298) to release the module and obtain the status code for the operation.

**Preconditions**

SMTPBeginUsage (see page 297) returned TRUE on a previous call.

**Return Values**

Return Values	Description
TRUE	The SMTP Client is busy with internal tasks or sending an on-the-fly message.
FALSE	The SMTP Client is terminated and is ready to be released.

## 10.13.2.10 SMTPIsPutReady Function

**File**

SMTP.h

**C**

```
WORD SMTPIsPutReady( );
```

**Returns**

The number of free bytes the SMTP TX FIFO.

**Description**

Use this function to determine how much data can be written to the SMTP client when generating an on-the-fly message.

**Remarks**

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, SMTPSendMail (see page 303) was called with SMTPClient.Body set to NULL.)

**Preconditions**

SMTPBeginUsage (see page 297) returned TRUE on a previous call, and an on-the-fly message is being generated. This requires that SMTPSendMail (see page 303) was called with SMTPClient.Body set to NULL.

## 10.13.2.11 SMTPPut Function

**File**

SMTP.h

**C**

```
BOOL SMTPPut(
    BYTE c
);
```

**Description**

Writes a single byte to the SMTP client.

**Remarks**

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, SMTPSendMail (see page 303) was called with SMTPClient.Body set to NULL.)

**Preconditions**

SMTPBeginUsage (see page 297) returned TRUE on a previous call.

**Parameters**

Parameters	Description
c	The byte to be written

**Return Values**

Return Values	Description
TRUE	The byte was successfully written
FALSE	The byte was not written, most likely because the buffer was full

## 10.13.2.12 SMTPPutArray Function

**File**

SMTP.h

**C**

```
WORD SMTPPutArray(
    BYTE* Data,
    WORD Len
);
```

**Returns**

The number of bytes written. If less than Len, then the TX FIFO became full before all bytes could be written.

**Description**

Writes a series of bytes to the SMTP client.

**Remarks**

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, SMTPSendMail (see page 303) was called with SMTPClient.Body set to NULL.)

**Internal**

SMTPPut (see page 300) must be used instead of TCPPutArray (see page 454) because "rn." must be transparently replaced by "rn..".

**Preconditions**

SMTPBeginUsage (see page 297) returned TRUE on a previous call.

**Parameters**

Parameters	Description
Data	The data to be written
Len	How many bytes should be written

## 10.13.2.13 SMTPPutDone Function

**File**

SMTP.h

**C**

```
void SMTPPutDone ( );
```

**Returns**

None

**Description**

Indicates that the on-the-fly message is complete.

**Preconditions**

SMTPBeginUsage (see page 297) returned TRUE on a previous call, and the SMTP client is generated an on-the-fly message. (That is, SMTPSendMail (see page 303) was called with SMTPClient.Body set to NULL.)

## 10.13.2.14 SMTPPutROMArray Function

**File**

SMTP.h

**C**

```
WORD SMTPPutROMArray(
    ROM BYTE* Data,
    WORD Len
);
```

**Returns**

The number of bytes written. If less than Len, then the TX FIFO became full before all bytes could be written.

**Description**

Writes a series of bytes from ROM to the SMTP client.

**Remarks**

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, SMTPSendMail (see page 303) was called with SMTPClient.Body set to NULL.)

This function is aliased to SMTPPutArray (see page 300) on non-PIC18 platforms.

#### Internal

SMTPPut (see page 300) must be used instead of TCPPutArray (see page 454) because "rn." must be transparently replaced by "rn..".

#### Preconditions

SMTPBeginUsage (see page 297) returned TRUE on a previous call.

#### Parameters

Parameters	Description
Data	The data to be written
Len	How many bytes should be written

## 10.13.2.15 SMTPPutROMString Function

#### File

SMTP.h

#### C

```
WORD SMTPPutROMString(
    ROM BYTE* Data
);
```

#### Returns

The number of bytes written. If less than the length of Data, then the TX FIFO became full before all bytes could be written.

#### Description

Writes a string from ROM to the SMTP client.

#### Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, SMTPSendMail (see page 303) was called with SMTPClient.Body set to NULL.)

This function is aliased to SMTPPutString (see page 302) on non-PIC18 platforms.

#### Internal

SMTPPut (see page 300) must be used instead of TCPPutString (see page 456) because "rn." must be transparently replaced by "rn..".

#### Preconditions

SMTPBeginUsage (see page 297) returned TRUE on a previous call.

#### Parameters

Parameters	Description
Data	The data to be written

## 10.13.2.16 SMTPPutString Function

#### File

SMTP.h

#### C

```
WORD SMTPPutString(
    BYTE* Data
);
```

**Returns**

The number of bytes written. If less than the length of Data, then the TX FIFO became full before all bytes could be written.

**Description**

Writes a string to the SMTP client.

**Remarks**

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, SMTPSendMail (see page 303) was called with SMTPClient.Body set to NULL.)

**Internal**

SMTPPut (see page 300) must be used instead of TCPPutString (see page 456) because "rn." must be transparently replaced by "rn..".

**Preconditions**

SMTPBeginUsage (see page 297) returned TRUE on a previous call.

**Parameters**

Parameters	Description
Data	The data to be written

## 10.13.2.17 SMTPSendMail Function

**File**

SMTP.h

**C**

```
void SMTPSendMail();
```

**Returns**

None

**Description**

This function starts the state machine that performs the actual transmission of the message. Call this function after all the fields in SMTPClient (see page 298) have been set.

**Preconditions**

SMTPBeginUsage (see page 297) returned TRUE on a previous call.

## 10.13.3 SMTP Client Stack Members

**Functions**

	Name	Description
	SMTPTask (see page 304)	Performs any pending SMTP client tasks

**Module**

SMTP Client (see page 291)

**Description**

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.13.3.1 SMTPTask Function

**File**

SMTP.h

**C**

```
void SMTPTask( );
```

**Returns**

None

**Description**

This function handles periodic tasks associated with the SMTP client, such as processing initial connections and command sequences.

**Remarks**

This function acts as a task (similar to one in an RTOS). It performs its task in a co-operative manner, and the main application must call this function repeatedly to ensure that all open or new connections are served in a timely fashion.

**Preconditions**

None

---

## 10.13.4 SMTP Client Internal Members

**Functions**

	Name	Description
	FindEmailAddress ( <a href="#">see page 305</a> )	Searches a string for an e-mail address.
	FindROMEmailAddress ( <a href="#">see page 306</a> )	Searches a ROM string for an e-mail address.

**Macros**

	Name	Description
	SMTP_PORT ( <a href="#">see page 308</a> )	Default port to use when unspecified
	SMTP_SERVER_REPLY_TIMEOUT ( <a href="#">see page 308</a> )	How long to wait before assuming the connection has been dropped (default 8 seconds)

**Module**

SMTP Client ([see page 291](#))

**Variables**

	Name	Description
	CRPeriod ( <a href="#">see page 305</a> )	State machine for the CR LF Period replacement Used by SMTPPut ( <a href="#">see page 300</a> ) to transparently replace "rn." with "rn.."
	MySocket ( <a href="#">see page 306</a> )	Socket currently in use by the SMTP client
	PutHeadersState ( <a href="#">see page 306</a> )	State machine for writing the SMTP message headers
	ResponseCode ( <a href="#">see page 307</a> )	Response code from server when an error exists
	RXParserState ( <a href="#">see page 307</a> )	State machine for parsing incoming responses

	SMTPFlags (see page 308)	Internal flags used by the SMTP Client
	SMTPServer (see page 308)	IP address of the remote SMTP server
	SMTPState (see page 309)	Message state machine for the SMTP Client
	TransportState (see page 310)	State of the transport for the SMTP Client

**Description**

The following functions and variables are designated for internal use by the SMTP Client module.

### 10.13.4.1 CRPeriod Variable

**File**

SMTP.c

**C**

```
union {
    BYTE * Pos;
    enum {
        CR_PERIOD_SEEK_CR = 0,
        CR_PERIOD_SEEK_LF,
        CR_PERIOD_SEEK_PERIOD,
        CR_PERIOD_NEED_INSERTION
    } State;
} CRPeriod;
```

**Members**

Members	Description
CR_PERIOD_SEEK_CR = 0	Idle state, waiting for 'r'
CR_PERIOD_SEEK_LF	'r' has been written, so check next byte for 'n'
CR_PERIOD_SEEK_PERIOD	'rn' has been written, so check next byte for '.'
CR_PERIOD_NEED_INSERTION	'rn.' has been written, so an additional '.' must be written before the next byte.

**Description**

State machine for the CR LF Period replacement Used by SMTPPut (see page 300) to transparently replace "rn." with "rn.."

### 10.13.4.2 FindEmailAddress Function

**File**

SMTP.c

**C**

```
static BYTE * FindEmailAddress(
    BYTE * str,
    WORD * wLen
);
```

**Returns**

A pointer to the e-mail address

**Description**

This function locates an e-mail address in a string. It is used internally by the SMTP client to parse out the actual address from the From and To strings so that the MAIL FROM and RCPT TO commands can be sent to the SMTP server.

**Preconditions**

SMTPBeginUsage (see page 297) returned TRUE on a previous call.

**Parameters**

Parameters	Description
str	The string in which to search for an e-mail address
wLen	the length of str

**Section**

SMTP Client Internal Function Prototypes

## 10.13.4.3 FindROMEmailAddress Function

**File**

SMTP.c

**C**

```
static ROM BYTE * FindROMEmailAddress(
    ROM BYTE * str,
    WORD * wLen
);
```

**Returns**

A pointer to the e-mail address

**Description**

This function locates an e-mail address in a string. It is used internally by the SMTP client to parse out the actual address from the From and To strings so that the MAIL FROM and RCPT TO commands can be sent to the SMTP server.

**Preconditions**

SMTPBeginUsage (see page 297) returned TRUE on a previous call.

**Parameters**

Parameters	Description
str	The ROM string in which to search for an e-mail address
wLen	the length of str

## 10.13.4.4 MySocket Variable

**File**

SMTP.c

**C**

```
TCP_SOCKET MySocket = INVALID_SOCKET;
```

**Description**

Socket currently in use by the SMTP client

## 10.13.4.5 PutHeadersState Variable

**File**

SMTP.c

**C**

```
enum {
    PUTHEADERS_FROM_INIT = 0,
    PUTHEADERS_FROM,
    PUTHEADERS_TO_INIT,
    PUTHEADERS_TO,
    PUTHEADERS_CC_INIT,
    PUTHEADERS_CC,
    PUTHEADERS_SUBJECT_INIT,
    PUTHEADERS_SUBJECT,
    PUTHEADERS_OTHER_INIT,
    PUTHEADERS_OTHER,
    PUTHEADERS_DONE
} PutHeadersState;
```

**Members**

Members	Description
PUTHEADERS_FROM_INIT = 0	Preparing to send From header
PUTHEADERS_FROM	Sending From header
PUTHEADERS_TO_INIT	Preparing to send To header
PUTHEADERS_TO	Sending To header
PUTHEADERS_CC_INIT	Preparing to send CC header
PUTHEADERS_CC	Sending CC header
PUTHEADERS_SUBJECT_INIT	Preparing to send Subject header
PUTHEADERS_SUBJECT	Sending Subject header
PUTHEADERS_OTHER_INIT	Preparing to send additional headers
PUTHEADERS_OTHER	Sending additional headers
PUTHEADERS_DONE	Done writing all headers

**Description**

State machine for writing the SMTP message headers

### 10.13.4.6 ResponseCode Variable

**File**

SMTP.c

**C**

```
WORD ResponseCode;
```

**Description**

Response code from server when an error exists

### 10.13.4.7 RXParserState Variable

**File**

SMTP.c

**C**

```
enum {
    RX_BYTE_0 = 0,
    RX_BYTE_1,
    RX_BYTE_2,
    RX_BYTE_3,
    RX_SEEK_CR,
    RX_SEEK_LF
}
```

```
} RXParserState;
```

**Description**

State machine for parsing incoming responses

## 10.13.4.8 SMTP\_PORT Macro

**File**

SMTP.c

**C**

```
#define SMTP_PORT 25 // Default port to use when unspecified
```

**Description**

Default port to use when unspecified

## 10.13.4.9 SMTP\_SERVER\_REPLY\_TIMEOUT Macro

**File**

SMTP.c

**C**

```
#define SMTP_SERVER_REPLY_TIMEOUT (TICK_SECOND*8) // How long to wait before  
assuming the connection has been dropped (default 8 seconds)
```

**Description**

How long to wait before assuming the connection has been dropped (default 8 seconds)

## 10.13.4.10 SMTPFlags Variable

**File**

SMTP.c

**C**

```
union {  
    BYTE Val;  
    struct {  
        unsigned char RXSkipResponse : 1;  
        unsigned char SMTPInUse : 1;  
        unsigned char SentSuccessfully : 1;  
        unsigned char ReadyToStart : 1;  
        unsigned char ReadyToFinish : 1;  
        unsigned char ConnectedOnce : 1;  
        unsigned char filler : 2;  
    } bits;  
} SMTPFlags;
```

**Description**

Internal flags used by the SMTP Client

## 10.13.4.11 SMTPServer Variable

**File**

SMTP.c

**C**

```
IP_ADDR SMTPServer;
```

**Description**

IP address of the remote SMTP server

### 10.13.4.12 SMTPState Variable

**File**

SMTP.c

**C**

```
enum {
    SMTP_HOME = 0,
    SMTP_HELO,
    SMTP_HELO_ACK,
    SMTP_AUTH_LOGIN,
    SMTP_AUTH_LOGIN_ACK,
    SMTP_AUTH_USERNAME,
    SMTP_AUTH_USERNAME_ACK,
    SMTP_AUTH_PASSWORD,
    SMTP_AUTH_PASSWORD_ACK,
    SMTP_MAILFROM,
    SMTP_MAILFROM_ACK,
    SMTP_RCPTTO_INIT,
    SMTP_RCPTTO,
    SMTP_RCPTTO_ACK,
    SMTP_RCPTTO_ISDONE,
    SMTP_RCPTTOCC_INIT,
    SMTP_RCPTTOCC,
    SMTP_RCPTTOCC_ACK,
    SMTP_RCPTTOCC_ISDONE,
    SMTP_RCPTTOBCC_INIT,
    SMTP_RCPTTOBCC,
    SMTP_RCPTTOBCC_ACK,
    SMTP_RCPTTOBCC_ISDONE,
    SMTP_DATA,
    SMTP_DATA_ACK,
    SMTP_DATA_HEADER,
    SMTP_DATA_BODY_INIT,
    SMTP_DATA_BODY,
    SMTP_DATA_BODY_ACK,
    SMTP_QUIT_INIT,
    SMTP_QUIT
} SMTPState;
```

**Members**

Members	Description
SMTP_HOME = 0	Idle start state for SMTP client (application is preparing message)
SMTP_HELO	HELO is being sent to server
SMTP_HELO_ACK	Received an ACK for the HELO
SMTP_AUTH_LOGIN	Requesting to log in
SMTP_AUTH_LOGIN_ACK	Log in request accepted
SMTP_AUTH_USERNAME	Sending user name
SMTP_AUTH_USERNAME_ACK	User name accepted
SMTP_AUTH_PASSWORD	Sending password
SMTP_AUTH_PASSWORD_ACK	Password was accepted
SMTP_MAILFROM	Sending initial MAIL FROM command
SMTP_MAILFROM_ACK	MAIL FROM was accepted
SMTP_RCPTTO_INIT	Preparing to send RCPT TO

SMTP_RCPTTO	Sending RCPT TO command
SMTP_RCPTTO_ACK	RCPT TO was accepted
SMTP_RCPTTO_ISDONE	Done sending RCPT TO commands
SMTP_RCPTTOCC_INIT	Preparing to send RCPT TO CC commands
SMTP_RCPTTOCC	Sending RCPT TO CC commands
SMTP_RCPTTOCC_ACK	RCPT TO CC was accepted
SMTP_RCPTTOCC_ISDONE	Done sending RCPT TO CC
SMTP_RCPTTOBCC_INIT	Preparing to send RCPT TO BCC commands
SMTP_RCPTTOBCC	Sending RCPT TO BCC commands
SMTP_RCPTTOBCC_ACK	RCPT TO BCC was accepted
SMTP_RCPTTOBCC_ISDONE	Done sending RCPT TO BCC
SMTP_DATA	Sending DATA command
SMTP_DATA_ACK	DATA command accpted
SMTP_DATA_HEADER	Sending message headers
SMTP_DATA_BODY_INIT	Preparing for message body
SMTP_DATA_BODY	Sending message body
SMTP_DATA_BODY_ACK	Message body accepted
SMTP_QUIT_INIT	Sending QUIT command
SMTP_QUIT	QUIT accepted, connection closing

**Description**

Message state machine for the SMTP Client

### 10.13.4.13 TransportState Variable

**File**

SMTP.c

**C**

```
enum {
    TRANSPORT_HOME = 0,
    TRANSPORT_BEGIN,
    TRANSPORT_NAME_RESOLVE,
    TRANSPORT_OBTAIN_SOCKET,
    TRANSPORT_SECURING_SOCKET,
    TRANSPORT_SOCKET_OBTAINED,
    TRANSPORT_CLOSE
} TransportState;
```

**Members**

Members	Description
TRANSPORT_HOME = 0	Idle state
TRANSPORT_BEGIN	Preparing to make connection
TRANSPORT_NAME_RESOLVE	Resolving the SMTP server address
TRANSPORT_OBTAIN_SOCKET	Obtaining a socket for the SMTP connection
TRANSPORT_SECURING_SOCKET	Securing the socket for the SMTP over SSL connection
TRANSPORT_SOCKET_OBTAINED	SMTP connection successful
TRANSPORT_CLOSE	STMP socket is closed

**Description**

State of the transport for the SMTP Client

# 10.14 Reboot

The Reboot module will allow a user to remotely reboot the PIC microcontroller that is running the TCP/IP stack. This feature is primarily used for bootloader applications, which must reset the microcontroller to enter the bootloader code section. This module will execute a task that listens on a specified UDP port for a packet, and then reboots if it receives one. The port can be configured in `Reboot.c` with the following macro:

```
#define REBOOT_PORT 69
```

For improved security, you can limit reboot capabilities to users on the same subnet by specifying the following macro in `Reboot.c`:

```
#define REBOOT_SAME_SUBNET_ONLY
```

## 10.14.1 Reboot Stack Members

### Functions

	Name	Description
	RebootTask ( <a href="#">see page 311</a> )	Checks for incoming traffic on port 69. Resets the PIC if a 'R' is received.

### Macros

	Name	Description
	REBOOT_PORT ( <a href="#">see page 312</a> )	UDP TFTP port
	REBOOT_SAME_SUBNET_ONLY ( <a href="#">see page 312</a> )	For improved security, you might want to limit reboot capabilities to only users on the same IP subnet. Define REBOOT_SAME_SUBNET_ONLY to enable this access restriction.

### Module

Reboot ([see page 311](#))

### Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.14.1.1 RebootTask Function

#### File

Reboot.h

#### C

```
void RebootTask();
```

#### Side Effects

None

#### Returns

None

#### Description

Checks for incoming traffic on port 69. Resets the PIC if a 'R' is received.

**Remarks**

This module is primarily for use with the Ethernet bootloader. By resetting, the Ethernet bootloader can take control for a second and let a firmware upgrade take place.

**Preconditions**

Stack is initialized()

## 10.14.1.2 REBOOT\_PORT Macro

**File**

Reboot.c

**C**

```
#define REBOOT_PORT 69 // UDP TFTP port
```

**Description**

UDP TFTP port

## 10.14.1.3 REBOOT\_SAME\_SUBNET\_ONLY Macro

**File**

Reboot.c

**C**

```
#define REBOOT_SAME_SUBNET_ONLY
```

**Description**

For improved security, you might want to limit reboot capabilities to only users on the same IP subnet. Define REBOOT\_SAME\_SUBNET\_ONLY to enable this access restriction.

# 10.15 SNMP

**Functions**

	Name	Description
◆	getSnmV2GenTrapOid (see page 356)	Resolves generic trap code to generic trap OID.
◆	ProcessGetBulkVar (see page 356)	This is function ProcessGetBulkVar.
◆	ProcessGetNextVar (see page 357)	This is function ProcessGetNextVar.
◆	ProcessGetVar (see page 357)	This is function ProcessGetVar.
◆	ProcessSnmV3MsgData (see page 357)	This is function ProcessSnmV3MsgData.
◆	SNMPGetExactIndex (see page 357)	To search for exact index node in case of a Sequence variable.
◆	SNMPIdRecrdValidation (see page 358)	Used to validate the support of Var ID for A particular SNMP Version.

	SNMPIsValidSetLen ( <a href="#">see page 359</a> )	Validates the set variable data length to data type.
	Snmpv3AESDecryptRxedScopedPdu ( <a href="#">see page 359</a> )	This is function Snmpv3AESDecryptRxedScopedPdu.
	Snmpv3BufferPut ( <a href="#">see page 359</a> )	This is function Snmpv3BufferPut.
	Snmpv3FormulateEngineID ( <a href="#">see page 360</a> )	This is function Snmpv3FormulateEngineID.
	Snmpv3GetAuthEngineTime ( <a href="#">see page 360</a> )	This is function Snmpv3GetAuthEngineTime.
	Snmpv3GetBufferData ( <a href="#">see page 360</a> )	This is function Snmpv3GetBufferData.
	Snmpv3InitializeUserDataBase ( <a href="#">see page 360</a> )	This is function Snmpv3InitializeUserDataBase.
	Snmpv3MsgProcessingModelProcessPDU ( <a href="#">see page 361</a> )	This is function Snmpv3MsgProcessingModelProcessPDU.
	Snmpv3Notify ( <a href="#">see page 361</a> )	This is function Snmpv3Notify.
	Snmpv3ScopedPduProcessing ( <a href="#">see page 361</a> )	This is function Snmpv3ScopedPduProcessing.
	Snmpv3TrapScopedpdu ( <a href="#">see page 361</a> )	This is function Snmpv3TrapScopedpdu.
	Snmpv3UserSecurityModelProcessPDU ( <a href="#">see page 362</a> )	This is function Snmpv3UserSecurityModelProcessPDU.
	Snmpv3UsmAesEncryptDecryptInitVector ( <a href="#">see page 362</a> )	This is function Snmpv3UsmAesEncryptDecryptInitVector.
	Snmpv3UsmOutMsgAuthenticationParam ( <a href="#">see page 362</a> )	This is function Snmpv3UsmOutMsgAuthenticationParam.
	Snmpv3ValidateEngineId ( <a href="#">see page 362</a> )	This is function Snmpv3ValidateEngineId.
	Snmpv3ValidateSecNameAndSecLvl ( <a href="#">see page 363</a> )	This is function Snmpv3ValidateSecNameAndSecLvl.
	Snmpv3ValidateSecurityName ( <a href="#">see page 363</a> )	This is function Snmpv3ValidateSecurityName.

**Macros**

	Name	Description
	IS_SNMPV3_AUTH_STRUCTURE ( <a href="#">see page 366</a> )	This is macro IS_SNMPV3_AUTH_STRUCTURE.
	REPORT_RESPONSE ( <a href="#">see page 367</a> )	This is macro REPORT_RESPONSE.
	SNMP_MAX_MSG_SIZE ( <a href="#">see page 367</a> )	SNMP MIN and MAX message 484 bytes in size As per RFC 3411 snmpEngineMaxMessageSize and RFC 1157 ( section 4- protocol specification ) and implementation supports more than 484 whenever feasible.
	SNMP_MAX_NON_REC_ID_OID ( <a href="#">see page 367</a> )	Update the Non record id OID value which is part of CustomSnmpDemo.c file
	SNMP_V3 ( <a href="#">see page 367</a> )	This is macro SNMP_V3.

**Types**

	Name	Description
	INOUT_SNMP_PDU ( <a href="#">see page 363</a> )	This is type INOUT_SNMP_PDU.
	SNMPNONMIBRECDINFO ( <a href="#">see page 364</a> )	This is type SNMPNONMIBRECDINFO.
	SNMPV3MSGDATA ( <a href="#">see page 364</a> )	SNMPv3

**Variables**

	<b>Name</b>	<b>Description</b>
	getZeroInstance ( <a href="#">see page 365</a> )	This variable is used for next request for zero instance
	gSnmNonMibReclInfo ( <a href="#">see page 365</a> )	Below oidStr is the collection of OID variables which are not part of MIB.h file
	gSNMPv3ScopedPduDataPos ( <a href="#">see page 365</a> )	This is variable gSNMPv3ScopedPduDataPos.
	gSNMPv3ScopedPduRequestBuf ( <a href="#">see page 365</a> )	This is variable gSNMPv3ScopedPduRequestBuf.
	gSNMPv3ScopedPduResponseBuf ( <a href="#">see page 366</a> )	This is variable gSNMPv3ScopedPduResponseBuf.
	msgSecrtyParamLenOffset ( <a href="#">see page 366</a> )	This is variable msgSecrtyParamLenOffset.

**Description**

Simple Network Management Protocol V2c (community) agent implementation of [RFC 3416](#).

## 10.15.1 SNMP Public Members

**Enumerations**

	<b>Name</b>	<b>Description</b>
	GENERIC_TRAP_NOTIFICATION_TYPE ( <a href="#">see page 316</a> )	This is type GENERIC_TRAP_NOTIFICATION_TYPE.
	VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE ( <a href="#">see page 316</a> )	This is type VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE.
	SNMP_ACTION ( <a href="#">see page 316</a> )	This is the list of SNMP action a remote NMS can perform. This information is passed to application via callback <code>SNMPValidateCommunity</code> ( <a href="#">see page 320</a> ()). Application should validate the action for given community string.
	COMMUNITY_TYPE ( <a href="#">see page 317</a> )	This is type COMMUNITY_TYPE.

**Functions**

	<b>Name</b>	<b>Description</b>
	SNMPSendTrap ( <a href="#">see page 319</a> )	Prepare, validate remote node which will receive trap and send trap pdu.
	SNMPValidateCommunity ( <a href="#">see page 320</a> )	This is function SNMPValidateCommunity.
	SNMPNotify ( <a href="#">see page 320</a> )	Creates and Sends TRAP pdu.
	SNMPSetVar ( <a href="#">see page 321</a> )	This routine Set the mib variable with the requested value.
	SNMPGetVar ( <a href="#">see page 322</a> )	Used to Get/collect OID variable information.
	SNMPisNotifyReady ( <a href="#">see page 322</a> )	Resolves given remoteHost IP address into MAC address.
	SNMPNotifyPrepare ( <a href="#">see page 323</a> )	Collects trap notification info and send ARP to remote host.
	SNMPGetNextIndex ( <a href="#">see page 324</a> )	To search for next index node in case of a Sequence variable.

**Macros**

	<b>Name</b>	<b>Description</b>
↔	SNMP_COMMUNITY_MAX_LEN (see page 325)	This is the maximum length for community string. Application must ensure that this length is observed. SNMP module adds one byte extra after SNMP_COMMUNITY_MAX_LEN for adding '0' NULL character.
↔	OID_MAX_LEN (see page 325)	Change this to match your OID string length.
↔	SNMP_START_OF_VAR (see page 325)	This is macro SNMP_START_OF_VAR.
↔	SNMP_END_OF_VAR (see page 325)	This is macro SNMP_END_OF_VAR.
↔	SNMP_INDEX_INVALID (see page 326)	This is macro SNMP_INDEX_INVALID.
↔	TRAP_TABLE_SIZE (see page 326)	Trap information. This table maintains list of interested receivers who should receive notifications when some interesting event occurs.
↔	TRAP_COMMUNITY_MAX_LEN (see page 326)	This is macro TRAP_COMMUNITY_MAX_LEN.
↔	NOTIFY_COMMUNITY_LEN (see page 326)	This is macro NOTIFY_COMMUNITY_LEN.

**Module**

SNMP (see page 312)

**Structures**

	<b>Name</b>	<b>Description</b>
🔗	TRAP_INFO (see page 318)	This is type TRAP_INFO.

**Types**

	<b>Name</b>	<b>Description</b>
🔗	SNMP_ID (see page 324)	This is the SNMP OID variable id. This id is assigned via MIB file. Only dynamic and AgentID variables can contain ID. MIB2BIB utility enforces this rules when BIB was generated.
🔗	SNMP_INDEX (see page 324)	This is type SNMP_INDEX.

**Unions**

	<b>Name</b>	<b>Description</b>
🔗	SNMP_VAL (see page 317)	This is type SNMP_VAL.

**Variables**

	<b>Name</b>	<b>Description</b>
🔗	gSendTrapFlag (see page 318)	global flag to send Trap
🔗	gSetTrapSendFlag (see page 318)	#if defined(SNMP_STACK_USE_V2_TRAP)    defined(SNMP_V1_V2_TRAP_WITH_SNMPV3) //if gSetTrapSendFlag == FALSE then the last varbind variable for //multiple varbind variable pdu structure or if there is only varbind variable send. // if gSetTrapSendFlag == TRUE, then v2 trap pdu is expecting more varbind variable. BYTE gSetTrapSendFlag = FALSE; #endif
🔗	gGenericTrapNotification (see page 319)	Global flag for Generic trap notification
🔗	gSpecificTrapNotification (see page 319)	Vendor specific trap code
🔗	gOIDCorrespondingSnmplibID (see page 319)	Global var to store SNMP ID of var for OID received in SNMP request.

**Description**

The following functions and variables are available to the stack application.

### 10.15.1.1 GENERIC\_TRAP\_NOTIFICATION\_TYPE Enumeration

**File**

SNMP.h

**C**

```
typedef enum {
    COLD_START = 0x0,
    WARM_START = 0x1,
    LINK_DOWN = 0x2,
    LINK_UP = 0x3,
    AUTH_FAILURE = 0x4,
    EGP_NEBOR_LOSS = 0x5,
    ENTERPRISE_SPECIFIC = 0x6
} GENERIC_TRAP_NOTIFICATION_TYPE;
```

**Description**

This is type GENERIC\_TRAP\_NOTIFICATION\_TYPE.

### 10.15.1.2 VENDOR\_SPECIFIC\_TRAP\_NOTIFICATION\_TYPE Enumeration

**File**

SNMP.h

**C**

```
typedef enum {
    VENDOR_TRAP_DEFAULT = 0x0,
    BUTTON_PUSH_EVENT = 0x1,
    POT_READING_MORE_512 = 0x2
} VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE;
```

**Description**

This is type VENDOR\_SPECIFIC\_TRAP\_NOTIFICATION\_TYPE.

### 10.15.1.3 SNMP\_ACTION Enumeration

**File**

SNMP.h

**C**

```
typedef enum {
    SNMP_GET = 0xa0,
    SNMP_GET_NEXT = 0xa1,
    SNMP_GET_RESPONSE = 0xa2,
    SNMP_SET = 0xa3,
    SNMP_TRAP = 0xa4,
    SNMP_V2C_GET_BULK = 0xa5,
    SNMP_V2_TRAP = 0xa7,
    SNMPV3_ENCRYPTION = 0x04,
    SNMP_ACTION_UNKNOWN = 0
} SNMP_ACTION;
```

**Members**

Members	Description
SNMP_GET = 0xa0	Snmp GET identifier
SNMP_GET_NEXT = 0xa1	Snmp GET_NEXT identifier
SNMP_GET_RESPONSE = 0xa2	Snmp GET_RESPONSE (see page 334) identifier
SNMP_SET = 0xa3	Snmp SET identifier
SNMP_TRAP = 0xa4	Snmp TRAP identifier
SNMP_V2C_GET_BULK = 0xa5	Snmp GET_BULK identifier
SNMP_V2_TRAP = 0xa7	Snmp v2 Trap Identifier
SNMP_ACTION_UNKNOWN = 0	Snmp requested action unknown

**Description**

This is the list of SNMP action a remote NMS can perform. This information is passed to application via callback `SNMPValidateCommunity` (see page 320). Application should validate the action for given community string.

### 10.15.1.4 COMMUNITY\_TYPE Enumeration

**File**

SNMP.h

**C**

```
typedef enum {
    READ_COMMUNITY = 1,
    WRITE_COMMUNITY = 2,
    INVALID_COMMUNITY = 3
} COMMUNITY_TYPE;
```

**Members**

Members	Description
READ_COMMUNITY = 1	Read only community
WRITE_COMMUNITY = 2	Read write community
INVALID_COMMUNITY = 3	Community invalid

**Description**

This is type COMMUNITY\_TYPE.

### 10.15.1.5 SNMP\_VAL Union

**File**

SNMP.h

**C**

```
typedef union {
    DWORD dword;
    WORD word;
    BYTE byte;
    BYTE v[sizeof(DWORD)];
} SNMP_VAL;
```

**Members**

Members	Description
DWORD dword;	double word value
WORD word;	word value

BYTE byte;	byte value
BYTE v[sizeof(DWORD)];	byte array

**Description**

This is type SNMP\_VAL.

## 10.15.1.6 TRAP\_INFO Structure

**File**

SNMP.h

**C**

```
typedef struct {
    BYTE Size;
    struct {
        BYTE communityLen;
        char community[TRAP_COMMUNITY_MAX_LEN];
        IP_ADDR IPAddress;
        struct {
            unsigned int bEnabled : 1;
        } Flags;
    } table[TRAP_TABLE_SIZE];
} TRAP_INFO;
```

**Members**

Members	Description
BYTE communityLen;	Community name length
char community[TRAP_COMMUNITY_MAX_LEN];	Community name array
IP_ADDR IPAddress;	IP address to which trap to be sent
unsigned int bEnabled : 1;	Trap enabled flag

**Description**

This is type TRAP\_INFO.

## 10.15.1.7 gSendTrapFlag Variable

**File**

CustomSNMPApp.c

**C**

```
BYTE gSendTrapFlag = FALSE;
```

**Description**

global flag to send Trap

## 10.15.1.8 gSetTrapSendFlag Variable

**File**

CustomSNMPApp.c

**C**

```
BYTE gSetTrapSendFlag = FALSE;
```

**Description**

```
#if defined(SNMP_STACK_USE_V2_TRAP) || defined(SNMP_V1_V2_TRAP_WITH_SNMPV3) //if gSetTrapSendFlag ==
FALSE then the last varbind variable for //multiple varbind variable pdu structure or if there is only varbind variable send. // if
gSetTrapSendFlag == TRUE, then v2 trap pdu is expecting more varbind variable. BYTE gSetTrapSendFlag = FALSE;
#endif
```

## 10.15.1.9 gGenericTrapNotification Variable

**File**

CustomSNMPApp.c

**C**

```
BYTE gGenericTrapNotification = ENTERPRISE_SPECIFIC;
```

**Description**

Global flag for Generic trap notification

## 10.15.1.10 gSpecificTrapNotification Variable

**File**

CustomSNMPApp.c

**C**

```
BYTE gSpecificTrapNotification = VENDOR_TRAP_DEFAULT;
```

**Description**

Vendor specific trap code

## 10.15.1.11 gOIDCorrespondingSnmplibID Variable

**File**

CustomSNMPApp.c

**C**

```
BYTE gOIDCorrespondingSnmplibID = MICROCHIP;
```

**Description**

Global var to store SNMP ID of var for OID received in SNMP request.

## 10.15.1.12 SNMPSTransmitTrap Function

**File**

SNMP.h

**C**

```
void SNMPSTransmitTrap();
```

**Returns**

None.

**Description**

This function is used to send trap notification to previously configured ip address if trap notification is enabled. There are different trap notification code. The current implementation sends trap for authentication failure (4).

**Remarks**

This is a callback function called by the application on certain predefined events. This routine only implemented to send a authentication failure Notification-type macro with PUSH\_BUTTON oid stored in MPFS. If the ARP is no resolved i.e. if `SNMP_IsNotifyReady` (see page 322)() returns FALSE, this routine times out in 5 seconds. This routine should be modified according to event occurred and should update corresponding OID and notification type to the trap pdu.

**Preconditions**

If application defined event occurs to send the trap.

## 10.15.1.13 SNMPValidateCommunity Function

**File**

SNMP.h

**C**

```
BYTE SNMPValidateCommunity(
    BYTE* community
);
```

**Description**

This is function `SNMPValidateCommunity`.

## 10.15.1.14 SNMPNotify Function

**File**

SNMP.h

**C**

```
BOOL SNMPNotify(
    SNMP_ID var,
    SNMP_VAL val,
    SNMP_INDEX index
);
```

**Description**

This function creates SNMP trap PDU and sends it to previously specified remoteHost. snmpv1 trap pdu: | PDU-type | enterprise | agent-addr | generic-trap | specific-trap | | time-stamp | varbind-list |

The v1 enterprise is mapped directly to `SNMPv2TrapOID.0`

**Remarks**

This would fail if there were not UDP socket to open.

**Preconditions**

`SNMP_IsNotifyReady` (see page 322)() is already called and returned TRUE.

**Parameters**

Parameters	Description
var	SNMP var ID that is to be used in notification
val	Value of var. Only value of BYTE, WORD or DWORD can be sent.

index	Index of var. If this var is a single, index would be 0, or else if this var is a sequence, index could be any value from 0 to 127
-------	--

**Return Values**

Return Values	Description
TRUE	if SNMP notification was successful sent. This does not guarantee that remoteHost recieved it.
FALSE	Notification sent failed.
This would fail under following contions	1) Given SNMP_BIB_FILE does not exist in MPFS 2) Given var does not exist. 3) Previously given agentID does not exist
4) Data type of given var is unknown	only possible if MPFS itself was corrupted.

## 10.15.1.15 SNMPSetVar Function

**File**

SNMP.h

**C**

```

BOOL SNMPSetVar(
    SNMP_ID var,
    SNMP_INDEX index,
    BYTE ref,
    SNMP_VAL val
);
    
```

**Description**

This is a callback function called by module for the snmp SET request. User application must modify this function for the new variables address.

**Remarks**

This function may get called more than once depending on number of bytes in a specific set request for given variable. only dynamic read-write variables needs to be handled.

**Preconditions**

ProcessVariables (see page 352)() is called.

**Parameters**

Parameters	Description
var	Variable id whose value is to be set
ref	Variable reference used to transfer multi-byte data 0 if first byte is set otherwise nonzero value to indicate corresponding byte being set.
val	Up to 4 byte data value. If var data type is BYTE, variable value is in val->byte If var data type is WORD, variable value is in val->word If var data type is DWORD, variable value is in val->dword. If var data type is IP_ADDRESS, COUNTER32, or GAUGE32, value is in val->dword If var data type is OCTET_STRING (see page 339), ASCII_STRING value is in val->byte; multi-byte transfer will be performed to transfer remaining bytes of data.

**Return Values**

Return Values	Description
TRUE	if it is OK to set more byte(s).
FALSE	if otherwise.

## 10.15.1.16 SNMPGetVar Function

### File

SNMP.h

### C

```

BOOL SNMPGetVar(
    SNMP_ID var,
    SNMP_INDEX index,
    BYTE* ref,
    SNMP_VAL* val
);
    
```

### Description

This is a callback function called by SNMP module. SNMP user must implement this function in user application and provide appropriate data when called.

### Remarks

None.

### Preconditions

None

### Parameters

Parameters	Description
var	Variable id whose value is to be returned
index	Index of variable that should be transferred
ref	Variable reference used to transfer multi-byte data It is always SNMP_START_OF_VAR (see page 325) when very first byte is requested. Otherwise, use this as a reference to keep track of multi-byte transfers.
val	Pointer to up to 4 byte buffer. If var data type is BYTE, transfer data in val->byte If var data type is WORD, transfer data in val->word If var data type is DWORD, transfer data in val->dword If var data type is IP_ADDRESS, transfer data in val->v[] or val->dword If var data type is COUNTER32, TIME_TICKS or GAUGE32, transfer data in val->dword If var data type is ASCII_STRING or OCTET_STRING (see page 339) transfer data in val->byte using multi-byte transfer mechanism.

### Return Values

Return Values	Description
TRUE	If a value exists for given variable at given index.
FALSE	Otherwise.

## 10.15.1.17 SNMPIsNotifyReady Function

### File

SNMP.h

### C

```

BOOL SNMPIsNotifyReady(
    IP_ADDR* remoteHost
);
    
```

### Description

This function resolves given remoteHost IP address into MAC address using ARP module. If remoteHost is not available, this function would never return TRUE. Application must implement timeout logic to handle "remoteHost not available"

situation.

#### Remarks

This would fail if there were not UDP socket to open.

#### Preconditions

SNMPNotifyPrepare (see page 323)() is already called.

#### Parameters

Parameters	Description
remoteHost	Pointer to remote Host IP address

#### Return Values

Return Values	Description
TRUE	If remoteHost IP address is resolved and SNMPNotify (see page 320) may be called.
FALSE	If remoteHost IP address is not resolved.

## 10.15.1.18 SNMPNotifyPrepare Function

#### File

SNMP.h

#### C

```
void SNMPNotifyPrepare(
    IP_ADDR* remoteHost,
    char* community,
    BYTE communityLen,
    SNMP_ID agentIDVar,
    BYTE notificationCode,
    DWORD timestamp
);
```

#### Returns

None

#### Description

This function prepares SNMP module to send SNMP trap notification to remote host. It sends ARP request to remote host to learn remote host MAC address.

#### Remarks

This is first of series of functions to complete SNMP notification.

#### Preconditions

SNMPInit (see page 354)() is already called.

#### Parameters

Parameters	Description
remoteHost	pointer to remote Host IP address
community	Community string to use to notify
communityLen	Community string length
agentIDVar	System ID to use identify this agent
notificaitonCode	Notification Code to use
timestamp	Notification timestamp in 100th of second.

## 10.15.1.19 SNMPGetNextIndex Function

### File

SNMP.h

### C

```

BOOL SNMPGetNextIndex(
    SNMP_ID var,
    SNMP_INDEX* index
);

```

### Description

This is a callback function called by SNMP module. SNMP user must implement this function in user application and provide appropriate data when called. This function will only be called for OID variable of type sequence.

### Remarks

Only sequence index needs to be handled in this function.

### Preconditions

None

### Parameters

Parameters	Description
var	Variable id whose value is to be returned
index	Next Index of variable that should be transferred

### Return Values

Return Values	Description
TRUE	If a next index value exists for given variable at given index and index parameter contains next valid index.
FALSE	Otherwise.

## 10.15.1.20 SNMP\_ID Type

### File

SNMP.h

### C

```

typedef int SNMP_ID;

```

### Description

This is the SNMP OID variable id. This id is assigned via MIB file. Only dynamic and AgentID variables can contain ID. MIB2BIB utility enforces this rules when BIB was generated.

## 10.15.1.21 SNMP\_INDEX Type

### File

SNMP.h

### C

```

typedef BYTE SNMP_INDEX;

```

**Description**

This is type SNMP\_INDEX.

## 10.15.1.22 SNMP\_COMMUNITY\_MAX\_LEN Macro

**File**

TCPIP MRF24WB.h

**C**

```
#define SNMP_COMMUNITY_MAX_LEN (8u)
```

**Description**

This is the maximum length for community string. Application must ensure that this length is observed. SNMP module adds one byte extra after SNMP\_COMMUNITY\_MAX\_LEN for adding '0' NULL character.

## 10.15.1.23 OID\_MAX\_LEN Macro

**File**

SNMP.h

**C**

```
#define OID_MAX_LEN (18)
```

**Description**

Change this to match your OID string length.

## 10.15.1.24 SNMP\_START\_OF\_VAR Macro

**File**

SNMP.h

**C**

```
#define SNMP_START_OF_VAR (0)
```

**Description**

This is macro SNMP\_START\_OF\_VAR.

## 10.15.1.25 SNMP\_END\_OF\_VAR Macro

**File**

SNMP.h

**C**

```
#define SNMP_END_OF_VAR (0xff)
```

**Description**

This is macro SNMP\_END\_OF\_VAR.

## 10.15.1.26 SNMP\_INDEX\_INVALID Macro

### File

SNMP.h

### C

```
#define SNMP_INDEX_INVALID (0xff)
```

### Description

This is macro SNMP\_INDEX\_INVALID.

## 10.15.1.27 TRAP\_TABLE\_SIZE Macro

### File

SNMP.h

### C

```
#define TRAP_TABLE_SIZE (2)
```

### Description

Trap information. This table maintains list of interested receivers who should receive notifications when some interesting event occurs.

## 10.15.1.28 TRAP\_COMMUNITY\_MAX\_LEN Macro

### File

SNMP.h

### C

```
#define TRAP_COMMUNITY_MAX_LEN (8)
```

### Description

This is macro TRAP\_COMMUNITY\_MAX\_LEN.

## 10.15.1.29 NOTIFY\_COMMUNITY\_LEN Macro

### File

TCPIP MRF24WB.h

### C

```
#define NOTIFY_COMMUNITY_LEN (SNMP_COMMUNITY_MAX_LEN)
```

### Description

This is macro NOTIFY\_COMMUNITY\_LEN.

## 10.15.2 SNMP Internal Members

### Enumerations

	Name	Description
	DATA_TYPE ( <a href="#">see page 332</a> )	
	SNMP_ERR_STATUS ( <a href="#">see page 342</a> )	

### Functions

	Name	Description
	_SNMPDuplexInit ( <a href="#">see page 330</a> )	
	_SNMPGet ( <a href="#">see page 330</a> )	This is function _SNMPGet.
	_SNMPPut ( <a href="#">see page 330</a> )	
	FindOIDsInRequest ( <a href="#">see page 334</a> )	This is function FindOIDsInRequest.
	GetDataTypeInfo ( <a href="#">see page 337</a> )	This is function GetDataTypeInfo.
	IsASNNull ( <a href="#">see page 338</a> )	static BYTE IsValidStructure ( <a href="#">see page 350</a> )(WORD* dataLen);
	SetErrorStatus ( <a href="#">see page 341</a> )	This is function SetErrorStatus.
	IsValidLength ( <a href="#">see page 345</a> )	This is function IsValidLength.
	GetDataTypeInfo ( <a href="#">see page 348</a> )	This is function GetDataTypeInfo.
	GetNextLeaf ( <a href="#">see page 348</a> )	This is function GetNextLeaf.
	GetOIDStringByAddr ( <a href="#">see page 349</a> )	This is function GetOIDStringByAddr.
	GetOIDStringByID ( <a href="#">see page 349</a> )	This function is used only when TRAP is enabled.
	IsValidCommunity ( <a href="#">see page 349</a> )	This is function IsValidCommunity.
	IsValidInt ( <a href="#">see page 349</a> )	This is function IsValidInt.
	IsValidLength ( <a href="#">see page 350</a> )	This is function IsValidLength.
	IsValidOID ( <a href="#">see page 350</a> )	
	IsValidPDU ( <a href="#">see page 350</a> )	static BOOL IsValidInt ( <a href="#">see page 349</a> )(DWORD* val);
	IsValidStructure ( <a href="#">see page 350</a> )	This is function IsValidStructure.
	OIDLookup ( <a href="#">see page 351</a> )	
	ProcessGetSetHeader ( <a href="#">see page 351</a> )	Validates the received udp packet Get/Set request header.
	ProcessHeader ( <a href="#">see page 352</a> )	Validates the received udp packet Snmp header.
	ProcessSetVar ( <a href="#">see page 352</a> )	This is function ProcessSetVar.
	ProcessVariables ( <a href="#">see page 352</a> )	This routine processes the snmp request and parallelly creates the response pdu.

	ReadMIBRecord ( <a href="#">see page 353</a> )	This is function ReadMIBRecord.
	SNMPCheckIfPvtMibObjRequested ( <a href="#">see page 353</a> )	

**Macros**

	<b>Name</b>	<b>Description</b>
	_SNMPGetTxOffset ( <a href="#">see page 330</a> )	This is macro _SNMPGetTxOffset.
	_SNMPSetTxOffset ( <a href="#">see page 331</a> )	
	AGENT_NOTIFY_PORT ( <a href="#">see page 331</a> )	This is macro AGENT_NOTIFY_PORT.
	ASN_INT ( <a href="#">see page 331</a> )	This is macro ASN_INT.
	ASN_NULL ( <a href="#">see page 332</a> )	This is macro ASN_NULL.
	ASN_OID ( <a href="#">see page 332</a> )	This is macro ASN_OID.
	DATA_TYPE_TABLE_SIZE ( <a href="#">see page 333</a> )	
	GET_BULK_REQUEST ( <a href="#">see page 334</a> )	This is macro GET_BULK_REQUEST.
	GET_NEXT_REQUEST ( <a href="#">see page 334</a> )	This is macro GET_NEXT_REQUEST.
	GET_REQUEST ( <a href="#">see page 334</a> )	
	GET_RESPONSE ( <a href="#">see page 334</a> )	This is macro GET_RESPONSE.
	IS_AGENT_PDU ( <a href="#">see page 335</a> )	This is macro IS_AGENT_PDU.
	IS_ASN_INT ( <a href="#">see page 336</a> )	This is macro IS_ASN_INT.
	IS_ASN_NULL ( <a href="#">see page 336</a> )	This is macro IS_ASN_NULL.
	IS_GET_NEXT_REQUEST ( <a href="#">see page 336</a> )	This is macro IS_GET_NEXT_REQUEST.
	IS_GET_REQUEST ( <a href="#">see page 336</a> )	This is macro IS_GET_REQUEST.
	IS_GET_RESPONSE ( <a href="#">see page 336</a> )	This is macro IS_GET_RESPONSE.
	IS_OCTET_STRING ( <a href="#">see page 337</a> )	This is macro IS_OCTET_STRING.
	IS_OID ( <a href="#">see page 337</a> )	This is macro IS_OID.
	IS_SET_REQUEST ( <a href="#">see page 337</a> )	This is macro IS_SET_REQUEST.
	IS_STRUCTURE ( <a href="#">see page 338</a> )	
	IS_TRAP ( <a href="#">see page 338</a> )	This is macro IS_TRAP.
	OCTET_STRING ( <a href="#">see page 339</a> )	This is macro OCTET_STRING.
	SET_REQUEST ( <a href="#">see page 341</a> )	This is macro SET_REQUEST.
	SNMP_AGENT_PORT ( <a href="#">see page 341</a> )	
	SNMP_BIB_FILE_NAME ( <a href="#">see page 342</a> )	This is the file that contains SNMP bib file. File name must contain all upper case letter and must match with what was included in MPFS2 image.

	SNMP_COUNTER32 ( <a href="#">see page 342</a> )	This is macro SNMP_COUNTER32.
	SNMP_GAUGE32 ( <a href="#">see page 343</a> )	This is macro SNMP_GAUGE32.
	SNMP_IP_ADDR ( <a href="#">see page 343</a> )	This is macro SNMP_IP_ADDR.
	SNMP_NMS_PORT ( <a href="#">see page 344</a> )	This is macro SNMP_NMS_PORT.
	SNMP_NSAP_ADDR ( <a href="#">see page 344</a> )	This is macro SNMP_NSAP_ADDR.
	SNMP_OPAQUE ( <a href="#">see page 345</a> )	This is macro SNMP_OPAQUE.
	SNMP_TIME_TICKS ( <a href="#">see page 345</a> )	This is macro SNMP_TIME_TICKS.
	SNMP_V1 ( <a href="#">see page 346</a> )	
	SNMP_V2C ( <a href="#">see page 346</a> )	This is macro SNMP_V2C.
	STRUCTURE ( <a href="#">see page 347</a> )	
	TRAP ( <a href="#">see page 348</a> )	This is macro TRAP.

**Module**

SNMP ([see page 312](#))

**Structures**

	Name	Description
	DATA_TYPE_INFO ( <a href="#">see page 333</a> )	
	OID_INFO ( <a href="#">see page 339</a> )	
	PDU_INFO ( <a href="#">see page 340</a> )	
	reqVarErrStatus ( <a href="#">see page 340</a> )	
	SNMP_NOTIFY_INFO ( <a href="#">see page 344</a> )	

**Unions**

	Name	Description
	INDEX_INFO ( <a href="#">see page 335</a> )	
	MIB_INFO ( <a href="#">see page 338</a> )	
	SNMP_STATUS ( <a href="#">see page 345</a> )	

**Variables**

	Name	Description
	appendZeroToOID ( <a href="#">see page 331</a> )	global flag to modify OID by appending zero
	dataTypeTable ( <a href="#">see page 333</a> )	ASN format datatype for snmp v1 and v2c
	hMPFS ( <a href="#">see page 335</a> )	MPFS file handler
	SNMPAgentSocket ( <a href="#">see page 346</a> )	Snmp udp socket
	SNMPNotifyInfo ( <a href="#">see page 346</a> )	notify info for trap
	snmpReqVarErrStatus ( <a href="#">see page 347</a> )	vars from req list processing err status

	SNMPRxOffset ( <a href="#">see page 347</a> )	Snmp udp buffer rx offset
	SNMPStatus ( <a href="#">see page 347</a> )	MIB file access status
	SNMPTxOffset ( <a href="#">see page 347</a> )	Snmp udp buffer tx offset
	trapInfo ( <a href="#">see page 348</a> )	Initialize trap table with no entries.

**Description**

The following functions and variables are designated as internal to the SNMP module.

## 10.15.2.1 **\_SNMPDuplexInit Function**

**File**

SNMP.c

**C**

```
void _SNMPDuplexInit(
    UDP_SOCKET socket
);
```

**Section**

Global variables configuration for pdu processings

## 10.15.2.2 **\_SNMPGet Function**

**File**

SNMP.h

**C**

```
BYTE _SNMPGet();
```

**Description**

This is function \_SNMPGet.

## 10.15.2.3 **\_SNMPGetTxOffset Macro**

**File**

SNMP.h

**C**

```
#define _SNMPGetTxOffset SNMPTxOffset
```

**Description**

This is macro \_SNMPGetTxOffset.

## 10.15.2.4 **\_SNMPPut Function**

**File**

SNMP.c

**C**

```
void _SNMPPut(  
    BYTE v  
);
```

**Section**

Process SNMP request pdus, form response pdus routines

```
static BYTE _SNMPGet(void);
```

## 10.15.2.5 `_SNMPSetTxOffset` Macro

**File**

SNMP.h

**C**

```
#define _SNMPSetTxOffset(o) (SNMPTxOffset = o)
```

**Section**

SNMP Tx pdu offset settings

## 10.15.2.6 `AGENT_NOTIFY_PORT` Macro

**File**

SNMP.h

**C**

```
#define AGENT_NOTIFY_PORT (0xfffe)
```

**Description**

This is macro `AGENT_NOTIFY_PORT`.

## 10.15.2.7 `appendZeroToOID` Variable

**File**

SNMP.c

**C**

```
BYTE appendZeroToOID;
```

**Description**

global flag to modify OID by appending zero

## 10.15.2.8 `ASN_INT` Macro

**File**

SNMP.h

**C**

```
#define ASN_INT (0x02u)
```

**Description**

This is macro `ASN_INT`.

## 10.15.2.9 ASN\_NULL Macro

### File

SNMP.h

### C

```
#define ASN_NULL (0x05u)
```

### Description

This is macro ASN\_NULL.

## 10.15.2.10 ASN\_OID Macro

### File

SNMP.h

### C

```
#define ASN_OID (0x06u)
```

### Description

This is macro ASN\_OID.

## 10.15.2.11 DATA\_TYPE Enumeration

### File

SNMP.h

### C

```
typedef enum {
    INT8_VAL = 0x00,
    INT16_VAL = 0x01,
    INT32_VAL = 0x02,
    BYTE_ARRAY = 0x03,
    ASCII_STRING = 0x04,
    IP_ADDRESS = 0x05,
    COUNTER32 = 0x06,
    TIME_TICKS_VAL = 0x07,
    GAUGE32 = 0x08,
    OID_VAL = 0x09,
    DATA_TYPE_UNKNOWN
} DATA_TYPE;
```

### Members

Members	Description
INT8_VAL = 0x00	8 bit integer value
INT16_VAL = 0x01	16 bit integer value
INT32_VAL = 0x02	32 bit integer value
BYTE_ARRAY = 0x03	Array of bytes
ASCII_STRING = 0x04	Ascii string type
IP_ADDRESS = 0x05	IP address variable
COUNTER32 = 0x06	32 bit counter variable
TIME_TICKS_VAL = 0x07	Timer value counter variable
GAUGE32 = 0x08	32 bit gauge variable
OID_VAL = 0x09	Object id value var

DATA_TYPE_UNKNOWN	Unknown data type
-------------------	-------------------

**Section**

Data Structures and Enumerations

\*\*\*\*\*

SNMP specific data types

## 10.15.2.12 DATA\_TYPE\_INFO Structure

**File**

SNMP.h

**C**

```
typedef struct {
    BYTE asnType;
    BYTE asnLen;
} DATA_TYPE_INFO;
```

**Members**

Members	Description
BYTE asnType;	ASN data type
BYTE asnLen;	ASN data length

**Section**

ASN data type info

## 10.15.2.13 DATA\_TYPE\_TABLE\_SIZE Macro

**File**

SNMP.h

**C**

```
#define DATA_TYPE_TABLE_SIZE (sizeof(dataTypeTable)/sizeof(dataTypeTable[0]))
```

**Section**

Macros and Definitions

## 10.15.2.14 dataTypeTable Variable

**File**

SNMP.c

**C**

```
ROM DATA_TYPE_INFO dataTypeTable[] = { { ASN_INT, 1 }, { ASN_INT, 2 }, { ASN_INT, 4 }, {
OCTET_STRING, 0xff }, { OCTET_STRING, 0xff }, { SNMP_IP_ADDR, 4 }, { SNMP_COUNTER32, 4 }, {
SNMP_TIME_TICKS, 4 }, { SNMP_GAUGE32, 4 }, { ASN_OID, 0xff } };
```

**Description**

ASN format datatype for snmp v1 and v2c

## 10.15.2.15 FindOIDsInRequest Function

### File

SNMP.c

### C

```
static BYTE FindOIDsInRequest(  
    WORD pduLen  
);
```

### Description

This is function FindOIDsInRequest.

## 10.15.2.16 GET\_BULK\_REQUEST Macro

### File

SNMP.h

### C

```
#define GET_BULK_REQUEST (0xa5)
```

### Description

This is macro GET\_BULK\_REQUEST.

## 10.15.2.17 GET\_NEXT\_REQUEST Macro

### File

SNMP.h

### C

```
#define GET_NEXT_REQUEST (0xa1)
```

### Description

This is macro GET\_NEXT\_REQUEST.

## 10.15.2.18 GET\_REQUEST Macro

### File

SNMP.h

### C

```
#define GET_REQUEST (0xa0)
```

### Section

SNMP v1 and v2c pdu types

## 10.15.2.19 GET\_RESPONSE Macro

### File

SNMP.h

**C**

```
#define GET_RESPONSE (0xa2)
```

**Description**

This is macro GET\_RESPONSE.

## 10.15.2.20 hMPFS Variable

**File**

SNMP.c

**C**

```
MPFS_HANDLE hMPFS;
```

**Description**

MPFS file handler

## 10.15.2.21 INDEX\_INFO Union

**File**

SNMP.h

**C**

```
typedef union {
    struct {
        unsigned int bIsOID : 1;
    } Flags;
    BYTE Val;
} INDEX_INFO;
```

**Members**

Members	Description
unsigned int bIsOID : 1;	value is OID/index int flag
BYTE Val;	value is OID/index byte flag

**Section**

SNMP OID index information

## 10.15.2.22 IS\_AGENT\_PDU Macro

**File**

SNMP.h

**C**

```
#define IS_AGENT_PDU(a) (a==GET_REQUEST || \
                        a==GET_NEXT_REQUEST || \
                        a==SET_REQUEST || \
                        a==SNMP_V2C_GET_BULK)
```

**Description**

This is macro IS\_AGENT\_PDU.

## 10.15.2.23 IS\_ASN\_INT Macro

### File

SNMP.h

### C

```
#define IS_ASN_INT(a) (a==ASN_INT)
```

### Description

This is macro IS\_ASN\_INT.

## 10.15.2.24 IS\_ASN\_NULL Macro

### File

SNMP.h

### C

```
#define IS_ASN_NULL(a) (a==ASN_NULL)
```

### Description

This is macro IS\_ASN\_NULL.

## 10.15.2.25 IS\_GET\_NEXT\_REQUEST Macro

### File

SNMP.h

### C

```
#define IS_GET_NEXT_REQUEST(a) (a==GET_NEXT_REQUEST)
```

### Description

This is macro IS\_GET\_NEXT\_REQUEST.

## 10.15.2.26 IS\_GET\_REQUEST Macro

### File

SNMP.h

### C

```
#define IS_GET_REQUEST(a) (a==GET_REQUEST)
```

### Description

This is macro IS\_GET\_REQUEST.

## 10.15.2.27 IS\_GET\_RESPONSE Macro

### File

SNMP.h

**C**

```
#define IS_GET_RESPONSE(a) (a==GET_RESPONSE)
```

**Description**

This is macro IS\_GET\_RESPONSE.

## 10.15.2.28 IS\_OCTET\_STRING Macro

**File**

SNMP.h

**C**

```
#define IS_OCTET_STRING(a) (a==OCTET_STRING)
```

**Description**

This is macro IS\_OCTET\_STRING.

## 10.15.2.29 IS\_OID Macro

**File**

SNMP.h

**C**

```
#define IS_OID(a) (a==ASN_OID)
```

**Description**

This is macro IS\_OID.

## 10.15.2.30 GetDataTypeInfo Function

**File**

SNMP.h

**C**

```
BOOL GetDataTypeInfo(  
    DATA_TYPE dataType,  
    DATA_TYPE_INFO * info  
);
```

**Description**

This is function GetDataTypeInfo.

## 10.15.2.31 IS\_SET\_REQUEST Macro

**File**

SNMP.h

**C**

```
#define IS_SET_REQUEST(a) (a==SET_REQUEST)
```

**Description**

This is macro IS\_SET\_REQUEST.

## 10.15.2.32 IS\_STRUCTURE Macro

### File

SNMP.h

### C

```
#define IS_STRUCTURE(a) (a==STRUCTURE)
```

### Section

SNMP specific data validation

## 10.15.2.33 IS\_TRAP Macro

### File

SNMP.h

### C

```
#define IS_TRAP(a) (a==TRAP)
```

### Description

This is macro IS\_TRAP.

## 10.15.2.34 IsASNNull Function

### File

SNMP.c

### C

```
static BOOL IsASNNull();
```

### Description

static BYTE IsValidStructure (see page 350)(WORD\* dataLen);

## 10.15.2.35 MIB\_INFO Union

### File

SNMP.h

### C

```
typedef union {  
    struct {  
        unsigned int bIsDistantSibling : 1;  
        unsigned int bIsConstant : 1;  
        unsigned int bIsSequence : 1;  
        unsigned int bIsSibling : 1;  
        unsigned int bIsParent : 1;  
        unsigned int bIsEditable : 1;  
        unsigned int bIsAgentID : 1;  
        unsigned int bIsIDPresent : 1;  
    } Flags;  
    BYTE Val;  
} MIB_INFO;
```

**Members**

Members	Description
unsigned int blsDistantSibling : 1;	Object have distant sibling node
unsigned int blsConstant : 1;	Object is constant
unsigned int blsSequence : 1;	Object is sequence
unsigned int blsSibling : 1;	Sibling node flag
unsigned int blsParent : 1;	Node is parent flag
unsigned int blsEditable : 1;	Node is editable flag
unsigned int blsAgentID : 1;	Node have agent id flag
unsigned int blsIDPresent : 1;	Id present flag
BYTE Val;	MIB Obj info as byte value

**Section**

SNMP object information

### 10.15.2.36 OCTET\_STRING Macro

**File**

SNMP.h

**C**

```
#define OCTET_STRING (0x04u)
```

**Description**

This is macro OCTET\_STRING.

### 10.15.2.37 OID\_INFO Structure

**File**

SNMP.h

**C**

```
typedef struct {
    DWORD hNode;
    BYTE oid;
    MIB_INFO nodeInfo;
    DATA_TYPE dataType;
    SNMP_ID id;
    WORD_VAL dataLen;
    DWORD hData;
    DWORD hSibling;
    DWORD hChild;
    BYTE index;
    BYTE indexLen;
} OID_INFO;
```

**Members**

Members	Description
DWORD hNode;	Node location in the mib
BYTE oid;	Object Id
MIB_INFO nodeInfo;	Node info
DATA_TYPE dataType;	Data type
SNMP_ID id;	Snmp Id
WORD_VAL dataLen;	Data length

DWORD hData;	Data
DWORD hSibling;	Sibling info
DWORD hChild;	Child info
BYTE index;	Index of object
BYTE indexLen;	Index length

**Section**

SNMP MIB variable object information

## 10.15.2.38 PDU\_INFO Structure

**File**

SNMP.h

**C**

```
typedef struct {
    DWORD_VAL requestID;
    BYTE nonRepeaters;
    BYTE maxRepetitions;
    BYTE pduType;
    BYTE errorStatus;
    BYTE erroIndex;
    BYTE snmpVersion;
    WORD pduLength;
} PDU_INFO;
```

**Members**

Members	Description
DWORD_VAL requestID;	Snmp request id
BYTE nonRepeaters;	non repeaters in the request
BYTE maxRepetitions;	max repeaters in the request
BYTE pduType;	Snmp pdu type
BYTE errorStatus;	Pdu error status
BYTE erroIndex;	Pdu error Index
BYTE snmpVersion;	Snmp version
WORD pduLength;	Pdu length

**Section**

SNMP pdu information database

## 10.15.2.39 reqVarErrStatus Structure

**File**

SNMP.h

**C**

```
typedef struct {
    WORD noSuchObjectErr;
    WORD noSuchNameErr;
    WORD noSuchInstanceErr;
    WORD endOfMibViewErr;
} reqVarErrStatus;
```

**Members**

Members	Description
WORD noSuchObjectErr;	Var list no such obj errors flags
WORD noSuchNameErr;	Var list no such name error
WORD noSuchInstanceErr;	Var list no such instance error
WORD endOfMibViewErr;	Var list end of mib view error

**Section**

SNMP requested variable list error status information.

Max variable in a request supported 15

## 10.15.2.40 SET\_REQUEST Macro

**File**

SNMP.h

**C**

```
#define SET_REQUEST (0xa3)
```

**Description**

This is macro SET\_REQUEST.

## 10.15.2.41 SetErrorStatus Function

**File**

SNMP.c

**C**

```
void SetErrorStatus(
    WORD errorStatusOffset,
    WORD errorIndexOffset,
    SNMP_ERR_STATUS errorStatus,
    BYTE errorIndex
);
```

**Description**

This is function SetErrorStatus.

## 10.15.2.42 SNMP\_AGENT\_PORT Macro

**File**

SNMP.h

**C**

```
#define SNMP_AGENT_PORT (161)
```

**Section**

SNMP Udp ports

### 10.15.2.43 SNMP\_BIB\_FILE\_NAME Macro

#### File

SNMP.h

#### C

```
#define SNMP_BIB_FILE_NAME "snmp.bib"
```

#### Description

This is the file that contains SNMP bib file. File name must contain all upper case letter and must match with what was included in MPFS2 image.

### 10.15.2.44 SNMP\_COUNTER32 Macro

#### File

SNMP.h

#### C

```
#define SNMP_COUNTER32 (0x41)
```

#### Description

This is macro SNMP\_COUNTER32.

### 10.15.2.45 SNMP\_ERR\_STATUS Enumeration

#### File

SNMP.h

#### C

```
typedef enum {
    SNMP_NO_ERR = 0,
    SNMP_TOO_BIG,
    SNMP_NO_SUCH_NAME,
    SNMP_BAD_VALUE,
    SNMP_READ_ONLY,
    SNMP_GEN_ERR,
    SNMP_NO_ACCESS,
    SNMP_WRONG_TYPE,
    SNMP_WRONG_LENGTH,
    SNMP_WRONG_ENCODING,
    SNMP_WRONG_VALUE,
    SNMP_NO_CREATION,
    SNMP_INCONSISTENT_VAL,
    SNMP_RESOURCE_UNAVAILABE,
    SNMP_COMMIT_FAILED,
    SNMP_UNDO_FAILED,
    SNMP_AUTH_ERROR,
    SNMP_NOT_WRITABLE,
    SNMP_INCONSISTENT_NAME,
    SNMP_NO_SUCH_OBJ = 128,
    SNMP_NO_SUCH_INSTANCE = 129,
    SNMP_END_OF_MIB_VIEW = 130
} SNMP_ERR_STATUS;
```

#### Members

Members	Description
SNMP_NO_ERR = 0	Snmp no error

SNMP_TOO_BIG	Value too big error
SNMP_NO_SUCH_NAME	No such name in MIB error
SNMP_BAD_VALUE	Not assignable value for the var error
SNMP_READ_ONLY	Read only variable, write not allowed err
SNMP_GEN_ERR	Snmp gen error
SNMP_NO_ACCESS	Access to modify or read not granted err
SNMP_WRONG_TYPE	Variable data type wrong error
SNMP_WRONG_LENGTH	Wrong data length error
SNMP_WRONG_ENCODING	Wrong encoding error
SNMP_WRONG_VALUE	Wrong value for the var type
SNMP_NO_CREATION	No creating error
SNMP_INCONSISTENT_VAL	Inconsistent value error
SNMP_RESOURCE_UNAVAILABE	Resource unavailbe error
SNMP_COMMIT_FAILED	Modification update failed error
SNMP_UNDO_FAILED	Modification undo failed
SNMP_AUTH_ERROR	Authorization failed error
SNMP_NOT_WRITABLE	Variable read only
SNMP_INCONSISTENT_NAME	Inconsistent name
SNMP_NO_SUCH_OBJ = 128	No such object error
SNMP_NO_SUCH_INSTANCE = 129	No such instance error
SNMP_END_OF_MIB_VIEW = 130	Reached to end of mib error

**Section**

SNMP specific errors

**10.15.2.46 SNMP\_GAUGE32 Macro****File**

SNMP.h

**C**

```
#define SNMP_GAUGE32 (0x42)
```

**Description**

This is macro SNMP\_GAUGE32.

**10.15.2.47 SNMP\_IP\_ADDR Macro****File**

SNMP.h

**C**

```
#define SNMP_IP_ADDR (0x40)
```

**Description**

This is macro SNMP\_IP\_ADDR.

## 10.15.2.48 SNMP\_NMS\_PORT Macro

### File

SNMP.h

### C

```
#define SNMP_NMS_PORT (162)
```

### Description

This is macro SNMP\_NMS\_PORT.

## 10.15.2.49 SNMP\_NOTIFY\_INFO Structure

### File

SNMP.h

### C

```
typedef struct {
    char community[NOTIFY_COMMUNITY_LEN];
    BYTE communityLen;
    SNMP_ID agentIDVar;
    BYTE notificationCode;
    UDP_SOCKET socket;
    DWORD_VAL timestamp;
    SNMP_ID trapIDVar;
} SNMP_NOTIFY_INFO;
```

### Members

Members	Description
char community[NOTIFY_COMMUNITY_LEN];	Community name array
BYTE communityLen;	Community name length
SNMP_ID agentIDVar;	Agent id for trap identification
BYTE notificationCode;	Trap notification code
UDP_SOCKET socket;	Udp socket number
DWORD_VAL timestamp;	Time stamp for trap
SNMP_ID trapIDVar;	SNMPV2 specific trap

### Section

SNMP trap notification information for agent

## 10.15.2.50 SNMP\_NSAP\_ADDR Macro

### File

SNMP.h

### C

```
#define SNMP_NSAP_ADDR (0x45)
```

### Description

This is macro SNMP\_NSAP\_ADDR.

## 10.15.2.51 IsValidLength Function

### File

SNMP.h

### C

```
BYTE IsValidLength(
    WORD * len
);
```

### Description

This is function IsValidLength.

## 10.15.2.52 SNMP\_OPAQUE Macro

### File

SNMP.h

### C

```
#define SNMP_OPAQUE (0x44)
```

### Description

This is macro SNMP\_OPAQUE.

## 10.15.2.53 SNMP\_STATUS Union

### File

SNMP.h

### C

```
typedef union {
    struct {
        unsigned int bIsFileOpen : 1;
    } Flags;
    BYTE Val;
} SNMP_STATUS;
```

### Members

Members	Description
unsigned int bIsFileOpen : 1;	MIB file access int flag
BYTE Val;	MIB file access byte flag

### Section

SNMP specific mib file access information

## 10.15.2.54 SNMP\_TIME\_TICKS Macro

### File

SNMP.h

### C

```
#define SNMP_TIME_TICKS (0x43)
```

**Description**

This is macro SNMP\_TIME\_TICKS.

## 10.15.2.55 SNMP\_V1 Macro

**File**

SNMP.h

**C**

```
#define SNMP_V1 (0)
```

**Section**

SNMP agent version types

## 10.15.2.56 SNMP\_V2C Macro

**File**

SNMP.h

**C**

```
#define SNMP_V2C (1)
```

**Description**

This is macro SNMP\_V2C.

## 10.15.2.57 SNMPAgentSocket Variable

**File**

SNMP.c

**C**

```
UDP_SOCKET SNMPAgentSocket = INVALID_UDP_SOCKET;
```

**Description**

Sntp udp socket

## 10.15.2.58 SNMPNotifyInfo Variable

**File**

SNMP.c

**C**

```
SNMP_NOTIFY_INFO SNMPNotifyInfo;
```

**Description**

notify info for trap

## 10.15.2.59 snmpReqVarErrStatus Variable

**File**

SNMP.c

**C**

```
reqVarErrStatus snmpReqVarErrStatus;
```

**Description**

vars from req list processing err status

## 10.15.2.60 SNMPRxOffset Variable

**File**

SNMP.c

**C**

```
WORD SNMPRxOffset;
```

**Description**

Sntp udp buffer rx offset

## 10.15.2.61 SNMPStatus Variable

**File**

SNMP.c

**C**

```
SNMP_STATUS SNMPStatus;
```

**Description**

MIB file access status

## 10.15.2.62 SNMPTxOffset Variable

**File**

SNMP.c

**C**

```
WORD SNMPTxOffset;
```

**Description**

Sntp udp buffer tx offset

## 10.15.2.63 STRUCTURE Macro

**File**

SNMP.h

**C**

```
#define STRUCTURE (0x30u)
```

**Section**

SNMP specific variables

## 10.15.2.64 TRAP Macro

**File**

SNMP.h

**C**

```
#define TRAP (0xa4)
```

**Description**

This is macro TRAP.

## 10.15.2.65 trapInfo Variable

**File**

CustomSNMPApp.c

**C**

```
TRAP_INFO trapInfo = { TRAP_TABLE_SIZE };
```

**Description**

Initialize trap table with no entries.

## 10.15.2.66 GetDataTypeInfo Function

**File**

SNMP.c

**C**

```
BOOL GetDataTypeInfo(  
    DATA_TYPE dataType,  
    DATA_TYPE_INFO* info  
);
```

**Description**

This is function GetDataTypeInfo.

## 10.15.2.67 GetNextLeaf Function

**File**

SNMP.c

**C**

```
BOOL GetNextLeaf(  
    OID_INFO* rec  
);
```

**Description**

This is function GetNextLeaf.

## 10.15.2.68 GetOIDStringByAddr Function

**File**

SNMP.c

**C**

```
BOOL GetOIDStringByAddr(  
    OID_INFO* rec,  
    BYTE* oidString,  
    BYTE* len  
);
```

**Description**

This is function GetOIDStringByAddr.

## 10.15.2.69 GetOIDStringByID Function

**File**

SNMP.c

**C**

```
BOOL GetOIDStringByID(  
    SNMP_ID id,  
    OID_INFO* info,  
    BYTE* oidString,  
    BYTE* len  
);
```

**Description**

This function is used only when TRAP is enabled.

## 10.15.2.70 IsValidCommunity Function

**File**

SNMP.c

**C**

```
static BOOL IsValidCommunity(  
    char* community,  
    BYTE* len  
);
```

**Description**

This is function IsValidCommunity.

## 10.15.2.71 IsValidInt Function

**File**

SNMP.h

**C**

```
BOOL IsValidInt(  
    DWORD* val  
);
```

**Description**

This is function IsValidInt.

## 10.15.2.72 IsValidLength Function

**File**

SNMP.c

**C**

```
BYTE IsValidLength(  
    WORD* len  
);
```

**Description**

This is function IsValidLength.

## 10.15.2.73 IsValidOID Function

**File**

SNMP.c

**C**

```
static BOOL IsValidOID(  
    BYTE* oid,  
    BYTE* len  
);
```

**Section**

Routines to validate snmp request pdu elements for SNMP format

## 10.15.2.74 IsValidPDU Function

**File**

SNMP.c

**C**

```
static BOOL IsValidPDU(  
    SNMP_ACTION* pdu  
);
```

**Description**

static BOOL IsValidInt (see page 349)(DWORD\* val);

## 10.15.2.75 IsValidStructure Function

**File**

SNMP.h

**C**

```

BYTE IsValidStructure(
    WORD* dataLen
);

```

**Description**

This is function IsValidStructure.

## 10.15.2.76 OIDLookup Function

**File**

SNMP.c

**C**

```

BYTE OIDLookup(
    PDU_INFO* pduDbPtr,
    BYTE* oid,
    BYTE oidLen,
    OID_INFO* rec
);

```

**Section**

Routines to read/search OIDs, objects from the SNMP MIB database

## 10.15.2.77 ProcessGetSetHeader Function

**File**

SNMP.c

**C**

```

static BOOL ProcessGetSetHeader(
    PDU_INFO* pduDbPtr
);

```

**Description**

All the variables of snmp pdu request header are validated for their data types. Collects request\_id for the snmp request pdu. Fetch, validates error status, error index and discard as they are need not to be processed as received in request pdu. Collects non repeaters and max repeaters values in case of Get\_Bulk request.

**Remarks**

The request pdu will be processed only if this routine returns TRUE

**Preconditions**

ProcessHeader (see page 352)() is called and returns pdu type and do not returns SNMP\_ACTION\_UNKNOWN

**Parameters**

Parameters	Description
pduDbPtr	Pointer to received pdu information database.

**Return Values**

Return Values	Description
TRUE	If the received request header is validated and passed.
FALSE	If rxed request header is not valid.

## 10.15.2.78 ProcessHeader Function

### File

SNMP.c

### C

```
static SNMP_ACTION ProcessHeader(
    PDU_INFO* pduDbPtr,
    char* community,
    BYTE* len
);
```

### Description

Collects PDU\_INFO (see page 340) (SNMP pdu information database), community name, community length and length of data payload. This function validates the received udp packet for these different variables of snmp pdu. The sequence in which these elements are received is important. The validation is done for the agent processing capabilities and the max UDP packet length as UDP packets can not be fragmented.

### Remarks

The received pdu will be processed only if this routine returns the pdu type else the pdu is discarded as not Snmp pdu.

### Preconditions

UDPIsGetReady (see page 523)(SNMPAgentSocket (see page 346)) is called in SNMPTask (see page 354)(), it check if there is any packet on SNMP Agent socket, should return TRUE.

### Parameters

Parameters	Description
pduDbPtr	Pointer to received pdu information database
community	Pointer to var storing, community string in rxed pdu
len	Pointer to var storing, community string length rxed in pdu

### Return Values

Return Values	Description
SNMP_ACTION (see page 316)	Snmp request pdu type.

## 10.15.2.79 ProcessSetVar Function

### File

SNMP.h

### C

```
BYTE ProcessSetVar(
    PDU_INFO* pduDbPtr,
    OID_INFO* rec,
    SNMP_ERR_STATUS* errorStatus
);
```

### Description

This is function ProcessSetVar.

## 10.15.2.80 ProcessVariables Function

### File

SNMP.c

**C**

```
static BOOL ProcessVariables(
    PDU_INFO* pduDbPtr,
    char* community,
    BYTE len
);
```

**Description**

Once the received pdu is validated as Snmp pdu, it is forwarded for processing to this routine. This routine handles Get, Get\_Next, Get\_Bulk, Set request and creates appropriate response as Get\_Response. This routine will decide on whether the request pdu should be processed or be discarded.

**Remarks**

None

**Preconditions**

The received udp packet is verified as SNMP request. ProcessHeader (see page 352)() and ProcessGetSetHeader (see page 351)() returns but FALSE.

**Parameters**

Parameters	Description
pduDbPtr	Pointer to received pdu information database
community	Pointer to var, storing community string in rxed pdu
len	Pointer to var, storing community string length rxed in pdu

**Return Values**

Return Values	Description
TRUE	If the snmp request processing is successful.
FALSE	If the processing failed else the processing is not completed.

## 10.15.2.81 ReadMIBRecord Function

**File**

SNMP.c

**C**

```
static void ReadMIBRecord(
    DWORD h,
    OID_INFO* rec
);
```

**Description**

This is function ReadMIBRecord.

## 10.15.2.82 SNMPCheckIfPvtMibObjRequested Function

**File**

SNMP.c

**C**

```
static BOOL SNMPCheckIfPvtMibObjRequested(
    BYTE* OIDValuePtr
);
```

**Section**

Routine to check if private mib object is requested by NMS.

---

## 10.15.3 SNMP Stack Members

**Functions**

	Name	Description
☞	SNMPInit (☞ see page 354)	Initialize SNMP module internals.
☞	SNMPTask (☞ see page 354)	Polls for every snmp pdu received.

**Module**

SNMP (☞ see page 312)

**Description**

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.15.3.1 SNMPInit Function

**File**

SNMP.h

**C**

```
void SNMPInit();
```

**Returns**

None

**Description**

This function initializes the Snmp agent. One udp socket is initialized and opened at port 161. Agent will receive and transmit all the snmp pdus on this udp socket.

**Remarks**

This function is called only once during lifetime of the application. One UDP socket will be used.

**Preconditions**

At least one UDP socket must be available. UDPInit (☞ see page 530)() is already called.

**Section**

Function Prototypes

### 10.15.3.2 SNMPTask Function

**File**

SNMP.h

**C**

```
BOOL SNMPTask();
```

**Description**

Handle incoming SNMP requests as well as any outgoing SNMP responses and timeout conditions.

**Remarks**

None

**Preconditions**

SNMPInit (see page 354)() is already called.

**Return Values**

Return Values	Description
TRUE	If SNMP module has finished with a state
FALSE	If a state has not been finished.

## 10.15.4 Functions

**Functions**

	Name	Description
◆	getSnmV2GenTrapOid (see page 356)	Resolves generic trap code to generic trap OID.
◆	ProcessGetBulkVar (see page 356)	This is function ProcessGetBulkVar.
◆	ProcessGetNextVar (see page 357)	This is function ProcessGetNextVar.
◆	ProcessGetVar (see page 357)	This is function ProcessGetVar.
◆	ProcessSnmV3MsgData (see page 357)	This is function ProcessSnmV3MsgData.
◆	SNMPGetExactIndex (see page 357)	To search for exact index node in case of a Sequence variable.
◆	SNMPIdRecrdValidation (see page 358)	Used to validate the support of Var ID for A particular SNMP Version.
◆	SNMPIsValidSetLen (see page 359)	Validates the set variable data length to data type.
◆	SnmV3AESDecryptRxedScopedPdu (see page 359)	This is function SnmV3AESDecryptRxedScopedPdu.
◆	SnmV3BufferPut (see page 359)	This is function SnmV3BufferPut.
◆	SnmV3FormulateEngineID (see page 360)	This is function SnmV3FormulateEngineID.
◆	SnmV3GetAuthEngineTime (see page 360)	This is function SnmV3GetAuthEngineTime.
◆	SnmV3GetBufferData (see page 360)	This is function SnmV3GetBufferData.
◆	SnmV3InitializeUserDataBase (see page 360)	This is function SnmV3InitializeUserDataBase.
◆	SnmV3MsgProcessingModelProcessPDU (see page 361)	This is function SnmV3MsgProcessingModelProcessPDU.
◆	SnmV3Notify (see page 361)	This is function SnmV3Notify.
◆	SnmV3ScopedPduProcessing (see page 361)	This is function SnmV3ScopedPduProcessing.
◆	SnmV3TrapScopedpdu (see page 361)	This is function SnmV3TrapScopedpdu.
◆	SnmV3UserSecurityModelProcessPDU (see page 362)	This is function SnmV3UserSecurityModelProcessPDU.
◆	SnmV3UsmAesEncryptDecryptInitVector (see page 362)	This is function SnmV3UsmAesEncryptDecryptInitVector.

	Snmpv3UsmOutMsgAuthenticationParam (see page 362)	This is function Snmpv3UsmOutMsgAuthenticationParam.
	Snmpv3ValidateEngineId (see page 362)	This is function Snmpv3ValidateEngineId.
	Snmpv3ValidateSecNameAndSecLvl (see page 363)	This is function Snmpv3ValidateSecNameAndSecLvl.
	Snmpv3ValidateSecurityName (see page 363)	This is function Snmpv3ValidateSecurityName.

**Module**

SNMP (see page 312)

## 10.15.4.1 getSnmpV2GenTrapOid Function

**File**

SNMP.h

**C**

```

BYTE * getSnmpV2GenTrapOid(
    BYTE generic_trap_code,
    BYTE * len
);

```

**Description**

This function resolves given generic trap code to generic trap OID.

**Remarks**

This would fail if generic\_trap\_code is not coming under GENERIC\_TRAP\_NOTIFICATION\_TYPE (see page 316)

**Preconditions**

SNMPNotifyPrepare (see page 323)() is already called.

**Parameters**

Parameters	Description
generic_trap_code	GENERIC_TRAP_NOTIFICATION_TYPE (see page 316)
len	generic trap OID length

**Return Values**

Return Values	Description
BYTE *	TRAP OID

## 10.15.4.2 ProcessGetBulkVar Function

**File**

SNMP.h

**C**

```

BYTE ProcessGetBulkVar(
    OID_INFO* rec,
    BYTE* oidValuePtr,
    BYTE* oidLenPtr,
    BYTE* successor,
    PDU_INFO* pduDbPtr
);

```

**Description**

This is function ProcessGetBulkVar.

## 10.15.4.3 ProcessGetNextVar Function

**File**

SNMP.h

**C**

```
BYTE ProcessGetNextVar(  
    OID_INFO* rec,  
    PDU_INFO* pduDbPtr  
);
```

**Description**

This is function ProcessGetNextVar.

## 10.15.4.4 ProcessGetVar Function

**File**

SNMP.h

**C**

```
BYTE ProcessGetVar(  
    OID_INFO* rec,  
    BOOL bAsOID,  
    PDU_INFO* pduDbPtr  
);
```

**Description**

This is function ProcessGetVar.

## 10.15.4.5 ProcessSnmpv3MsgData Function

**File**

SNMP.h

**C**

```
BOOL ProcessSnmpv3MsgData(  
    PDU_INFO* pduDbPtr  
);
```

**Description**

This is function ProcessSnmpv3MsgData.

## 10.15.4.6 SNMPGetExactIndex Function

**File**

SNMP.h

**C**

```
BOOL SNMPGetExactIndex(  
    SNMP_ID var,
```

```
SNMP_INDEX index
);
```

**Description**

This is a callback function called by SNMP module. SNMP user must implement this function in user application and provide appropriate data when called. This function will only be called for OID variable of type sequence.

**Remarks**

Only sequence index needs to be handled in this function.

**Preconditions**

None

**Parameters**

Parameters	Description
var	Variable id as per mib.h (input)
index	Index of variable (input)

**Return Values**

Return Values	Description
TRUE	If the exact index value exists for given variable at given index.
FALSE	Otherwise.

## 10.15.4.7 SNMPIdRecrdValidation Function

**File**

SNMP.h

**C**

```
BOOL SNMPIdRecrdValidation(
    PDU_INFO * pduPtr,
    OID_INFO * var,
    BYTE * oidValuePtr,
    BYTE oidLen
);
```

**Description**

This is a callback function called by SNMP module. SNMP user must implement this function as per SNMP version. One need to add the new SNMP MIB IDs hereas per SNMP version. e.g - SYS\_UP\_TIME (250) is common for V1/V2/V3 ENGINE\_ID - is the part of V3, So put the all the SNMPv3 var ids within Macro STACK\_USE\_SNMPV3\_SERVER.

**Remarks**

None.

**Preconditions**

None

**Parameters**

Parameters	Description
var	Variable rec whose record id need to be validated
oidValuePtr	OID Value
oidLen	oidValuePtr length

**Return Values**

Return Values	Description
TRUE	If a Var ID exists .

FALSE	Otherwise.
-------	------------

### 10.15.4.8 SNMPIsValidSetLen Function

#### File

SNMP.h

#### C

```

BOOL SNMPIsValidSetLen(
    SNMP_ID var,
    BYTE len,
    BYTE index
);

```

#### Description

This routine is used to validate the dynamic variable data length to the variable data type. It is used when SET request is processed. This is a callback function called by module. User application must implement this function.

#### Remarks

This function will be called for only dynamic variables that are defined as ASCII\_STRING and OCTET\_STRING (see page 339) (i.e. data length greater than 4 bytes)

#### Preconditions

ProcessSetVar (see page 352)() is called.

#### Parameters

Parameters	Description
var	Variable id whose value is to be set
len	Length value that is to be validated.
index	instance of a OID

#### Return Values

Return Values	Description
TRUE	if given var can be set to given len
FALSE	if otherwise.

### 10.15.4.9 Snmpv3AESDecryptRxedScopedPdu Function

#### File

SNMP.h

#### C

```

BYTE Snmpv3AESDecryptRxedScopedPdu();

```

#### Description

This is function Snmpv3AESDecryptRxedScopedPdu.

### 10.15.4.10 Snmpv3BufferPut Function

#### File

SNMP.h

**C**

```
BOOL Snmpv3BufferPut(  
    BYTE val,  
    SNMPV3MSGDATA * putbuf  
);
```

**Description**

This is function Snmpv3BufferPut.

## 10.15.4.11 Snmpv3FormulateEngineID Function

**File**

SNMP.h

**C**

```
void Snmpv3FormulateEngineID(  
    UINT8 fifthOctectIdentifier  
);
```

**Description**

This is function Snmpv3FormulateEngineID.

## 10.15.4.12 Snmpv3GetAuthEngineTime Function

**File**

SNMP.h

**C**

```
void Snmpv3GetAuthEngineTime();
```

**Description**

This is function Snmpv3GetAuthEngineTime.

## 10.15.4.13 Snmpv3GetBufferData Function

**File**

SNMP.h

**C**

```
BYTE Snmpv3GetBufferData(  
    SNMPV3MSGDATA getbuf,  
    UINT16 pos  
);
```

**Description**

This is function Snmpv3GetBufferData.

## 10.15.4.14 Snmpv3InitializeUserDataBase Function

**File**

SNMP.c

**C**

```
void Snmpv3InitializeUserDataBase();
```

**Description**

This is function Snmpv3InitializeUserDataBase.

## 10.15.4.15 Snmpv3MsgProcessingModelProcessPDU Function

**File**

SNMP.h

**C**

```
SNMP_ACTION Snmpv3MsgProcessingModelProcessPDU(  
    BYTE inOutPdu  
);
```

**Description**

This is function Snmpv3MsgProcessingModelProcessPDU.

## 10.15.4.16 Snmpv3Notify Function

**File**

SNMP.h

**C**

```
BOOL Snmpv3Notify(  
    SNMP_ID var,  
    SNMP_VAL val,  
    SNMP_INDEX index,  
    UINT8 targetIndex  
);
```

**Description**

This is function Snmpv3Notify.

## 10.15.4.17 Snmpv3ScopedPduProcessing Function

**File**

SNMP.h

**C**

```
SNMP_ACTION Snmpv3ScopedPduProcessing(  
    BYTE inOutPdu  
);
```

**Description**

This is function Snmpv3ScopedPduProcessing.

## 10.15.4.18 Snmpv3TrapScopedpdu Function

**File**

SNMP.h

**C**

```
UINT8 Snmpv3TrapScopedpdu(  
    SNMP_ID var,  
    SNMP_VAL val,  
    SNMP_INDEX index,  
    UINT8 targetIndex  
);
```

**Description**

This is function Snmpv3TrapScopedpdu.

## 10.15.4.19 Snmpv3UserSecurityModelProcessPDU Function

**File**

SNMP.h

**C**

```
SNMP_ACTION Snmpv3UserSecurityModelProcessPDU(  
    BYTE inOutPdu  
);
```

**Description**

This is function Snmpv3UserSecurityModelProcessPDU.

## 10.15.4.20 Snmpv3UsmAesEncryptDecryptInitVector Function

**File**

SNMP.h

**C**

```
void Snmpv3UsmAesEncryptDecryptInitVector(  
    BYTE inOutPdu  
);
```

**Description**

This is function Snmpv3UsmAesEncryptDecryptInitVector.

## 10.15.4.21 Snmpv3UsmOutMsgAuthenticationParam Function

**File**

SNMP.h

**C**

```
void Snmpv3UsmOutMsgAuthenticationParam(  
    UINT8 hashType  
);
```

**Description**

This is function Snmpv3UsmOutMsgAuthenticationParam.

## 10.15.4.22 Snmpv3ValidateEngineId Function

**File**

SNMP.h

**C**

```
BOOL Snmpv3ValidateEngineId();
```

**Description**

This is function Snmpv3ValidateEngineId.

**10.15.4.23 Snmpv3ValidateSecNameAndSecLvl Function****File**

SNMP.h

**C**

```
BOOL Snmpv3ValidateSecNameAndSecLvl();
```

**Description**

This is function Snmpv3ValidateSecNameAndSecLvl.

**10.15.4.24 Snmpv3ValidateSecurityName Function****File**

SNMP.h

**C**

```
BOOL Snmpv3ValidateSecurityName();
```

**Description**

This is function Snmpv3ValidateSecurityName.

**10.15.5 Types****Enumerations**

	Name	Description
	INOUT_SNMP_PDU ( <a href="#">see page 363</a> )	This is type INOUT_SNMP_PDU.

**Module**

SNMP ([see page 312](#))

**Structures**

	Name	Description
	SNMPNONMIBRECDINFO ( <a href="#">see page 364</a> )	This is type SNMPNONMIBRECDINFO.
	SNMPV3MSGDATA ( <a href="#">see page 364</a> )	SNMPv3

**10.15.5.1 INOUT\_SNMP\_PDU Enumeration****File**

SNMP.h

**C**

```
typedef enum {
    SNMP_RESPONSE_PDU = 0x01,
    SNMP_REQUEST_PDU = 0x02
} INOUT_SNMP_PDU;
```

**Description**

This is type INOUT\_SNMP\_PDU.

## 10.15.5.2 SNMPNONMIBRECDINFO Structure

**File**

SNMP.h

**C**

```
typedef struct {
    UINT8 oidstr[16];
    UINT8 version;
} SNMPNONMIBRECDINFO;
```

**Description**

This is type SNMPNONMIBRECDINFO.

## 10.15.5.3 SNMPV3MSGDATA Structure

**File**

SNMP.h

**C**

```
typedef struct {
    UINT8 * head;
    WORD length;
    WORD maxlength;
    WORD msgAuthParamOffset;
} SNMPV3MSGDATA;
```

**Description**

SNMPv3

## 10.15.6 Variables

**Module**

SNMP ([see page 312](#))

**Variables**

	Name	Description
	getZeroInstance ( <a href="#">see page 365</a> )	This variable is used for get next request for zero instance
	gSnmNonMibRecInfo ( <a href="#">see page 365</a> )	Below oidStr is the collection of OID variables which are not part of MIB.h file
	gSNMPv3ScopedPduDataPos ( <a href="#">see page 365</a> )	This is variable gSNMPv3ScopedPduDataPos.

	gSNMPv3ScopedPduRequestBuf ( <a href="#">see page 365</a> )	This is variable gSNMPv3ScopedPduRequestBuf.
	gSNMPv3ScopedPduResponseBuf ( <a href="#">see page 366</a> )	This is variable gSNMPv3ScopedPduResponseBuf.
	msgSecrtyParamLenOffset ( <a href="#">see page 366</a> )	This is variable msgSecrtyParamLenOffset.

### 10.15.6.1 getZeroInstance Variable

#### File

SNMP.c

#### C

```
BOOL getZeroInstance;
```

#### Description

This variable is used for next request for zero instance

### 10.15.6.2 gSnmplibNonMibRecInfo Variable

#### File

SNMP.c

#### C

```
SNMPNONMIBRECDINFO gSnmplibNonMibRecInfo[SNMP_MAX_NON_REC_ID_OID] = {
  {{43,6,1,4,1,0x81,0x85,0x47,0x1,6},SNMP_V3}, {{43,6,1,2,1,1},SNMP_V2C},
  {{43,6,1,4,1,0x81,0x85,0x47,0x1,1},SNMP_V2C}, };
```

#### Description

Below oidStr is the collection of OID variables which are not part of MIB.h file

### 10.15.6.3 gSNMPv3ScopedPduDataPos Variable

#### File

SNMP.h

#### C

```
UINT16 gSNMPv3ScopedPduDataPos;
```

#### Description

This is variable gSNMPv3ScopedPduDataPos.

### 10.15.6.4 gSNMPv3ScopedPduRequestBuf Variable

#### File

SNMP.h

#### C

```
SNMPV3MSGDATA gSNMPv3ScopedPduRequestBuf;
```

#### Description

This is variable gSNMPv3ScopedPduRequestBuf.

## 10.15.6.5 gSNMPv3ScopedPduResponseBuf Variable

### File

SNMP.h

### C

```
SNMPV3MSGDATA gSNMPv3ScopedPduResponseBuf;
```

### Description

This is variable gSNMPv3ScopedPduResponseBuf.

## 10.15.6.6 msgSecrtyParamLenOffset Variable

### File

SNMP.c

### C

```
WORD msgSecrtyParamLenOffset;
```

### Description

This is variable msgSecrtyParamLenOffset.

## 10.15.7 Macros

### Macros

	Name	Description
	IS_SNMPV3_AUTH_STRUCTURE ( <a href="#">see page 366</a> )	This is macro IS_SNMPV3_AUTH_STRUCTURE.
	REPORT_RESPONSE ( <a href="#">see page 367</a> )	This is macro REPORT_RESPONSE.
	SNMP_MAX_MSG_SIZE ( <a href="#">see page 367</a> )	SNMP MIN and MAX message 484 bytes in size As per RFC 3411 snmpEngineMaxMessageSize and RFC 1157 ( section 4- protocol specification ) and implementation supports more than 484 whenever feasible.
	SNMP_MAX_NON_REC_ID_OID ( <a href="#">see page 367</a> )	Update the Non record id OID value which is part of CustomSnmDemo.c file
	SNMP_V3 ( <a href="#">see page 367</a> )	This is macro SNMP_V3.

### Module

SNMP ([see page 312](#))

## 10.15.7.1 IS\_SNMPV3\_AUTH\_STRUCTURE Macro

### File

SNMP.h

### C

```
#define IS_SNMPV3_AUTH_STRUCTURE(a) (a==SNMPV3_ENCRYPTION)
```

**Description**

This is macro IS\_SNMPV3\_AUTH\_STRUCTURE.

## 10.15.7.2 REPORT\_RESPONSE Macro

**File**

SNMP.h

**C**

```
#define REPORT_RESPONSE (0xa8)
```

**Description**

This is macro REPORT\_RESPONSE.

## 10.15.7.3 SNMP\_MAX\_MSG\_SIZE Macro

**File**

SNMP.h

**C**

```
#define SNMP_MAX_MSG_SIZE 484
```

**Description**

SNMP MIN and MAX message 484 bytes in size As per RFC 3411 snmpEngineMaxMessageSize and RFC 1157 ( section 4- protocol specification ) and implementation supports more than 484 whenever feasible.

## 10.15.7.4 SNMP\_MAX\_NON\_REC\_ID\_OID Macro

**File**

SNMP.h

**C**

```
#define SNMP_MAX_NON_REC_ID_OID 3
```

**Description**

Update the Non record id OID value which is part of CustomSnmpDemo.c file

## 10.15.7.5 SNMP\_V3 Macro

**File**

SNMP.h

**C**

```
#define SNMP_V3 (3)
```

**Description**

This is macro SNMP\_V3.

## 10.16 SNTP Client

The SNTP module implements the Simple Network Time Protocol. The module (by default) updates its internal time every 10 minutes using a pool of public global time servers. It then calculates reference times on any call to `SNTPGetUTCSeconds` (see page 368) using the internal Tick timer module.

The SNTP module is good for providing absolute time stamps. However, it should not be relied upon for measuring time differences (especially small differences). The pool of public time servers is implemented using round-robin DNS, so each update will come from a different server. Differing network delays and the fact that these servers are not verified implies that this time could be non-linear. While it is deemed reliable, it is not guaranteed to be accurate.

The Tick module provides much better accuracy (since it is driven by a hardware clock) and resolution, and should be used for measuring timeouts and other internal requirements.

Developers can change the value of `NTP_SERVER` (see page 372) if they wish to always point to a preferred time server, or to specify a region when accessing time servers. The default is to use the global pool.

### 10.16.1 SNTP Client Public Members

#### Functions

	Name	Description
	<code>SNTPGetUTCSeconds</code> (see page 368)	Obtains the current time from the SNTP module.

#### Module

SNTP Client (see page 368)

#### Description

The following functions and variables are available to the stack application.

#### 10.16.1.1 SNTPGetUTCSeconds Function

##### File

SNTP.h

##### C

```
DWORD SNTPGetUTCSeconds ( ) ;
```

##### Returns

The number of seconds since the Epoch. (Default 01-Jan-1970 00:00:00)

##### Description

This function obtains the current time as reported by the SNTP module. Use this value for absolute time stamping. The value returned is (by default) the number of seconds since 01-Jan-1970 00:00:00.

##### Remarks

Do not use this function for time difference measurements. The Tick module is more appropriate for those requirements.

##### Preconditions

None

## 10.16.2 SNTP Client Stack Members

### Functions

	Name	Description
	SNTPClient ( <a href="#">see page 369</a> )	Periodically checks the current time from a pool of servers.

### Module

SNTP Client ([see page 368](#))

### Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.16.2.1 SNTPClient Function

#### File

SNTP.h

#### C

```
void SNTPClient();
```

#### Returns

None

#### Description

This function periodically checks a pool of time servers to obtain the current date/time.

#### Remarks

This function requires once available UDP socket while processing, but frees that socket when the SNTP module is idle.

#### Preconditions

UDP is initialized.

## 10.16.3 SNTP Client Internal Members

### Macros

	Name	Description
	NTP_EPOCH ( <a href="#">see page 372</a> )	Reference Epoch to use. (default: 01-Jan-1970 00:00:00)
	NTP_FAST_QUERY_INTERVAL ( <a href="#">see page 372</a> )	Defines how long to wait to retry an update after a failure. Updates may take up to 6 seconds to fail, so this 14 second delay is actually only an 8-second retry.
	NTP_QUERY_INTERVAL ( <a href="#">see page 372</a> )	Defines how frequently to resynchronize the date/time (default: 10 minutes)
	NTP_REPLY_TIMEOUT ( <a href="#">see page 372</a> )	Defines how long to wait before assuming the query has failed

	NTP_SERVER ( <a href="#">see page 372</a> )	These are normally available network time servers. The actual IP returned from the pool will vary every minute so as to spread the load around stratum 1 timeservers. For best accuracy and network overhead you should locate the pool server closest to your geography, but it will still work if you use the global pool.ntp.org address or choose the wrong one or ship your embedded device to another geography.
	NTP_SERVER_PORT ( <a href="#">see page 373</a> )	Port for contacting NTP servers

**Module**SNTP Client ([see page 368](#))**Structures**

	Name	Description
	NTP_PACKET ( <a href="#">see page 370</a> )	Defines the structure of an NTP packet

**Variables**

	Name	Description
	dwLastUpdateTick ( <a href="#">see page 371</a> )	Tick count of last update
	dwSNTPSeconds ( <a href="#">see page 371</a> )	Seconds value obtained by last update

**Description**

The following functions and variables are designated as internal to the SNTP Client module.

## 10.16.3.1 NTP\_PACKET Structure

**File**

SNTP.c

**C**

```
typedef struct {
    struct {
        BYTE mode : 3;
        BYTE versionNumber : 3;
        BYTE leapIndicator : 2;
    } flags;
    BYTE stratum;
    CHAR poll;
    CHAR precision;
    DWORD root_delay;
    DWORD root_dispersion;
    DWORD ref_identifier;
    DWORD ref_ts_secs;
    DWORD ref_ts_fraq;
    DWORD orig_ts_secs;
    DWORD orig_ts_fraq;
    DWORD recv_ts_secs;
    DWORD recv_ts_fraq;
    DWORD tx_ts_secs;
    DWORD tx_ts_fraq;
} NTP_PACKET;
```

**Members**

Members	Description
struct { BYTE mode : 3; BYTE versionNumber : 3; BYTE leapIndicator : 2; } flags;	Flags for the packet
BYTE mode : 3;	NTP mode
BYTE versionNumber : 3;	SNTP version number
BYTE leapIndicator : 2;	Leap second indicator
BYTE stratum;	Stratum level of local clock
CHAR poll;	Poll interval
CHAR precision;	Precision (seconds to nearest power of 2)
DWORD root_delay;	Root delay between local machine and server
DWORD root_dispersion;	Root dispersion (maximum error)
DWORD ref_identifier;	Reference clock identifier
DWORD ref_ts_secs;	Reference timestamp (in seconds)
DWORD ref_ts_fraq;	Reference timestamp (fractions)
DWORD orig_ts_secs;	Origination timestamp (in seconds)
DWORD orig_ts_fraq;	Origination timestamp (fractions)
DWORD recv_ts_secs;	Time at which request arrived at sender (seconds)
DWORD recv_ts_fraq;	Time at which request arrived at sender (fractions)
DWORD tx_ts_secs;	Time at which request left sender (seconds)
DWORD tx_ts_fraq;	Time at which request left sender (fractions)

**Description**

Defines the structure of an NTP packet

### 10.16.3.2 dwLastUpdateTick Variable

**File**

SNTP.c

**C**

```
DWORD dwLastUpdateTick = 0;
```

**Description**

Tick count of last update

### 10.16.3.3 dwSNTPSeconds Variable

**File**

SNTP.c

**C**

```
DWORD dwSNTPSeconds = 0;
```

**Description**

Seconds value obtained by last update

### 10.16.3.4 NTP\_EPOCH Macro

**File**

SNTP.c

**C**

```
#define NTP_EPOCH (86400ul * (365ul * 70ul + 17ul))
```

**Description**

Reference Epoch to use. (default: 01-Jan-1970 00:00:00)

### 10.16.3.5 NTP\_FAST\_QUERY\_INTERVAL Macro

**File**

SNTP.c

**C**

```
#define NTP_FAST_QUERY_INTERVAL (14ul * TICK_SECOND)
```

**Description**

Defines how long to wait to retry an update after a failure. Updates may take up to 6 seconds to fail, so this 14 second delay is actually only an 8-second retry.

### 10.16.3.6 NTP\_QUERY\_INTERVAL Macro

**File**

SNTP.c

**C**

```
#define NTP_QUERY_INTERVAL (10ul*60ul * TICK_SECOND)
```

**Description**

Defines how frequently to resynchronize the date/time (default: 10 minutes)

### 10.16.3.7 NTP\_REPLY\_TIMEOUT Macro

**File**

SNTP.c

**C**

```
#define NTP_REPLY_TIMEOUT (6ul*TICK_SECOND)
```

**Description**

Defines how long to wait before assuming the query has failed

### 10.16.3.8 NTP\_SERVER Macro

**File**

SNTP.c

**C**

```
#define NTP_SERVER "pool.ntp.org"
```

**Description**

These are normally available network time servers. The actual IP returned from the pool will vary every minute so as to spread the load around stratum 1 timeservers. For best accuracy and network overhead you should locate the pool server closest to your geography, but it will still work if you use the global pool.ntp.org address or choose the wrong one or ship your embedded device to another geography.

## 10.16.3.9 NTP\_SERVER\_PORT Macro

**File**

SNTP.c

**C**

```
#define NTP_SERVER_PORT (123u1)
```

**Description**

Port for contacting NTP servers

---

# 10.17 SSL

The SSL module adds encryption support to the TCP layer by implementing the SSLv3 protocol. This protocol is the standard for secure communications across the Internet, and prevents snooping or tampering of data as it travels across an untrusted network.

**To comply with US Export Control restrictions, the encryption portion of the SSL module must be purchased separately from Microchip.** The library of Data Encryption Routines (SW300052) is available for a nominal fee from <http://www.microchipdirect.com/productsearch.aspx?Keywords=SW300052>.

**SSL Client Support**

An SSL client can be initiated by first opening a TCP connection, then calling TCPStartSSLSession to initiate the SSL handshake process. The handshake uses the public key from the certificate provided by the server. Key lengths up to 1024 bits are supported.

Once the handshake has started, call TCPSSLIsHandshaking (see page 378) until it returns FALSE. This will indicate that the handshake has completed and all traffic is now secured using 128-bit ARCFOUR encryption. If the handshake fails for any reason, the TCP connection will automatically be terminated as required by the SSL protocol specification.

For faster performance, the SSL module caches security parameters for the most recently made connections. This allows quick reconnections to the same node without the computational expense of another RSA handshake. By default, the two most recent connections are cached, but this can be modified in TCPIPConfig.h.

SSL client support is already enabled for SMTP. When STACK\_USE\_SSL\_CLIENT is defined, the SMTP module automatically adds a field to SMTPClient (see page 298) called UseSSL. That field controls whether or not the SMTP client module will attempt to make an SSL connection before transmitting any data.

**SSL Server Support**

To initiate an SSL server, first open a TCP socket for listening using TCPOpen (see page 451). Then call TCPAddSSLListener (see page 378) to listen (see page 169) for incoming SSL connections on an alternate port. This allows a single socket to share application-level resources and listen (see page 169) for connections on two different ports. Connections occurring on the originally opened port will proceed unsecured, while connections on the SSL port will

first complete an SSL handshake to secure the data.

If your application will not accept (see page 163) unsecured traffic, simply open a non-secured socket on a free port number, then verify that each incoming connection is secured (not on that port) by calling TCPIsSSL (see page 379).

SSL server support is automatically enabled for HTTP2 when `STACK_USE_SSL_SERVER` is defined. By default, the HTTP2 module will then listen (see page 169) for unsecured traffic on port 80 and secured connections on port 443.

### Limitations

SSL was designed for desktop PCs with faster processors and significantly more resources than are available on an embedded platform. A few compromises must be made in order to use SSL in a less resource-intensive manner.

The SSL client module does not perform any validation or verification of certificates. Doing so would require many root certificates to be stored locally for verification, which is not feasible for memory-limited parts. This does not compromise security once the connection has been established, but does not provide complete security against man-in-the-middle attacks. (This sort of attack is uncommon and would be difficult to execute.)

Neither the SSL client nor the server can completely verify MACs before processing data. SSL records include a signature to verify that messages were not modified in transit. This Message Authentication Code, or MAC, is inserted after at least every 16kB of traffic. (It usually is inserted much more frequently than that.) Without 16kB of RAM to buffer packets for each socket, incoming data must be handed to the application layer before the MAC can be completely verified. Invalid MACs will still cause the connection to terminate immediately, but by the time this is detected some bad data may have already reached the application. Since the ARCFOUR cipher in use is a stream cipher, it would be difficult to exploit this in any meaningful way. An attacker would not be able to control what data is actually modified or inserted, as doing so without knowledge of the key would yield garbage. However, it is important to understand that incoming data is not completely verified before being passed to the application.

---

## 10.17.1 Generating Server Certificates

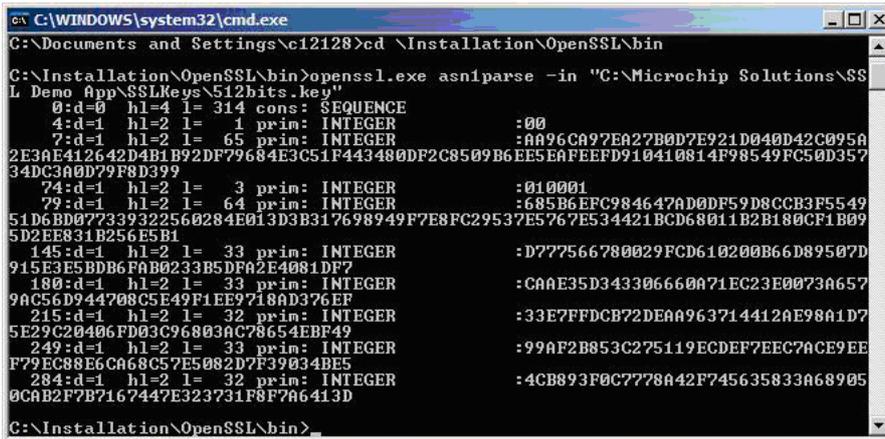
### Module

SSL (see page 373)

### Description

The SSL certificates used by the TCP/IP Stack's SSL module are stored in the CustomSSLCert.c source file. The following series of steps describe how to create the structures in CustomSSLCert.c using an SSL certificate.

1. Download and install the OpenSSL library. There are several third-party sites that offer SSL installers (e.g. <http://www.slproweb.com/products/Win32OpenSSL.html>). Note that some distributions may not include all commands specified by the OpenSSL documentation.
2. Open a console and change directory to the **OpenSSL/bin** folder.
3. If you don't have a key and certificate, you can generate them first. The following example console commands will generate a 512-bit key:
  1. Generate the key: **openssl genrsa -out 512bits.key 512**
  2. Generate the Certificate Signing Request (CSR). You will need to add additional information when prompted: **openssl req -new -key 512bits.key -out 512bits.csr**
  3. Generate the X.509 certificate if self-signing (or send the CSR to a Certificate Authority for signing): **openssl x509 -req -days 365 -in 512bits.csr -signkey 512bits.key -out 512bits.crt** (note that if the -days option is not specified, the default expiration time is 30 days)
  4. For additional documentation, refer to <http://www.openssl.org/docs/apps/openssl.html>.
4. Parse your key file using the command: **openssl.exe asn1parse -in "[directory containing your key]\512bits.key"**
5. You should see a screen like this:



6. If you are not using an ENCX24J600 family device, then the last 5 integers displayed here are the SSL\_P, SSL\_Q, SSL\_dP, SSL\_dQ, and SSL\_qlnv parameters, respectively. However, they are displayed here in big-endian format, and the Microchip cryptographic library implementation requires parameters in little-endian format, so you will have to enter the parameters into the C arrays in opposite order. For example, the INTEGER at offset 145:

```

145:d=1 h1=2 l= 33 prim: INTEGER
:D777566780029FCD610200B66D89507D
915E3E5BDB6FAB0233B5DFA2E4081DF7
    
```

will be swapped in the CustomSSLCert.c file:

```

ROM BYTE SSL_P[] = {
    0xF7, 0x1D, 0x08, 0xE4, 0xA2, 0xDF, 0xB5, 0x33,
    0x02, 0xAB, 0x6F, 0xDB, 0x5B, 0x3E, 0x5E, 0x91,
    0x7D, 0x50, 0x89, 0x6D, 0xB6, 0x00, 0x02, 0x61,
    0xCD, 0x9F, 0x02, 0x80, 0x67, 0x56, 0x77, 0xD7
};
    
```

7. If you are using an ENCX24J600 family device, then the second and fourth integers displayed here are the SSL\_N and SSL\_D parameters, respectively. There is no need to do an endian format change for these parameters. For the example, the expected SSL\_N and SSL\_D values are shown in the figure below:

```

ROM BYTE SSL_N[] = {
    0xAA, 0x96, 0xCA, 0x97, 0xEA, 0x27, 0xB0, 0xD7,
    0xE9, 0x21, 0xD0, 0x40, 0xD4, 0x2C, 0x09, 0x5A,
    0x2E, 0x3A, 0xE4, 0x12, 0x64, 0x2D, 0x4B, 0x1B,
    0x92, 0xDF, 0x79, 0x68, 0x4E, 0x3C, 0x51, 0xF4,
    0x43, 0x48, 0x0D, 0xF2, 0xCB, 0x50, 0x9B, 0x6E,
    0xE5, 0xEA, 0xFE, 0xEF, 0xD9, 0x10, 0x41, 0x08,
    0x14, 0xF9, 0x85, 0x49, 0xFC, 0x50, 0xD3, 0x57,
    0x34, 0xDC, 0x3A, 0x0D, 0x79, 0xF8, 0xD3, 0x99
};

ROM BYTE SSL_D[] = {
    0x68, 0x5B, 0x6E, 0xFC, 0x98, 0x46, 0x47, 0xAD,
    0x0D, 0xF5, 0x9D, 0x8C, 0xCB, 0x3F, 0x55, 0x49,
    0x51, 0xD6, 0xBD, 0x07, 0x73, 0x39, 0x32, 0x25,
    0x60, 0x28, 0x4E, 0x01, 0x3D, 0x3B, 0x31, 0x76,
    0x98, 0x94, 0x9F, 0x7E, 0x8F, 0xC2, 0x95, 0x37,
    0xE5, 0x76, 0x7E, 0x53, 0x44, 0x21, 0xBC, 0xD6,
    0x80, 0x11, 0xB2, 0xB1, 0x80, 0xCF, 0x1B, 0x09,
    0x5D, 0x2E, 0xE8, 0x31, 0xB2, 0x56, 0xE5, 0xB1
};
    
```

8. Parse your X.509 certificate using the command: `openssl.exe asn1parse -in "[directory containing your cert]\512bits.crt" -out cert.bin`

9. Open the cert.bin output file in a hex editor. For example, here is the default certificate information generated from 512bits.crt given in the stack:



};

11. Update the `SSL_CERT_LEN` (see page 404) variable to contain the correct value.

## 10.17.2 SSL Public Members

### Enumerations

	Name	Description
	<code>SSL_SUPPLEMENTARY_DATA_TYPES</code> (see page 380)	This is type <code>SSL_SUPPLEMENTARY_DATA_TYPES</code> .

### Functions

	Name	Description
	<code>TCPAddSSLListener</code> (see page 378)	Listens for SSL connection on a specific port.
	<code>TCPSSLIsHandshaking</code> (see page 378)	Determines if an SSL session is still handshaking.
	<code>TCPStartSSLClient</code> (see page 379)	Begins an SSL client session.
	<code>TCPisSSL</code> (see page 379)	Determines if a TCP connection is secured with SSL.
	<code>SSLStartSession</code> (see page 380)	Begins a new SSL session for the given TCP connection.

### Macros

	Name	Description
	<code>SSL_INVALID_ID</code> (see page 377)	Identifier for invalid SSL allocations

### Module

SSL (see page 373)

### Structures

	Name	Description
	<code>SSL_PKEY_INFO</code> (see page 380)	To hash the public key information, we need to actually get the public key information... 1024 bit key at 8 bits/byte = 128 bytes needed for the public key.

### Description

The following functions and variables are available to the stack application.

## 10.17.2.1 SSL\_INVALID\_ID Macro

### File

SSL.h

### C

```
#define SSL_INVALID_ID (0xFFu) // Identifier for invalid SSL allocations
```

### Description

Identifier for invalid SSL allocations

## 10.17.2.2 TCPAddSSLListener Function

**File**

TCP.h

**C**

```

BOOL TCPAddSSLListener(
    TCP_SOCKET hTCP,
    WORD port
);
    
```

**Description**

This function adds an additional listening port to a TCP connection. Connections made on this alternate port will be secured via SSL.

**Preconditions**

TCP is initialized and hTCP is listening.

**Parameters**

Parameters	Description
hTCP	TCP connection to secure
port	SSL port to listen (see page 169) on

**Return Values**

Return Values	Description
TRUE	SSL port was added.
FALSE	The socket was not a listening socket.

## 10.17.2.3 TCPSSLIsHandshaking Function

**File**

TCP.h

**C**

```

BOOL TCPSSLIsHandshaking(
    TCP_SOCKET hTCP
);
    
```

**Description**

Call this function after calling TCPStartSSLClient (see page 379) until FALSE is returned. Then your application may continue with its normal data transfer (which is now secured).

**Preconditions**

TCP is initialized and hTCP is connected.

**Parameters**

Parameters	Description
hTCP	TCP connection to check

**Return Values**

Return Values	Description
TRUE	SSL handshake is still progressing
FALSE	SSL handshake has completed

## 10.17.2.4 TCPStartSSLClient Function

### File

TCP.h

### C

```

BOOL TCPStartSSLClient(
    TCP_SOCKET hTCP,
    BYTE* host
);

```

### Description

This function escalates the current connection to an SSL secured connection by initiating an SSL client handshake.

### Remarks

The host parameter is currently ignored and is not validated.

### Preconditions

TCP is initialized and hTCP is already connected.

### Parameters

Parameters	Description
hTCP	TCP connection to secure
host	Expected host name on certificate (currently ignored)

### Return Values

Return Values	Description
TRUE	an SSL connection was initiated
FALSE	Insufficient SSL resources (stubs) were available

## 10.17.2.5 TCPIsSSL Function

### File

TCP.h

### C

```

BOOL TCPIsSSL(
    TCP_SOCKET hTCP
);

```

### Description

Call this function to determine whether or not a TCP connection is secured with SSL.

### Preconditions

TCP is initialized and hTCP is connected.

### Parameters

Parameters	Description
hTCP	TCP connection to check

### Return Values

Return Values	Description
TRUE	Connection is secured via SSL
FALSE	Connection is not secured

## 10.17.2.6 SSLStartSession Function

### File

SSL.h

### C

```
BYTE SSLStartSession(
    TCP_SOCKET hTCP,
    void * buffer,
    BYTE supDataType
);
```

### Description

Begins a new SSL session for the given TCP connection.

### Preconditions

SSL has been initialized and hTCP is connected.

### Parameters

Parameters	Description
hTCP	the socket to begin the SSL connection on
buffer	pointer to a supplementary data buffer
supDataType	type of supplementary data to store

### Return Values

Return Values	Description
SSL_INVALID_ID (see page 377)	insufficient SSL resources to start a new connection
others	the allocated SSL stub ID

## 10.17.2.7 SSL\_SUPPLEMENTARY\_DATA\_TYPES Enumeration

### File

SSL.h

### C

```
typedef enum {
    SSL_SUPPLEMENTARY_DATA_NONE = 0,
    SSL_SUPPLEMENTARY_DATA_CERT_PUBLIC_KEY
} SSL_SUPPLEMENTARY_DATA_TYPES;
```

### Description

This is type SSL\_SUPPLEMENTARY\_DATA\_TYPES.

## 10.17.2.8 SSL\_PKEY\_INFO Structure

### File

SSL.h

### C

```
typedef struct {
    WORD pub_size_bytes;
    BYTE pub_key[128];
    BYTE pub_e[3];
    BYTE pub_guid;
} SSL_PKEY_INFO;
```

**Members**

Members	Description
BYTE pub_guid;	This is used as a TCP_SOCKET (see page 462) which is a BYTE

**Description**

To hash the public key information, we need to actually get the public key information... 1024 bit key at 8 bits/byte = 128 bytes needed for the public key.

## 10.17.3 SSL Stack Members

**Enumerations**

	Name	Description
	SSL_STATE (see page 382)	This is type SSL_STATE.

**Functions**

	Name	Description
	SSLInit (see page 382)	Initializes the SSL engine.
	SSLPeriodic (see page 382)	Performs any periodic tasks for the SSL module.
	TCPRequestSSLMessage (see page 383)	Requests an SSL message to be transmitted.
	TCPSSLGetPendingTxSize (see page 383)	Determines how many bytes are pending for a future SSL record.
	TCPSSLHandleIncoming (see page 384)	Hands newly arrive TCP data to the SSL module for processing.
	TCPSSLHandshakeComplete (see page 384)	Clears the SSL handshake flag.
	TCPSSLInPlaceMACEncrypt (see page 385)	Encrypts and MACs data in place in the TCP TX buffer.
	TCPSSLPutRecordHeader (see page 385)	Writes an SSL record header and sends an SSL record.
	TCPStartSSLServer (see page 386)	Begins an SSL server session.

**Macros**

	Name	Description
	SSL_MIN_SESSION_LIFETIME (see page 386)	Minimum lifetime for SSL Sessions Sessions cannot be reallocated until this much time has elapsed
	SSL_RSA_LIFETIME_EXTENSION (see page 387)	Lifetime extension for RSA operations Sessions lifetime is extended by this amount when an RSA calculation is made

**Module**

SSL (see page 373)

**Description**

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.17.3.1 SSL\_STATE Enumeration

#### File

TCP.h

#### C

```
typedef enum {
    SSL_NONE = 0,
    SSL_HANDSHAKING,
    SSL_ESTABLISHED,
    SSL_CLOSED
} SSL_STATE;
```

#### Members

Members	Description
SSL_NONE = 0	No security is enabled
SSL_HANDSHAKING	Handshake is progressing (no application data allowed)
SSL_ESTABLISHED	Connection is established and secured
SSL_CLOSED	Connection has been closed (no applicaiton data is allowed)

#### Description

This is type SSL\_STATE.

### 10.17.3.2 SSLInit Function

#### File

SSL.h

#### C

```
void SSLInit();
```

#### Returns

None

#### Description

Initializes the SSL engine.

#### Remarks

This function is called only one during lifetime of the application.

#### Preconditions

None

#### Section

Function Prototypes

### 10.17.3.3 SSLPeriodic Function

#### File

SSL.h

#### C

```
void SSLPeriodic(
    TCP_SOCKET hTCP,
```

```

    BYTE sslStubID
);

```

**Returns**

None

**Description**

This function performs periodic tasks for the SSL module. This includes processing for RSA operations.

**Preconditions**

SSL has already been initialized.

**Parameters**

Parameters	Description
hTCP	the socket for which to perform periodic functions
id	the SSL stub to use

### 10.17.3.4 TCPRequestSSLMessage Function

**File**

TCP.h

**C**

```

BOOL TCPRequestSSLMessage(
    TCP_SOCKET hTCP,
    BYTE msg
);

```

**Description**

This function is called to request that a specific SSL message be transmitted. This message should only be called by the SSL module.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	TCP connection to use
msg	One of the SSL_MESSAGE types to transmit.

**Return Values**

Return Values	Description
TRUE	The message was requested.
FALSE	Another message is already pending transmission.

### 10.17.3.5 TCPSSLGetPendingTxSize Function

**File**

TCP.h

**C**

```

WORD TCPSSLGetPendingTxSize(
    TCP_SOCKET hTCP
);

```

**Returns**

None

**Description**

This function determines how many bytes are pending for a future SSL record.

**Preconditions**

TCP is initialized, and hTCP is connected with an active SSL connection.

**Parameters**

Parameters	Description
hTCP	TCP connection to check

## 10.17.3.6 TCPSSLHandleIncoming Function

**File**

TCP.h

**C**

```
void TCPSSLHandleIncoming(
    TCP_SOCKET hTCP
);
```

**Returns**

None

**Description**

This function processes incoming TCP data as an SSL record and performs any necessary repositioning and decrypting.

**Remarks**

This function should never be called by an application. It is used only by the SSL module itself.

**Preconditions**

TCP is initialized, and hTCP is connected with an active SSL session.

**Parameters**

Parameters	Description
hTCP	TCP connection to handle incoming data on

## 10.17.3.7 TCPSSLHandshakeComplete Function

**File**

TCP.h

**C**

```
void TCPSSLHandshakeComplete(
    TCP_SOCKET hTCP
);
```

**Returns**

None

**Description**

This function clears the flag indicating that an SSL handshake is complete.

**Remarks**

This function should never be called by an application. It is used only by the SSL module itself.

**Preconditions**

TCP is initialized and hTCP is connected.

**Parameters**

Parameters	Description
hTCP	TCP connection to set

## 10.17.3.8 TCPSSLInPlaceMACEncrypt Function

**File**

TCP.h

**C**

```
void TCPSSLInPlaceMACEncrypt(
    TCP_SOCKET hTCP,
    ARCFOUR_CTX* ctx,
    BYTE* MACSecret,
    WORD len
);
```

**Returns**

None

**Description**

This function encrypts data in the TCP buffer while calculating a MAC. When encryption is finished, the MAC is appended to the buffer and the record will be ready to transmit.

**Remarks**

This function should never be called by an application. It is used only by the SSL module itself.

**Preconditions**

TCP is initialized, hTCP is connected, and ctx's Sbox is loaded.

**Parameters**

Parameters	Description
hTCP	TCP connection to encrypt in
ctx	ARCFOUR encryption context to use
MACSecret	MAC encryption secret to use
len	Number of bytes to crypt

## 10.17.3.9 TCPSSLPutRecordHeader Function

**File**

TCP.h

**C**

```
void TCPSSLPutRecordHeader(
    TCP_SOCKET hTCP,
    BYTE* hdr,
    BOOL recDone
);
```

**Returns**

None

**Description**

This function writes an SSL record header to the pending TCP SSL data, then indicates that the data is ready to be sent by moving the txHead pointer.

If the record is complete, set recDone to TRUE. The sslTxHead pointer will be moved forward 5 bytes to leave space for a future record header. If the record is only partially sent, use FALSE and to leave the pointer where it is so that more data can be added to the record. Partial records can only be used for the SERVER\_CERTIFICATE handshake message.

**Remarks**

This function should never be called by an application. It is used only by the SSL module itself.

**Preconditions**

TCP is initialized, and hTCP is connected with an active SSL session.

**Parameters**

Parameters	Description
hTCP	TCP connection to write the header and transmit with
hdr	Record header (5 bytes) to send or NULL to just move the pointerctx
recDone	TRUE if the record is done, FALSE otherwise

### 10.17.3.10 TCPStartSSLServer Function

**File**

TCP.h

**C**

```

BOOL TCPStartSSLServer(
    TCP_SOCKET hTCP
);
    
```

**Description**

This function sets up an SSL server session when a new connection is established on an SSL port.

**Preconditions**

TCP is initialized and hTCP is already connected.

**Parameters**

Parameters	Description
hTCP	TCP connection to secure

**Return Values**

Return Values	Description
TRUE	an SSL connection was initiated
FALSE	Insufficient SSL resources (stubs) were available

### 10.17.3.11 SSL\_MIN\_SESSION\_LIFETIME Macro

**File**

SSL.h

**C**

```
#define SSL_MIN_SESSION_LIFETIME (1*TICK_SECOND)
```

**Description**

Minimum lifetime for SSL Sessions Sessions cannot be reallocated until this much time has elapsed

### 10.17.3.12 SSL\_RSA\_LIFETIME\_EXTENSION Macro

**File**

SSL.h

**C**

```
#define SSL_RSA_LIFETIME_EXTENSION (8*TICK_SECOND)
```

**Description**

Lifetime extension for RSA operations Sessions lifetime is extended by this amount when an RSA calculation is made

## 10.17.4 SSL Internal Members

**Enumerations**

	Name	Description
	SM_SSL_RX_SERVER_HELLO (see page 401)	State machine for SSLRxServerHello (see page 423)
	SSL_ALERT_LEVEL (see page 401)	Describes the two types of Alert records
	SSL_MESSAGES (see page 406)	Describes the types of SSL messages (handshake and alerts)
	SSL_SESSION_TYPE (see page 409)	SSL Session Type Enumeration

**Functions**

	Name	Description
	CalculateFinishedHash (see page 392)	Calculates the handshake hash over the data. hashID can be either MD5 or SHA-1, and this function will calculate accordingly.
	GenerateHashRounds (see page 393)	Generates hash rounds to find either the Master Secret or the Key Block.
	GenerateSessionKeys (see page 393)	Generates the session write keys and MAC secrets
	HSEnd (see page 394)	Hashes (see page 196) the message contents into the Handshake hash structures and begins a new handshake hash.
	HSGet (see page 394)	Reads data from socket, transparently hashing it into the handshake hashes.

	<p>HSGetArray (<a href="#">see page 395</a>)</p>	<ul style="list-style-type: none"> <li>• Function: static WORD HSGetArray(TCP_SOCKET (<a href="#">see page 462</a>) skt, BYTE *data, WORD len)</li> <li>*</li> <li>• PreCondition: None</li> <li>*</li> <li>• Input: skt - socket to read data from</li> <li>• data - array to read into, or NULL</li> <li>• len - number of bytes to read</li> <li>*</li> <li>• Output: Number of bytes read</li> <li>*</li> <li>• Side Effects: None</li> <li>*</li> <li>• Overview: Reads data from socket, transparently hashing it into the handshake hashes.</li> <li>*</li> <li>• Note: None</li> </ul>
	<p>HSGetWord (<a href="#">see page 395</a>)</p>	<p>Reads data from socket, transparently hashing it into the handshake hashes.</p>
	<p>HSPut (<a href="#">see page 396</a>)</p>	<p>Writes data to socket, transparently hashing it into the handshake hashes.</p>
	<p>HSPutArray (<a href="#">see page 396</a>)</p>	<ul style="list-style-type: none"> <li>• Function: static WORD HSPutArray(TCP_SOCKET (<a href="#">see page 462</a>) skt, BYTE *data, BYTE len)</li> <li>*</li> <li>• PreCondition: None</li> <li>*</li> <li>• Input: skt - socket to write data to</li> <li>• data - data to write</li> <li>• len - number of bytes to write</li> <li>*</li> <li>• Output: Number of bytes written</li> <li>*</li> <li>• Side Effects: None</li> <li>*</li> <li>• Overview: Writes data to socket, transparently hashing it into the handshake hashes.</li> <li>*</li> <li>• Note: None</li> </ul>
	<p>HSPutROMArray (<a href="#">see page 397</a>)</p>	<p>This is function HSPutROMArray.</p>
	<p>HSPutWord (<a href="#">see page 397</a>)</p>	<p>Writes data to socket, transparently hashing it into the handshake hashes.</p>
	<p>HSStart (<a href="#">see page 398</a>)</p>	<p>Sets up the buffer to store data for handshake hash tracking</p>
	<p>LoadOffChip (<a href="#">see page 400</a>)</p>	<p>Copies data from Ethernet RAM to local RAM</p>

⇒	SaveOffChip (see page 400)	Copies data in PIC RAM to the Ethernet RAM
⇒	SSLBufferAlloc (see page 412)	Allocates a buffer for use.
⇒	SSLBufferFree (see page 412)	Specified buffer is released
⇒	SSLBufferSync (see page 413)	Specified buffer is loaded to RAM. Only loads if necessary, and saves any current buffer before switching.
⇒	SSLHashAlloc (see page 414)	Allocates a hash for use.
⇒	SSLHashFree (see page 415)	Specified hash is released
⇒	SSLHashSync (see page 415)	Specified hash is loaded to RAM. Only loads if necessary, and saves any current hash before switching.
⇒	SSLKeysSync (see page 416)	Specified key set is loaded to RAM. Only loads if necessary, and saves any current key set before switching.
⇒	SSLMACAdd (see page 417)	This is function SSLMACAdd.
⇒	SSLMACBegin (see page 417)	This is function SSLMACBegin.
⇒	SSLMACCalc (see page 417)	This is function SSLMACCalc.
⇒	SSLRSAOperation (see page 418)	Pauses connection processing until RSA calculation is complete.
⇒	SSLRxAlert (see page 418)	Receives an alert message and decides what to do
⇒	SSLRxAntiqueClientHello (see page 419)	Receives the SSLv2 ClientHello message, initiating a new SSL session with a client
⇒	SSLRxCCS (see page 419)	Receives a ChangeCipherSpec from the remote server
⇒	SSLRxClientHello (see page 420)	Receives the ClientHello message, initiating a new SSL session with a client
⇒	SSLRxClientKeyExchange (see page 421)	Receives the ClientKeyExchange message and begins the decryption process.
⇒	SSLRxFinished (see page 421)	Receives the Finished message from remote node
⇒	SSLRxHandshake (see page 422)	Receives a handshake message.
⇒	SSLRxRecord (see page 422)	Receives an SSL record.
⇒	SSLRxServerCertificate (see page 423)	Receives ServerCertificate from the remote server, locates the public key information, and executes RSA operation.
⇒	SSLRxServerHello (see page 423)	Receives the ServerHello from the remote server
⇒	SSLSessionMatchID (see page 424)	Locates a cached SSL session for reuse. Syncs found session into RAM.
⇒	SSLSessionMatchIP (see page 425)	Locates a cached SSL session for reuse
⇒	SSLSessionNew (see page 425)	Finds space for a new SSL session
⇒	SSLSessionSync (see page 426)	Specified session is loaded to RAM. Only loads if necessary, and saves any current session before switching if it has been updated.
⇒	SSLStartPartialRecord (see page 427)	Begins a long SSL record.
⇒	SSLStubAlloc (see page 428)	Allocates a stub for use.

⇒	SSLStubFree (see page 428)	Specified stub is released
⇒	SSLStubSync (see page 429)	Specified stub is loaded to RAM. Only loads if necessary, and saves any current stub before switching.
⇒	SSLTerminate (see page 430)	Terminates an SSL connection and releases allocated resources.
⇒	SSLTxCCSFin (see page 430)	Generates the session keys from the master secret, then allocates and generates the encryption context. Once processing is complete, transmits the Change Cipher Spec message and the Finished handshake message to the server.
⇒	SSLTxClientHello (see page 431)	Transmits the ClientHello message to initiate a new SSL session with the server.
⇒	SSLTxClientKeyExchange (see page 431)	Transmits the encrypted pre-master secret to the server and requests the Change Cipher Spec. Also generates the Master Secret from the pre-master secret that was used.
⇒	SSLTxMessage (see page 432)	Transmits an SSL message.
⇒	SSLTxRecord (see page 432)	Transmits an SSL record.
⇒	SSLTxServerCertificate (see page 433)	Transmits the Certificate message with the server's specified public key certificate.
⇒	SSLTxServerHello (see page 433)	Transmits the ServerHello message.
⇒	SSLTxServerHelloDone (see page 434)	Transmits the ServerHelloDone message.

**Macros**

	<b>Name</b>	<b>Description</b>
⇒	RESERVED_SSL_MEMORY (see page 399)	Total space needed by all SSL storage requirements
⇒	SSL_ALERT (see page 401)	Protocol code for Alert records
⇒	SSL_APPLICATION (see page 402)	Protocol code for Application data records
⇒	SSL_BASE_BUFFER_ADDR (see page 402)	Base address for SSL buffers
⇒	SSL_BASE_HASH_ADDR (see page 402)	Base address for SSL hashes
⇒	SSL_BASE_KEYS_ADDR (see page 402)	Base address for SSL keys
⇒	SSL_BASE_SESSION_ADDR (see page 403)	Base address for SSL sessions
⇒	SSL_BASE_STUB_ADDR (see page 403)	Base address for SSL stubs
⇒	SSL_BUFFER_SIZE (see page 403)	Amount of space needed by a single SSL buffer
⇒	SSL_BUFFER_SPACE (see page 404)	Amount of space needed by all SSL buffer
⇒	SSL_CHANGE_CIPHER_SPEC (see page 404)	Protocol code for Change Cipher Spec records
⇒	SSL_HANDSHAKE (see page 405)	Protocol code for Handshake records
⇒	SSL_HASH_SIZE (see page 405)	Amount of space needed by a single SSL hash
⇒	SSL_HASH_SPACE (see page 405)	Amount of space needed by all SSL hash

	SSL_KEYS_SIZE (see page 406)	Amount of space needed by a single SSL key
	SSL_KEYS_SPACE (see page 406)	Amount of space needed by all SSL key
	SSL_RSA_EXPORT_WITH_ARCFOUR_40_MD5 (see page 407)	This is macro SSL_RSA_EXPORT_WITH_ARCFOUR_40_MD5.
	SSL_RSA_WITH_ARCFOUR_128_MD5 (see page 407)	This is macro SSL_RSA_WITH_ARCFOUR_128_MD5.
	SSL_SESSION_SIZE (see page 408)	Amount of space needed by a single SSL session
	SSL_SESSION_SPACE (see page 408)	Amount of space needed by all SSL session
	SSL_STUB_SIZE (see page 411)	Amount of space needed by a single SSL stub
	SSL_STUB_SPACE (see page 411)	Amount of space needed by all SSL stubs
	SSL_VERSION (see page 411)	SSL version number
	SSL_VERSION_HI (see page 411)	SSL version number (high byte)
	SSL_VERSION_LO (see page 411)	SSL version number (low byte)
	SSLFinishPartialRecord (see page 413)	This is macro SSLFinishPartialRecord.
	SSLFlushPartialRecord (see page 414)	This is macro SSLFlushPartialRecord.
	SSLSessionUpdated (see page 427)	This is macro SSLSessionUpdated.

**Module**

SSL (see page 373)

**Structures**

	Name	Description
	SSL_KEYS (see page 405)	Memory definition for SSL keys. This area is split into Local and Remote areas. During the handshake, Local.random and Remote.random hold the ServerRandom and ClientRandom values. Once the session keys are calculated, the Local.app and Remote.app contain the MAC secret, record sequence number, and encryption context for the ARCFOUR module.
	SSL_SESSION (see page 408)	Storage space for SSL Session identifiers. (The SessionID and MasterSecret)
	SSL_SESSION_STUB (see page 409)	Stub value for an SSL_SESSION (see page 408). The tag associates this session with a remote node, either by matching to a remote IP address when we are the client or the first 3 bytes of the session ID when we are the host. When a session is free/expired, the tag is 0x00000000. The lastUsed value is the Tick count when the session was last used so that older sessions may be overwritten first.
	SSL_STUB (see page 409)	Memory holder for general information associated with an SSL connections.

**Unions**

	Name	Description
	SSL_BUFFER (see page 403)	Generic buffer space for SSL. The hashRounds element is used when this buffer is needed for handshake hash calculations, and the full element is used as the Sbox for ARCFOUR calculations.

**Variables**

	Name	Description
	isBufferUsed (see page 398)	Indicates which buffers are in use
	isHashUsed (see page 399)	Indicates which hashes are in use
	isStubUsed (see page 399)	Indicates which stubs are in use
	masks (see page 399)	Masks for each bit in the is*Used variables
	ptrHS (see page 399)	Used in buffering handshake results
	SSL_CERT (see page 404)	RSA public certificate data ?

◆	SSL_CERT_LEN (see page 404)	RSA public certificate length ?
◆	sslBufferID (see page 413)	Which buffer is loaded
◆	sslHash (see page 414)	Hash storage
◆	sslHashID (see page 415)	Which hash is loaded
◆	sslKeys (see page 416)	The current SSL session
◆	sslKeysID (see page 416)	Which SSL_KEYS (see page 405) are loaded
◆	sslRSASStubID (see page 418)	Which stub is using RSA, if any
◆	sslSession (see page 424)	Current session data
◆	sslSessionID (see page 424)	Which session is loaded
◆	sslSessionStubs (see page 426)	8 byte session stubs
◆	sslSessionUpdated (see page 427)	Whether or not it has been updated
◆	sslStub (see page 428)	The current SSL stub
◆	sslStubID (see page 429)	Which SSL_STUB (see page 409) is loaded

**Description**

The following functions and variables are designated as internal to the SSL module.

### 10.17.4.1 CalculateFinishedHash Function

**File**

SSL.c

**C**

```
static void CalculateFinishedHash(
    BYTE hashID,
    BOOL fromClient,
    BYTE * result
);
```

**Side Effects**

None

**Returns**

None

**Description**

Calculates the handshake hash over the data. hashID can be either MD5 or SHA-1, and this function will calculate accordingly.

**Remarks**

None

**Preconditions**

hashID has all handshake data hashed so far and the current session is synced in.

**Parameters**

Parameters	Description
hashID	the hash sum to use
fromClient	TRUE if client is sender
result	where to store results

## 10.17.4.2 GenerateHashRounds Function

### File

SSL.c

### C

```
static void GenerateHashRounds(
    BYTE num,
    BYTE* rand1,
    BYTE* rand2
);
```

### Side Effects

Destroys the SSL Buffer space

### Returns

None

### Description

Generates hash rounds to find either the Master Secret or the Key Block.

### Remarks

This function will overflow the buffer after 7 rounds, but in practice num = 3 or num = 4.

### Preconditions

The SSL buffer is allocated for temporary usage and the data to run rounds on is in sslSession.masterSecret

### Parameters

Parameters	Description
num	how many rounds to compute
rand1	the first random data block to use
rand2	the second random data block to use

## 10.17.4.3 GenerateSessionKeys Function

### File

SSL.c

### C

```
static void GenerateSessionKeys();
```

### Side Effects

Destroys the SSL Buffer Space

### Returns

None

### Description

Generates the session write keys and MAC secrets

### Remarks

None

### Preconditions

The SSL buffer is allocated for temporary usage, session keys are synced, and the TX and RX buffers are allocated for

S-boxes.

### 10.17.4.4 HSEnd Function

**File**

SSL.c

**C**

```
static void HSEnd();
```

**Side Effects**

None

**Returns**

None

**Description**

Hashes (see page 196) the message contents into the Handshake hash structures and begins a new handshake hash.

**Remarks**

None

**Preconditions**

None

### 10.17.4.5 HSGet Function

**File**

SSL.c

**C**

```
static WORD HSGet(
    TCP_SOCKET skt,
    BYTE * b
);
```

**Side Effects**

None

**Returns**

Number of bytes read

**Description**

Reads data from socket, transparently hashing it into the handshake hashes.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
skt	socket to read data from
b	byte to read into

## 10.17.4.6 HSGetArray Function

### File

SSL.c

### C

```
static WORD HSGetArray(  
    TCP_SOCKET skt,  
    BYTE * data,  
    WORD len  
);
```

### Description

- Function: static WORD HSGetArray(TCP\_SOCKET (see page 462) *skt*, BYTE \**data*, WORD *len*)  
\*
- PreCondition: None  
\*
- Input: *skt* - socket to read data from
- *data* - array to read into, or NULL
- *len* - number of bytes to read  
\*
- Output: Number of bytes read  
\*
- Side Effects: None  
\*
- Overview: Reads data from socket, transparently hashing it  
• into the handshake hashes.  
\*
- Note: None

## 10.17.4.7 HSGetWord Function

### File

SSL.c

### C

```
static WORD HSGetWord(  
    TCP_SOCKET skt,  
    WORD * w  
);
```

### Side Effects

None

### Returns

Number of bytes read

### Description

Reads data from socket, transparently hashing it into the handshake hashes.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
skt	socket to read data from
w	word to read into

## 10.17.4.8 HSPut Function

**File**

SSL.c

**C**

```
static WORD HSPut(
    TCP_SOCKET skt,
    BYTE b
);
```

**Side Effects**

None

**Returns**

Number of bytes written

**Description**

Writes data to socket, transparently hashing it into the handshake hashes.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
skt	socket to write data to
b	byte to write

## 10.17.4.9 HSPutArray Function

**File**

SSL.c

**C**

```
static WORD HSPutArray(
    TCP_SOCKET skt,
    BYTE * data,
    WORD len
);
```

**Description**

- Function: static WORD HSPutArray(TCP\_SOCKET (see page 462) skt, BYTE \*data, BYTE len)
- \*
- PreCondition: None
- \*
- Input: skt - socket to write data to
- data - data to write
- len - number of bytes to write
- \*
- Output: Number of bytes written
- \*
- Side Effects: None
- \*
- Overview: Writes data to socket, transparently hashing it
- into the handshake hashes.
- \*
- Note: None

## 10.17.4.10 HSPutROMArray Function

**File**

SSL.c

**C**

```
static WORD HSPutROMArray(
    TCP_SOCKET skt,
    ROM BYTE * data,
    WORD len
);
```

**Description**

This is function HSPutROMArray.

## 10.17.4.11 HSPutWord Function

**File**

SSL.c

**C**

```
static WORD HSPutWord(
    TCP_SOCKET skt,
    WORD w
);
```

**Side Effects**

None

**Returns**

Number of bytes written

**Description**

Writes data to socket, transparently hashing it into the handshake hashes.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
skt	socket to write data to
w	word to write

## 10.17.4.12 HSStart Function

**File**

SSL.c

**C**

```
static void HSStart();
```

**Side Effects**

None

**Returns**

None

**Description**

Sets up the buffer to store data for handshake hash tracking

**Remarks**

None

**Preconditions**

None

**Section**

Handshake Hash and I/O Functions

## 10.17.4.13 isBufferUsed Variable

**File**

SSL.c

**C**

```
WORD isBufferUsed;
```

**Description**

Indicates which buffers are in use

## 10.17.4.14 isHashUsed Variable

### File

SSL.c

### C

```
WORD isHashUsed;
```

### Description

Indicates which hashes are in use

## 10.17.4.15 isStubUsed Variable

### File

SSL.c

### C

```
WORD isStubUsed;
```

### Description

Indicates which stubs are in use

## 10.17.4.16 masks Variable

### File

SSL.c

### C

```
ROM WORD masks[16] = { 0x0001, 0x0002, 0x0004, 0x0008, 0x0010, 0x0020, 0x0040, 0x0080,  
0x0100, 0x0200, 0x0400, 0x0800, 0x1000, 0x2000, 0x4000, 0x8000 };
```

### Description

Masks for each bit in the is\*Used variables

## 10.17.4.17 ptrHS Variable

### File

SSL.c

### C

```
BYTE * ptrHS;
```

### Description

Used in buffering handshake results

## 10.17.4.18 RESERVED\_SSL\_MEMORY Macro

### File

SSL.h

**C**

```
#define RESERVED_SSL_MEMORY ((DWORD)(SSL_STUB_SPACE + SSL_KEYS_SPACE + SSL_HASH_SPACE +
SSL_BUFFER_SPACE + SSL_SESSION_SPACE))
```

**Description**

Total space needed by all SSL storage requirements

## 10.17.4.19 LoadOffChip Function

**File**

SSL.c

**C**

```
static void LoadOffChip(
    BYTE * ramAddr,
    PTR_BASE ethAddr,
    WORD len
);
```

**Side Effects**

None

**Returns**

None

**Description**

Copies data from Ethernet RAM to local RAM

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
ramAddr	destination address in RAM
ethAddr	source address in Ethernet RAM
len	number of bytes to copy

## 10.17.4.20 SaveOffChip Function

**File**

SSL.c

**C**

```
static void SaveOffChip(
    BYTE * ramAddr,
    PTR_BASE ethAddr,
    WORD len
);
```

**Side Effects**

None

**Returns**

None

**Description**

Copies data in PIC RAM to the Ethernet RAM

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
ramAddr	source address in RAM
ethAddr	destination address in Ethernet RAM
len	number of bytes to copy

## 10.17.4.21 SM\_SSL\_RX\_SERVER\_HELLO Enumeration

**File**

SSL.h

**C**

```
typedef enum {
    RX_SERVER_CERT_START = 0u,
    RX_SERVER_CERT_FIND_KEY,
    RX_SERVER_CERT_FIND_N,
    RX_SERVER_CERT_READ_N,
    RX_SERVER_CERT_READ_E,
    RX_SERVER_CERT_CLEAR
} SM_SSL_RX_SERVER_HELLO;
```

**Description**

State machine for SSLRxServerHello (see page 423)

## 10.17.4.22 SSL\_ALERT Macro

**File**

SSL.h

**C**

```
#define SSL_ALERT 21u // Protocol code for Alert records
```

**Description**

Protocol code for Alert records

## 10.17.4.23 SSL\_ALERT\_LEVEL Enumeration

**File**

SSL.h

**C**

```
typedef enum {
```

```

    SSL_ALERT_WARNING = 1u,
    SSL_ALERT_FATAL = 2u
} SSL_ALERT_LEVEL;

```

**Members**

Members	Description
SSL_ALERT_WARNING = 1u	Alert message is a warning (session can be resumed)
SSL_ALERT_FATAL = 2u	Alert message is fatal (session is non-resumable)

**Description**

Describes the two types of Alert records

## 10.17.4.24 SSL\_APPLICATION Macro

**File**

SSL.h

**C**

```
#define SSL_APPLICATION 23u // Protocol code for Application data records
```

**Description**

Protocol code for Application data records

## 10.17.4.25 SSL\_BASE\_BUFFER\_ADDR Macro

**File**

SSL.c

**C**

```
#define SSL_BASE_BUFFER_ADDR (BASE_SSLB_ADDR + SSL_STUB_SPACE + SSL_KEYS_SPACE +
SSL_HASH_SPACE)
```

**Description**

Base address for SSL buffers

## 10.17.4.26 SSL\_BASE\_HASH\_ADDR Macro

**File**

SSL.c

**C**

```
#define SSL_BASE_HASH_ADDR (BASE_SSLB_ADDR + SSL_STUB_SPACE + SSL_KEYS_SPACE)
```

**Description**

Base address for SSL hashes

## 10.17.4.27 SSL\_BASE\_KEYS\_ADDR Macro

**File**

SSL.c

**C**

```
#define SSL_BASE_KEYS_ADDR (BASE_SSLB_ADDR + SSL_STUB_SPACE)
```

**Description**

Base address for SSL keys

**10.17.4.28 SSL\_BASE\_SESSION\_ADDR Macro****File**

SSL.c

**C**

```
#define SSL_BASE_SESSION_ADDR (BASE_SSLB_ADDR + SSL_STUB_SPACE + SSL_KEYS_SPACE +
SSL_HASH_SPACE + SSL_BUFFER_SPACE)
```

**Description**

Base address for SSL sessions

**10.17.4.29 SSL\_BASE\_STUB\_ADDR Macro****File**

SSL.c

**C**

```
#define SSL_BASE_STUB_ADDR (BASE_SSLB_ADDR)
```

**Description**

Base address for SSL stubs

**10.17.4.30 SSL\_BUFFER Union****File**

SSL.h

**C**

```
typedef union {
    struct {
        HASH_SUM hash;
        BYTE md5_hash[16];
        BYTE sha_hash[20];
        BYTE temp[256-sizeof(HASH_SUM)-16-20];
    } hashRounds;
    BYTE full[256];
} SSL_BUFFER;
```

**Description**

Generic buffer space for SSL. The hashRounds element is used when this buffer is needed for handshake hash calculations, and the full element is used as the Sbox for ARCFOUR calculations.

**10.17.4.31 SSL\_BUFFER\_SIZE Macro****File**

SSL.h

**C**

```
#define SSL_BUFFER_SIZE ((DWORD)sizeof(SSL_BUFFER)) // Amount of space
```

*needed by a single SSL buffer*

**Description**

Amount of space needed by a single SSL buffer

## 10.17.4.32 SSL\_BUFFER\_SPACE Macro

**File**

SSL.h

**C**

```
#define SSL_BUFFER_SPACE (SSL_BUFFER_SIZE*MAX_SSL_BUFFERS) // Amount of space needed  
by all SSL buffer
```

**Description**

Amount of space needed by all SSL buffer

## 10.17.4.33 SSL\_CERT Variable

**File**

SSL.c

**C**

```
ROM BYTE SSL_CERT[];
```

**Description**

RSA public certificate data ?

## 10.17.4.34 SSL\_CERT\_LEN Variable

**File**

SSL.c

**C**

```
ROM WORD SSL_CERT_LEN;
```

**Description**

RSA public certificate length ?

## 10.17.4.35 SSL\_CHANGE\_CIPHER\_SPEC Macro

**File**

SSL.h

**C**

```
#define SSL_CHANGE_CIPHER_SPEC 20u // Protocol code for Change Cipher Spec records
```

**Description**

Protocol code for Change Cipher Spec records

### 10.17.4.36 SSL\_HANDSHAKE Macro

**File**

SSL.h

**C**

```
#define SSL_HANDSHAKE 22u           // Protocol code for Handshake records
```

**Description**

Protocol code for Handshake records

### 10.17.4.37 SSL\_HASH\_SIZE Macro

**File**

SSL.h

**C**

```
#define SSL_HASH_SIZE ((DWORD)sizeof(HASH_SUM))           // Amount of space needed by
a single SSL hash
```

**Description**

Amount of space needed by a single SSL hash

### 10.17.4.38 SSL\_HASH\_SPACE Macro

**File**

SSL.h

**C**

```
#define SSL_HASH_SPACE ((DWORD)(SSL_HASH_SIZE*MAX_SSL_HASHES)) // Amount of space needed
by all SSL hash
```

**Description**

Amount of space needed by all SSL hash

### 10.17.4.39 SSL\_KEYS Structure

**File**

SSL.h

**C**

```
typedef struct {
    union {
        struct {
            BYTE MACSecret[16];
            DWORD sequence;
            ARCFOUR_CTX cryptCtx;
            BYTE reserved[8];
        } app;
        BYTE random[32];
    } Local;
    union {
        struct {
            BYTE MACSecret[16];
            DWORD sequence;
        }
    }
};
```

```

    ARCFOUR_CTX cryptCtx;
    BYTE reserved[8];
    } app;
    BYTE random[32];
    } Remote;
} SSL_KEYS;

```

**Members**

Members	Description
BYTE MACSecret[16];	Server's MAC write secret
DWORD sequence;	Server's write sequence number
ARCFOUR_CTX cryptCtx;	Server's write encryption context
BYTE reserved[8];	Future expansion
BYTE random[32];	Server.random value
BYTE MACSecret[16];	Client's MAC write secret
DWORD sequence;	Client's write sequence number
ARCFOUR_CTX cryptCtx;	Client's write encryption context
BYTE reserved[8];	Future expansion
BYTE random[32];	Client.random value

**Description**

Memory definition for SSL keys. This area is split into Local and Remote areas. During the handshake, Local.random and Remote.random hold the ServerRandom and ClientRandom values. Once the session keys are calculated, the Local.app and Remote.app contain the MAC secret, record sequence number, and encryption context for the ARCFOUR module.

### 10.17.4.40 SSL\_KEYS\_SIZE Macro

**File**

SSL.h

**C**

```

#define SSL_KEYS_SIZE ((DWORD)sizeof(SSL_KEYS)) // Amount of space needed by
a single SSL key

```

**Description**

Amount of space needed by a single SSL key

### 10.17.4.41 SSL\_KEYS\_SPACE Macro

**File**

SSL.h

**C**

```

#define SSL_KEYS_SPACE (SSL_KEYS_SIZE*MAX_SSL_CONNECTIONS) // Amount of space needed
by all SSL key

```

**Description**

Amount of space needed by all SSL key

### 10.17.4.42 SSL\_MESSAGES Enumeration

**File**

SSL.h

**C**

```
typedef enum {
    SSL_HELLO_REQUEST = 0u,
    SSL_CLIENT_HELLO = 1u,
    SSL_ANTIQUUE_CLIENT_HELLO = 18u,
    SSL_SERVER_HELLO = 2u,
    SSL_CERTIFICATE = 11u,
    SSL_SERVER_HELLO_DONE = 14u,
    SSL_CLIENT_KEY_EXCHANGE = 16u,
    SSL_FINISHED = 20u,
    SSL_ALERT_CLOSE_NOTIFY = 0u+0x80,
    SSL_ALERT_UNEXPECTED_MESSAGE = 10u+0x80,
    SSL_ALERT_BAD_RECORD_MAC = 20u+0x80,
    SSL_ALERT_HANDSHAKE_FAILURE = 40u+0x80,
    SSL_NO_MESSAGE = 0xff
} SSL_MESSAGES;
```

**Members**

Members	Description
SSL_HELLO_REQUEST = 0u	HelloRequest handshake message (not currently supported)
SSL_CLIENT_HELLO = 1u	ClientHello handshake message
SSL_ANTIQUUE_CLIENT_HELLO = 18u	SSLv2 ClientHello handshake message (Supported for backwards compatibility. This is an internally defined value.)
SSL_SERVER_HELLO = 2u	ServerHello handshake message
SSL_CERTIFICATE = 11u	ServerCertificate handshake message
SSL_SERVER_HELLO_DONE = 14u	ServerHelloDone handshake message
SSL_CLIENT_KEY_EXCHANGE = 16u	ClientKeyExchange handshake message
SSL_FINISHED = 20u	Finished handshake message
SSL_ALERT_CLOSE_NOTIFY = 0u+0x80	CloseNotify alert message (dummy value used internally)
SSL_ALERT_UNEXPECTED_MESSAGE = 10u+0x80	UnexpectedMessage alert message (dummy value used internally)
SSL_ALERT_BAD_RECORD_MAC = 20u+0x80	BadRecordMAC alert message (dummy value used internally)
SSL_ALERT_HANDSHAKE_FAILURE = 40u+0x80	HandshakeFailure alert message (dummy value used internally)
SSL_NO_MESSAGE = 0xff	No message is currently requested (internally used value)

**Description**

Describes the types of SSL messages (handshake and alerts)

### 10.17.4.43 SSL\_RSA\_EXPORT\_WITH\_ARCFOUR\_40\_MD5 Macro

**File**

SSL.c

**C**

```
#define SSL_RSA_EXPORT_WITH_ARCFOUR_40_MD5 0x0003u
```

**Description**

This is macro SSL\_RSA\_EXPORT\_WITH\_ARCFOUR\_40\_MD5.

### 10.17.4.44 SSL\_RSA\_WITH\_ARCFOUR\_128\_MD5 Macro

**File**

SSL.c

**C**

```
#define SSL_RSA_WITH_ARCFOUR_128_MD5 0x0004u
```

**Description**

This is macro SSL\_RSA\_WITH\_ARCFOUR\_128\_MD5.

**10.17.4.45 SSL\_SESSION Structure****File**

SSL.h

**C**

```
typedef struct {
    BYTE sessionID[32];
    BYTE masterSecret[48];
} SSL_SESSION;
```

**Members**

Members	Description
BYTE sessionID[32];	The SSL Session ID for this session
BYTE masterSecret[48];	Associated Master Secret for this session

**Description**

Storage space for SSL Session identifiers. (The SessionID and MasterSecret)

**10.17.4.46 SSL\_SESSION\_SIZE Macro****File**

SSL.h

**C**

```
#define SSL_SESSION_SIZE ((DWORD)sizeof(SSL_SESSION)) // Amount of space needed
by a single SSL session
```

**Description**

Amount of space needed by a single SSL session

**10.17.4.47 SSL\_SESSION\_SPACE Macro****File**

SSL.h

**C**

```
#define SSL_SESSION_SPACE (SSL_SESSION_SIZE*MAX_SSL_SESSIONS) // Amount of space
needed by all SSL session
```

**Description**

Amount of space needed by all SSL session

### 10.17.4.48 SSL\_SESSION\_STUB Structure

**File**

SSL.h

**C**

```
typedef struct {
    DWORD_VAL tag;
    DWORD lastUsed;
} SSL_SESSION_STUB;
```

**Members**

Members	Description
DWORD_VAL tag;	Identifying tag for connection When we're a client, this is the remote IP When we're a host, this is 0x00 followed by first 3 bytes of session ID When this stub is free/expired, this is 0x00
DWORD lastUsed;	Tick count when session was last used

**Description**

Stub value for an SSL\_SESSION (see page 408). The tag associates this session with a remote node, either by matching to a remote IP address when we are the client or the first 3 bytes of the session ID when we are the host. When a session is free/expired, the tag is 0x00000000. The lastUsed value is the Tick count when the session was last used so that older sessions may be overwritten first.

### 10.17.4.49 SSL\_SESSION\_TYPE Enumeration

**File**

SSL.h

**C**

```
typedef enum {
    SSL_CLIENT,
    SSL_SERVER
} SSL_SESSION_TYPE;
```

**Members**

Members	Description
SSL_CLIENT	Local device is the SSL client
SSL_SERVER	Local device is the SSL host

**Description**

SSL Session Type Enumeration

### 10.17.4.50 SSL\_STUB Structure

**File**

SSL.h

**C**

```
typedef struct {
    WORD wRxBytesRem;
    WORD wRxHsBytesRem;
    BYTE rxProtocol;
    BYTE rxHSType;
```

```

BYTE idSession;
BYTE idMD5, idSHA1;
BYTE idRxHash;
BYTE idRxBuffer, idTxBuffer;
DWORD_VAL dwTemp;
struct {
    unsigned char bIsServer : 1;
    unsigned char bClientHello : 1;
    unsigned char bServerHello : 1;
    unsigned char bServerCertificate : 1;
    unsigned char bServerHelloDone : 1;
    unsigned char bClientKeyExchange : 1;
    unsigned char bRemoteChangeCipherSpec : 1;
    unsigned char bRemoteFinished : 1;
    unsigned char bLocalChangeCipherSpec : 1;
    unsigned char bLocalFinished : 1;
    unsigned char bExpectingMAC : 1;
    unsigned char bNewSession : 1;
    unsigned char bCloseNotify : 1;
    unsigned char bDone : 1;
    unsigned char bRSAINProgress : 1;
    unsigned char bKeysValid : 1;
} Flags;
BYTE requestedMessage;
void * supplementaryBuffer;
BYTE supplementaryDataType;
} SSL_STUB;
    
```

**Members**

Members	Description
WORD wRxBytesRem;	Bytes left to read in current record
WORD wRxHsBytesRem;	Bytes left to read in current Handshake submessage
BYTE rxProtocol;	Protocol for message being read
BYTE rxHSType;	Handshake message being received
BYTE idSession;	ID for associated session
BYTE idRxHash;	ID for MAC hash (TX needs no persistence)
DWORD_VAL dwTemp;	Used for state machine in RxCertificate
unsigned char blsServer : 1;	We are the server
unsigned char bClientHello : 1;	ClientHello has been sent/received
unsigned char bServerHello : 1;	ServerHello has been sent/received
unsigned char bServerCertificate : 1;	ServerCertificate has been sent/received
unsigned char bServerHelloDone : 1;	ServerHelloDone has been sent/received
unsigned char bClientKeyExchange : 1;	ClientKeyExchange has been sent/received
unsigned char bRemoteChangeCipherSpec : 1;	Remote node has sent a ChangeCipherSpec message
unsigned char bRemoteFinished : 1;	Remote node has sent a Finished message
unsigned char bLocalChangeCipherSpec : 1;	We have sent a ChangeCipherSpec message
unsigned char bLocalFinished : 1;	We have sent a Finished message
unsigned char bExpectingMAC : 1;	We expect a MAC at end of message
unsigned char bNewSession : 1;	TRUE if a new session, FALSE if resuming
unsigned char bCloseNotify : 1;	Whether or not a CloseNotify has been sent/received
unsigned char bDone : 1;	TRUE if the connection is closed
unsigned char bRSAINProgress : 1;	TRUE when RSA op is in progress
unsigned char bKeysValid : 1;	TRUE if the session keys have been generated
BYTE requestedMessage;	Currently requested message to send, or 0xff

**Description**

Memory holder for general information associated with an SSL connections.

### 10.17.4.51 SSL\_STUB\_SIZE Macro

**File**

SSL.h

**C**

```
#define SSL_STUB_SIZE ((DWORD)sizeof(SSL_STUB))           // Amount of space needed by  
a single SSL stub
```

**Description**

Amount of space needed by a single SSL stub

### 10.17.4.52 SSL\_STUB\_SPACE Macro

**File**

SSL.h

**C**

```
#define SSL_STUB_SPACE (SSL_STUB_SIZE*MAX_SSL_CONNECTIONS) // Amount of space needed  
by all SSL stubs
```

**Description**

Amount of space needed by all SSL stubs

### 10.17.4.53 SSL\_VERSION Macro

**File**

SSL.h

**C**

```
#define SSL_VERSION (0x0300u) // SSL version number
```

**Description**

SSL version number

### 10.17.4.54 SSL\_VERSION\_HI Macro

**File**

SSL.h

**C**

```
#define SSL_VERSION_HI (0x03u) // SSL version number (high byte)
```

**Description**

SSL version number (high byte)

### 10.17.4.55 SSL\_VERSION\_LO Macro

**File**

SSL.h

**C**

```
#define SSL_VERSION_LO (0x00u) // SSL version number (low byte)
```

**Description**

SSL version number (low byte)

## 10.17.4.56 SSLBufferAlloc Function

**File**

SSL.c

**C**

```
static void SSLBufferAlloc(
    BYTE * id
);
```

**Side Effects**

None

**Returns**

id - Allocated buffer ID, or SSL\_INVALID\_ID (see page 377) if none available

**Description**

Allocates a buffer for use.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
id	Where to store the allocated ID

## 10.17.4.57 SSLBufferFree Function

**File**

SSL.c

**C**

```
static void SSLBufferFree(
    BYTE * id
);
```

**Side Effects**

None

**Description**

Specified buffer is released

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
id	the buffer ID to free
Outputs	id - SSL_INVALID_ID (see page 377)

## 10.17.4.58 sslBufferID Variable

**File**

SSL.c

**C**

```
BYTE sslBufferID;
```

**Description**

Which buffer is loaded

## 10.17.4.59 SSLBufferSync Function

**File**

SSL.c

**C**

```
static void SSLBufferSync(
    BYTE id
);
```

**Side Effects**

None

**Returns**

None

**Description**

Specified buffer is loaded to RAM. Only loads if necessary, and saves any current buffer before switching.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
id	the buffer ID to sync to RAM

## 10.17.4.60 SSLFinishPartialRecord Macro

**File**

SSL.h

**C**

```
#define SSLFinishPartialRecord(a) TCPSSLPutRecordHeader(a, NULL, TRUE);
```

**Description**

This is macro SSLFinishPartialRecord.

## 10.17.4.61 SSLFlushPartialRecord Macro

**File**

SSL.h

**C**

```
#define SSLFlushPartialRecord(a) TCPSSLPutRecordHeader(a, NULL, FALSE);
```

**Description**

This is macro SSLFlushPartialRecord.

## 10.17.4.62 sslHash Variable

**File**

SSL.c

**C**

```
HASH_SUM sslHash;
```

**Description**

Hash storage

## 10.17.4.63 SSLHashAlloc Function

**File**

SSL.c

**C**

```
static void SSLHashAlloc(
    BYTE * id
);
```

**Side Effects**

None

**Description**

Allocates a hash for use.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
id	Where to store the allocated ID
Outputs	id - Allocated hash ID, or SSL_INVALID_ID (see page 377) if none available

## 10.17.4.64 SSLHashFree Function

### File

SSL.c

### C

```
static void SSLHashFree(
    BYTE * id
);
```

### Side Effects

None

### Description

Specified hash is released

### Remarks

None

### Preconditions

None

### Parameters

Parameters	Description
id	the hash ID to free
Outputs	id - SSL_INVALID_ID ( <a href="#">see page 377</a> )

## 10.17.4.65 sslHashID Variable

### File

SSL.c

### C

```
BYTE sslHashID;
```

### Description

Which hash is loaded

## 10.17.4.66 SSLHashSync Function

### File

SSL.c

### C

```
static void SSLHashSync(
    BYTE id
);
```

### Side Effects

None

### Returns

None

**Description**

Specified hash is loaded to RAM. Only loads if necessary, and saves any current hash before switching.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
id	the hash ID to sync to RAM

## 10.17.4.67 sslKeys Variable

**File**

SSL.c

**C**

```
SSL_KEYS sslKeys;
```

**Description**

The current SSL session

## 10.17.4.68 sslKeysID Variable

**File**

SSL.c

**C**

```
BYTE sslKeysID;
```

**Description**

Which SSL\_KEYS (see page 405) are loaded

## 10.17.4.69 SSLKeysSync Function

**File**

SSL.c

**C**

```
static void SSLKeysSync(
    BYTE id
);
```

**Side Effects**

None

**Returns**

None

**Description**

Specified key set is loaded to RAM. Only loads if necessary, and saves any current key set before switching.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
id	the key set ID to sync to RAM

## 10.17.4.70 SSLMACAdd Function

**File**

SSL.h

**C**

```
void SSLMACAdd(  
    BYTE* data,  
    WORD len  
);
```

**Description**

This is function SSLMACAdd.

## 10.17.4.71 SSLMACBegin Function

**File**

SSL.h

**C**

```
void SSLMACBegin(  
    BYTE* MACSecret,  
    DWORD seq,  
    BYTE protocol,  
    WORD len  
);
```

**Description**

This is function SSLMACBegin.

## 10.17.4.72 SSLMACCalc Function

**File**

SSL.h

**C**

```
void SSLMACCalc(  
    BYTE* MACSecret,  
    BYTE* result  
);
```

**Description**

This is function SSLMACCalc.

### 10.17.4.73 SSLRSAOperation Function

**File**

SSL.c

**C**

```
static RSA_STATUS SSLRSAOperation();
```

**Side Effects**

None

**Returns**

None

**Description**

Pauses connection processing until RSA calculation is complete.

**Remarks**

This function exists outside of the handshaking functions so that the system does not incur the expense of resuming and suspending handshake hashes.

**Preconditions**

The RSA Module has been secured, an RSA operation is pending, sslStub.wRxHsBytesRem is the value of sslStub.wRxBytesRem after completion, and sslStub.wRxBytesRem is the value of sslStub.rxProtocol after completion. Also requires sslStub (see page 428) to be synchronized.

**Section**

Function Prototypes

\*\*\*\*\*

Cryptographic Calculation Functions

### 10.17.4.74 sslRSASStubID Variable

**File**

SSL.c

**C**

```
BYTE sslRSASStubID;
```

**Description**

Which stub is using RSA, if any

### 10.17.4.75 SSLRxAlert Function

**File**

SSL.c

**C**

```
static void SSLRxAlert(
    TCP_SOCKET hTCP
);
```

**Side Effects**

None

**Returns**

None

**Description**

Receives an alert message and decides what to do

**Remarks**

None

**Preconditions**

sslStub (see page 428) is synchronized

**Parameters**

Parameters	Description
hTCP	the TCP Socket to read from

## 10.17.4.76 SSLRxAntiqueClientHello Function

**File**

SSL.c

**C**

```
static void SSLRxAntiqueClientHello(
    TCP_SOCKET hTCP
);
```

**Side Effects**

None

**Returns**

None

**Description**

Receives the SSLv2 ClientHello message, initiating a new SSL session with a client

**Remarks**

This is the only SSLv2 message we support, and is provided for browsers seeking backwards compatibility. Connections must be upgraded to SSLv3.0 immediately following, otherwise the connection will fail.

**Preconditions**

Handshake hasher is started, and SSL has a stub assigned.

**Parameters**

Parameters	Description
hTCP	the TCP Socket to send the message to

## 10.17.4.77 SSLRxCCS Function

**File**

SSL.c

**C**

```
static void SSLRxCCS(
    TCP_SOCKET hTCP
);
```

**Side Effects**

None

**Returns**

None

**Description**

Receives a ChangeCipherSpec from the remote server

**Remarks**

None

**Preconditions**

sslStub (see page 428) is synchronized.

**Parameters**

Parameters	Description
hTCP	the TCP Socket to read from

## 10.17.4.78 SSLRxClientHello Function

**File**

SSL.c

**C**

```
static void SSLRxClientHello(
    TCP_SOCKET hTCP
);
```

**Side Effects**

None

**Returns**

None

**Description**

Receives the ClientHello message, initiating a new SSL session with a client

**Remarks**

None

**Preconditions**

Handshake hasher is started, and SSL has a stub assigned.

**Parameters**

Parameters	Description
hTCP	the TCP Socket to send the message to

## 10.17.4.79 SSLRxClientKeyExchange Function

### File

SSL.c

### C

```
static void SSLRxClientKeyExchange(
    TCP_SOCKET hTCP
);
```

### Side Effects

None

### Returns

None

### Description

Receives the ClientKeyExchange message and begins the decryption process.

### Remarks

None

### Preconditions

sslStub (see page 428) is synchronized and HSStart (see page 398)() has been called.

### Parameters

Parameters	Description
hTCP	the TCP Socket to read from

## 10.17.4.80 SSLRxFinished Function

### File

SSL.c

### C

```
static void SSLRxFinished(
    TCP_SOCKET hTCP
);
```

### Side Effects

None

### Returns

None

### Description

Receives the Finished message from remote node

### Remarks

None

### Preconditions

sslStub (see page 428) is synchronized and HSStart (see page 398)() has been called.

**Parameters**

Parameters	Description
hTCP	the TCP Socket to read from

## 10.17.4.81 SSLRxHandshake Function

**File**

SSL.h

**C**

```
void SSLRxHandshake(
    TCP_SOCKET hTCP
);
```

**Returns**

None

**Description**

This function receives handshake messages, reads the handshake header, and passes the data off to the appropriate handshake parser.

**Preconditions**

The specified SSL stub is initialized and the TCP socket is connected. Also requires that rxBytesRem has been populated and the current SSL stub has been synced into memory.

**Parameters**

Parameters	Description
hTCP	The TCP socket to read a handshake message from

## 10.17.4.82 SSLRxRecord Function

**File**

SSL.h

**C**

```
WORD SSLRxRecord(
    TCP_SOCKET hTCP,
    BYTE sslStubID
);
```

**Returns**

WORD indicating the number of data bytes there were decrypted but left in the stream.

**Description**

Reads at most one SSL Record header from the TCP stream and determines what to do with the rest of the data. If not all of the data is available for the record, then the function returns and future call(s) to SSLRxRecord() will process the remaining data until the end of the record is reached. If this call process data from a past record, the next record will not be started until the next call.

**Remarks**

SSL record headers, MAC footers, and symmetric cipher block padding (if any) will be extracted from the TCP stream by this function. Data will be decrypted but left in the stream.

**Preconditions**

The specified SSL stub is initialized and the TCP socket is connected.

**Parameters**

Parameters	Description
hTCP	The TCP socket from which to read
id	The active SSL stub ID

**10.17.4.83 SSLRxServerCertificate Function****File**

SSL.c

**C**

```
static void SSLRxServerCertificate(
    TCP_SOCKET hTCP
);
```

**Side Effects**

None

**Returns**

None

**Description**

Receives ServerCertificate from the remote server, locates the public key information, and executes RSA operation.

**Remarks**

This shortcuts full parsing of the certificate by just finding the Public Key Algorithm identifier for RSA. From there, the following ASN.1 struct is the public key. That struct consists of the value for N, followed by the value for E.

**Preconditions**

sslStub (see page 428) is synchronized and HSStart (see page 398)() has been called.

**Parameters**

Parameters	Description
hTCP	the TCP Socket to read from

**10.17.4.84 SSLRxServerHello Function****File**

SSL.c

**C**

```
static void SSLRxServerHello(
    TCP_SOCKET hTCP
);
```

**Side Effects**

None

**Returns**

None

**Description**

Receives the ServerHello from the remote server

**Remarks**

None

**Preconditions**

sslStub (see page 428) is synchronized and HSStart (see page 398)() has been called.

**Parameters**

Parameters	Description
hTCP	the TCP Socket to read from

## 10.17.4.85 sslSession Variable

**File**

SSL.c

**C**

```
SSL_SESSION sslSession;
```

**Description**

Current session data

## 10.17.4.86 sslSessionID Variable

**File**

SSL.c

**C**

```
BYTE sslSessionID;
```

**Description**

Which session is loaded

## 10.17.4.87 SSLSessionMatchID Function

**File**

SSL.c

**C**

```
static BYTE SSLSessionMatchID(
    BYTE* SessionID
);
```

**Side Effects**

None

**Returns**

The matched session ID, or SSL\_INVALID\_ID (see page 377) if not found

**Description**

Locates a cached SSL session for reuse. Syncs found session into RAM.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
SessionID	the session identifier to match

**Section**

Server messages

## 10.17.4.88 SSLSessionMatchIP Function

**File**

SSL.c

**C**

```
static BYTE SSLSessionMatchIP(
    IP_ADDR ip
);
```

**Side Effects**

None

**Returns**

The matched session ID, or SSL\_INVALID\_ID (see page 377) if not found

**Description**

Locates a cached SSL session for reuse

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
ip	the host session to match

**Section**

Client messages

## 10.17.4.89 SSLSessionNew Function

**File**

SSL.c

**C**

```
static BYTE SSLSessionNew();
```

**Side Effects**

None

**Returns**

Allocated Session ID, or SSL\_INVALID\_ID (see page 377) if none available

**Description**

Finds space for a new SSL session

**Remarks**

None

**Preconditions**

None

## 10.17.4.90 sslSessionStubs Variable

**File**

SSL.c

**C**

```
SSL_SESSION_STUB sslSessionStubs[MAX_SSL_SESSIONS];
```

**Description**

8 byte session stubs

## 10.17.4.91 SSLSessionSync Function

**File**

SSL.c

**C**

```
static void SSLSessionSync(
    BYTE id
);
```

**Side Effects**

None

**Returns**

None

**Description**

Specified session is loaded to RAM. Only loads if necessary, and saves any current session before switching if it has been updated.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
id	the session ID to sync to RAM

### 10.17.4.92 SSLSessionUpdated Macro

**File**

SSL.c

**C**

```
#define SSLSessionUpdated sslSessionUpdated = TRUE;
```

**Description**

This is macro SSLSessionUpdated.

### 10.17.4.93 sslSessionUpdated Variable

**File**

SSL.c

**C**

```
BOOL sslSessionUpdated;
```

**Description**

Whether or not it has been updated

### 10.17.4.94 SSLStartPartialRecord Function

**File**

SSL.h

**C**

```
void SSLStartPartialRecord(
    TCP_SOCKET hTCP,
    BYTE sslStubID,
    BYTE txProtocol,
    WORD wLen
);
```

**Returns**

None

**Description**

This function allows messages longer than the TCP buffer to be sent, which is frequently the case for the Certificate handshake message. The final message length is required to be known in order to transmit the header. Once called, SSLFlushPartialRecord (see page 414) and SSLFinishPartialRecord (see page 413) must be called to write remaining data, finalize, and prepare for a new record.

**Remarks**

Partial messages do not support the current cipher spec, so this can only be used during the handshake procedure.

**Preconditions**

The specified SSL stub is initialized and the TCP socket is connected.

**Parameters**

Parameters	Description
hTCP	The TCP socket with data waiting to be transmitted
id	The active SSL stub ID

txPortocol	The SSL protocol number to attach to this record
wLen	The length of all the data to be sent

### 10.17.4.95 sslStub Variable

#### File

SSL.c

#### C

```
SSL_STUB sslStub;
```

#### Description

The current SSL stub

### 10.17.4.96 SSLStubAlloc Function

#### File

SSL.c

#### C

```
static BOOL SSLStubAlloc();
```

#### Side Effects

None

#### Returns

TRUE if stub was allocated, FALSE otherwise

#### Description

Allocates a stub for use.

#### Remarks

None

#### Preconditions

None

#### Parameters

Parameters	Description
Outputs	None

### 10.17.4.97 SSLStubFree Function

#### File

SSL.c

#### C

```
static void SSLStubFree(
    BYTE id
);
```

#### Side Effects

None

**Returns**

None

**Description**

Specified stub is released

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
id	the stub ID to free
Outputs	None

## 10.17.4.98 sslStubID Variable

**File**

SSL.c

**C**

```
BYTE sslStubID;
```

**Description**

Which SSL\_STUB (see page 409) is loaded

## 10.17.4.99 SSLStubSync Function

**File**

SSL.c

**C**

```
static void SSLStubSync(
    BYTE id
);
```

**Side Effects**

None

**Returns**

None

**Description**

Specified stub is loaded to RAM. Only loads if necessary, and saves any current stub before switching.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
id	the stub ID to sync to RAM

**Section**

Ethernet Buffer RAM Management

## 10.17.4.100 SSLTerminate Function

**File**

SSL.h

**C**

```
void SSLTerminate(
    BYTE sslStubId
);
```

**Returns**

None

**Description**

Terminates an SSL connection and releases allocated resources.

**Preconditions**

None

**Parameters**

Parameters	Description
id	the SSL stub ID to terminate

## 10.17.4.101 SSLTxCCSFin Function

**File**

SSL.c

**C**

```
static void SSLTxCCSFin(
    TCP_SOCKET hTCP
);
```

**Side Effects**

None

**Returns**

None

**Description**

Generates the session keys from the master secret, then allocates and generates the encryption context. Once processing is complete, transmits the Change Cipher Spec message and the Finished handshake message to the server.

**Remarks**

None

**Preconditions**

sslStub (see page 428) is synchronized, and the current session has a valid pre-master secret to use.

**Parameters**

Parameters	Description
hTCP	the TCP Socket to write the message to

**Section**

Client and server messages

## 10.17.4.102 SSLTxClientHello Function

**File**

SSL.c

**C**

```
static void SSLTxClientHello(
    TCP_SOCKET hTCP
);
```

**Side Effects**

None

**Returns**

None

**Description**

Transmits the ClientHello message to initiate a new SSL session with the server.

**Remarks**

None

**Preconditions**

Enough space is available in hTCP to write the entire message.

**Parameters**

Parameters	Description
hTCP	the TCP Socket to send the message to

## 10.17.4.103 SSLTxClientKeyExchange Function

**File**

SSL.c

**C**

```
static void SSLTxClientKeyExchange(
    TCP_SOCKET hTCP
);
```

**Side Effects**

None

**Returns**

None

**Description**

Transmits the encrypted pre-master secret to the server and requests the Change Cipher Spec. Also generates the Master Secret from the pre-master secret that was used.

**Remarks**

None

**Preconditions**

sslStub (see page 428) is synchronized, sslStub.dwTemp.v[1] contains the length of the public key, and the RxBuffer contains the encrypted pre-master secret at address 0x80.

**Parameters**

Parameters	Description
hTCP	the TCP Socket to write the message to

## 10.17.4.104 SSLTxMessage Function

**File**

SSL.h

**C**

```
void SSLTxMessage(
    TCP_SOCKET hTCP,
    BYTE sslStubID,
    BYTE msg
);
```

**Returns**

None

**Description**

This function transmits a specific SSL message for handshakes and alert messages. Supported messages are listed in SSL\_MESSAGES (see page 406).

**Preconditions**

The specified SSL stub is initialized and the TCP socket is connected.

**Parameters**

Parameters	Description
hTCP	The TCP socket with data waiting to be transmitted
id	The active SSL stub ID
msg	One of the SSL_MESSAGES (see page 406) types to send

## 10.17.4.105 SSLTxRecord Function

**File**

SSL.h

**C**

```
void SSLTxRecord(
    TCP_SOCKET hTCP,
    BYTE sslStubID,
    BYTE txProtocol
);
```

**Returns**

None

**Description**

Transmits all pending data in the TCP TX buffer as an SSL record using the specified protocol. This function transparently

encrypts and MACs the data if there is an active cipher spec.

#### Preconditions

The specified SSL stub is initialized and the TCP socket is connected.

#### Parameters

Parameters	Description
hTCP	The TCP socket with data waiting to be transmitted
id	The active SSL stub ID
txPortocol	The SSL protocol number to attach to this record

## 10.17.4.106 SSLTxServerCertificate Function

#### File

SSL.c

#### C

```
static void SSLTxServerCertificate(
    TCP_SOCKET hTCP
);
```

#### Side Effects

None

#### Returns

None

#### Description

Transmits the Certificate message with the server's specified public key certificate.

#### Remarks

Certificate is defined in CustomSSLCert.c. This function requires special handling for partial records because the certificate will likely be larger than the TCP buffer, and SSL handshake messages are constrained to fit in a single SSL handshake record

#### Preconditions

None

#### Parameters

Parameters	Description
hTCP	the TCP Socket to send the message to

## 10.17.4.107 SSLTxServerHello Function

#### File

SSL.c

#### C

```
static void SSLTxServerHello(
    TCP_SOCKET hTCP
);
```

#### Side Effects

None

**Returns**

None

**Description**

Transmits the ServerHello message.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
hTCP	the TCP Socket to send the message to

## 10.17.4.108 SSLTxServerHelloDone Function

**File**

SSL.c

**C**

```
static void SSLTxServerHelloDone(
    TCP_SOCKET hTCP
);
```

**Side Effects**

None

**Returns**

None

**Description**

Transmits the ServerHelloDone message.

**Remarks**

None

**Preconditions**

None

**Parameters**

Parameters	Description
hTCP	the TCP Socket to send the message to

## 10.18 TCP

**Variables**

	Name	Description
	NextPort ( <a href="#">🔗</a> see page 480)	Tracking variable for next local client port number

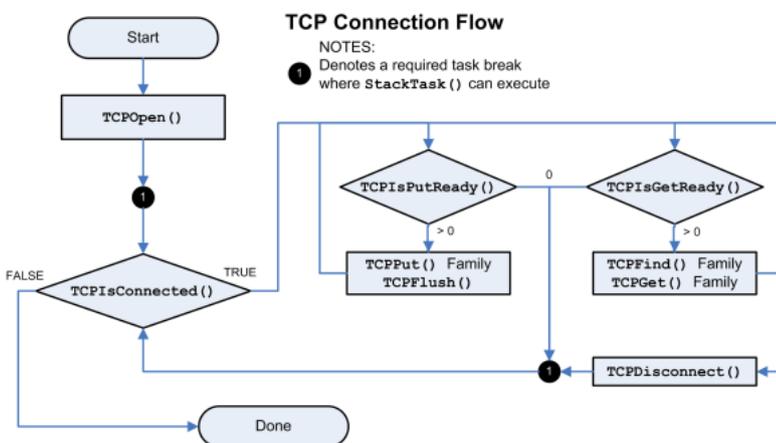
**Description**

TCP is a standard transport layer protocol described in RFC 793. It provides reliable stream-based connections over unreliable networks, and forms the foundation for HTTP, SMTP, and many other protocol standards.

Connections made over TCP guarantee data transfer at the expense of throughput. Connections are made through a three-way handshake process, ensuring a one-to-one connection. Remote nodes advertise how much data they are ready to receive, and all data transmitted must be acknowledged. If a remote node fails to acknowledge the receipt of data, it is automatically retransmitted. This ensures that network errors such as lost, corrupted, or out-of-order packets are automatically corrected.

To accomplish this, TCP must operate in a buffer. Once the transmit buffer is full, no more data can be sent until the remote node has acknowledged receipt. For the Microchip TCP/IP Stack, the application must return to the main stack loop in order for this to happen. Likewise, the remote node cannot transmit more data until the local device has acknowledged receipt and that space is available in the buffer. When a local application needs to read more data, it must return to the main stack loop and wait for a new packet to arrive.

The TCP flow diagram below provides an overview for the use of the TCP module:



Sockets (see page 146) are opened using TCPOpen (see page 451). This function can either open a listening socket to wait for client connections, or can make a client connection to the remote node. The remote node can be specified by a host name string to be resolved in DNS, an IP address, or a NODE\_INFO struct containing previously resolved IP and MAC address information.

Once connected, applications can read and write data. On each entry, the application must verify that the socket is still connected. For most applications a call to TCPIsConnected (see page 449) will be sufficient, but TCPWasReset (see page 458) may also be used for listening sockets that may turn over quickly.

To write data, call TCPIsPutReady (see page 450) to check how much space is available. Then, call any of the TCPPut (see page 454) family of functions to write data as space is available. Once complete, call TCPFlush (see page 446) to transmit data immediately. Alternately, return to the main stack loop. Data will be transmitted when either a) half of the transmit buffer becomes full or b) a delay time has passed (usually 40ms).

To read data, call TCPIsGetReady (see page 450) to determine how many bytes are ready to be retrieved. Then use the TCPGet (see page 446) family of functions to read data from the socket, and/or the TCPFind (see page 443) family of functions to locate data in the buffer. When no more data remains, return to the main stack loop to wait for more data to arrive.

If the application needs to close the connection, call TCPDisconnect (see page 442), then return to the main stack loop and wait for the remote node to acknowledge the disconnection. Client sockets will return to the idle state, while listening sockets will wait for a new connection.

For more information, refer to the GenericTCPClient (see page 93) or GenericTCPServer (see page 96) examples, or read the associated RFC.

## 10.18.1 TCP Public Members

### Functions

	Name	Description
◆	TCPAdjustFIFOSize (see page 440)	Adjusts the relative sizes of the RX and TX buffers.
◆	TCPClose (see page 441)	Disconnects an open socket and destroys the socket handle, including server mode socket handles.
◆	TCPDiscard (see page 442)	Discards any pending data in the TCP RX FIFO.
◆	TCPDisconnect (see page 442)	Disconnects an open socket.
◆	TCPFindArrayEx (see page 443)	Searches for a string in the TCP RX buffer.
◆	TCPFindEx (see page 444)	Searches for a byte in the TCP RX buffer.
◆	TCPFindROMArrayEx (see page 445)	Searches for a ROM string in the TCP RX buffer.
◆	TCPFlush (see page 446)	Immediately transmits all pending TX data.
◆	TCPGet (see page 446)	Retrieves a single byte to a TCP socket.
◆	TCPGetArray (see page 447)	Reads an array of data bytes from a TCP socket's receive FIFO. The data is removed from the FIFO in the process.
◆	TCPGetRemoteInfo (see page 447)	Obtains information about a currently open socket.
◆	TCPGetRxFIFOFree (see page 448)	Determines how many bytes are free in the RX FIFO.
◆	TCPGetTxFIFOFull (see page 449)	Determines how many bytes are pending in the TCP TX FIFO.
◆	TCPIsConnected (see page 449)	Determines if a socket has an established connection.
◆	TCPIsGetReady (see page 450)	Determines how many bytes can be read from the TCP RX buffer.
◆	TCPIsPutReady (see page 450)	Determines how much free space is available in the TCP TX buffer.
◆	TCPOpen (see page 451)	Opens a TCP socket for listening or as a client.
◆	TCPPeek (see page 453)	Peeks at one byte in the TCP RX FIFO without removing it from the buffer.
◆	TCPPeekArray (see page 453)	Reads a specified number of data bytes from the TCP RX FIFO without removing them from the buffer.
◆	TCPPut (see page 454)	Writes a single byte to a TCP socket.
◆	TCPPutArray (see page 454)	Writes an array from RAM to a TCP socket.
◆	TCPPutROMArray (see page 455)	Writes an array from ROM to a TCP socket.
◆	TCPPutROMString (see page 455)	Writes a null-terminated string from ROM to a TCP socket. The null-terminator is not copied to the socket.
◆	TCPPutString (see page 456)	Writes a null-terminated string from RAM to a TCP socket. The null-terminator is not copied to the socket.
◆	TCPRAMCopy (see page 456)	Copies data to/from various memory mediums.
◆	TCPRAMCopyROM (see page 457)	Copies data to/from various memory mediums.
◆	TCPWasReset (see page 458)	Self-clearing semaphore indicating socket reset.

**Macros**

	<b>Name</b>	<b>Description</b>
↔	INVALID_SOCKET (↗ see page 437)	The socket is invalid or could not be opened
↔	UNKNOWN_SOCKET (↗ see page 438)	The socket is not known
↔	TCP_ADJUST_GIVE_REST_TO_RX (↗ see page 438)	Resize flag: extra bytes go to RX
↔	TCP_ADJUST_GIVE_REST_TO_TX (↗ see page 438)	Resize flag: extra bytes go to TX
↔	TCP_ADJUST_PRESERVE_RX (↗ see page 438)	Resize flag: attempt to preserve RX buffer
↔	TCP_ADJUST_PRESERVE_TX (↗ see page 439)	Resize flag: attempt to preserve TX buffer
↔	TCP_OPEN_IP_ADDRESS (↗ see page 439)	Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_IP_ADDRESS while STACK_CLIENT_MODE feature is not enabled.
↔	TCP_OPEN_NODE_INFO (↗ see page 439)	Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_NODE_INFO while STACK_CLIENT_MODE feature is not enabled.
↔	TCP_OPEN_RAM_HOST (↗ see page 439)	Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_RAM_HOST while STACK_CLIENT_MODE feature is not enabled.
↔	TCP_OPEN_ROM_HOST (↗ see page 440)	Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_ROM_HOST while STACK_CLIENT_MODE feature is not enabled.
↔	TCP_OPEN_SERVER (↗ see page 440)	Create a server socket and ignore dwRemoteHost.
↔	TCPCConnect (↗ see page 441)	Alias to TCPOpen (↗ see page 451) as a client.
↔	TCPCFind (↗ see page 443)	Alias to TCPCFindEx (↗ see page 444) with no length parameter.
↔	TCPCFindArray (↗ see page 443)	Alias to TCPCFindArrayEx (↗ see page 443) with no length parameter.
↔	TCPCFindROMArray (↗ see page 445)	Alias to TCPCFindROMArrayEx (↗ see page 445) with no length parameter.
↔	TCPGetRxFIFOFull (↗ see page 448)	Alias to TCPCIsGetReady (↗ see page 450) provided for API completeness
↔	TCPGetTxFIFOFree (↗ see page 449)	Alias to TCPCIsPutReady (↗ see page 450) provided for API completeness
↔	TCPListen (↗ see page 451)	Alias to TCPOpen (↗ see page 451) as a server.

**Module**

TCP (↗ see page 434)

**Description**

The following functions and variables are available to the stack application.

### 10.18.1.1 INVALID\_SOCKET Macro

**File**

TCP.h

**C**

```
#define INVALID_SOCKET (0xFE) // The socket is invalid or could not be opened
```

**Description**

The socket is invalid or could not be opened

## 10.18.1.2 UNKNOWN\_SOCKET Macro

**File**

TCP.h

**C**

```
#define UNKNOWN_SOCKET (0xFF) // The socket is not known
```

**Description**

The socket is not known

## 10.18.1.3 TCP\_ADJUST\_GIVE\_REST\_TO\_RX Macro

**File**

TCP.h

**C**

```
#define TCP_ADJUST_GIVE_REST_TO_RX 0x01u // Resize flag: extra bytes go to RX
```

**Description**

Resize flag: extra bytes go to RX

## 10.18.1.4 TCP\_ADJUST\_GIVE\_REST\_TO\_TX Macro

**File**

TCP.h

**C**

```
#define TCP_ADJUST_GIVE_REST_TO_TX 0x02u // Resize flag: extra bytes go to TX
```

**Description**

Resize flag: extra bytes go to TX

## 10.18.1.5 TCP\_ADJUST\_PRESERVE\_RX Macro

**File**

TCP.h

**C**

```
#define TCP_ADJUST_PRESERVE_RX 0x04u // Resize flag: attempt to preserve RX buffer
```

**Description**

Resize flag: attempt to preserve RX buffer

## 10.18.1.6 TCP\_ADJUST\_PRESERVE\_TX Macro

### File

TCP.h

### C

```
#define TCP_ADJUST_PRESERVE_TX 0x08u // Resize flag: attempt to preserve TX buffer
```

### Description

Resize flag: attempt to preserve TX buffer

## 10.18.1.7 TCP\_OPEN\_IP\_ADDRESS Macro

### File

TCP.h

### C

```
#define TCP_OPEN_IP_ADDRESS You_need_to_enable_STACK_CLIENT_MODE_to_use_TCP_OPEN_IP_ADDRESS
```

### Description

Emit an undeclared identifier diagnostic if code tries to use TCP\_OPEN\_IP\_ADDRESS while STACK\_CLIENT\_MODE feature is not enabled.

## 10.18.1.8 TCP\_OPEN\_NODE\_INFO Macro

### File

TCP.h

### C

```
#define TCP_OPEN_NODE_INFO You_need_to_enable_STACK_CLIENT_MODE_to_use_TCP_OPEN_NODE_INFO
```

### Description

Emit an undeclared identifier diagnostic if code tries to use TCP\_OPEN\_NODE\_INFO while STACK\_CLIENT\_MODE feature is not enabled.

## 10.18.1.9 TCP\_OPEN\_RAM\_HOST Macro

### File

TCP.h

### C

```
#define TCP_OPEN_RAM_HOST You_need_to_enable_STACK_CLIENT_MODE_to_use_TCP_OPEN_RAM_HOST
```

### Description

Emit an undeclared identifier diagnostic if code tries to use TCP\_OPEN\_RAM\_HOST while STACK\_CLIENT\_MODE feature is not enabled.

## 10.18.1.10 TCP\_OPEN\_ROM\_HOST Macro

### File

TCP.h

### C

```
#define TCP_OPEN_ROM_HOST You_need_to_enable_STACK_CLIENT_MODE_to_use_TCP_OPEN_ROM_HOST
```

### Description

Emit an undeclared identifier diagnostic if code tries to use TCP\_OPEN\_ROM\_HOST while STACK\_CLIENT\_MODE feature is not enabled.

## 10.18.1.11 TCP\_OPEN\_SERVER Macro

### File

TCP.h

### C

```
#define TCP_OPEN_SERVER 0u
```

### Description

Create a server socket and ignore dwRemoteHost.

## 10.18.1.12 TCPAdjustFIFOSize Function

### File

TCP.h

### C

```
BOOL TCPAdjustFIFOSize(  
    TCP_SOCKET hTCP,  
    WORD wMinRXSize,  
    WORD wMinTXSize,  
    BYTE vFlags  
);
```

### Side Effects

Any unacknowledged or untransmitted data in the TX FIFO is always deleted.

### Description

This function can be used to adjust the relative sizes of the RX and TX FIFO depending on the immediate needs of an application. Since a larger FIFO can allow more data to be sent in a given packet, adjusting the relative sizes on the fly can allow for optimal transmission speed for one-sided application protocols. For example, HTTP typically begins by receiving large amounts of data from the client, then switches to serving large amounts of data back. Adjusting the FIFO at these points can increase performance substantially. Once the FIFO is adjusted, a window update is sent.

If neither or both of TCP\_ADJUST\_GIVE\_REST\_TO\_TX (see page 438) and TCP\_ADJUST\_GIVE\_REST\_TO\_RX (see page 438) are set, the function distributes the remaining space equally.

Received data can be preserved as long as the buffer is expanding and has not wrapped.

### Remarks

At least one byte must always be allocated to the RX buffer so that a FIN can be received. The function automatically corrects for this.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to be adjusted
wMinRXSize	Minimum number of byte for the RX FIFO
wMinTXSize	Minimum number of bytes for the RX FIFO
vFlags	Any combination of TCP_ADJUST_GIVE_REST_TO_RX (see page 438), TCP_ADJUST_GIVE_REST_TO_TX (see page 438), TCP_ADJUST_PRESERVE_RX (see page 438), TCP_ADJUST_PRESERVE_TX (see page 439) is not currently supported.

**Return Values**

Return Values	Description
TRUE	The FIFOs were adjusted successfully
FALSE	Minimum RX, Minimum TX, or flags couldn't be accommodated and therefore the socket was left unchanged.

## 10.18.1.13 TCPConnect Macro

**File**

TCP.h

**C**

```
#define TCPConnect(remote,port) TCPOpen((DWORD)remote, TCP_OPEN_NODE_INFO, port, TCP_PURPOSE_DEFAULT)
```

**Description**

This function is an alias to TCPOpen (see page 451) for client sockets. It is provided for backwards compatibility with older versions of the stack. New applications should use the TCPOpen (see page 451) API instead.

## 10.18.1.14 TCPClose Function

**File**

TCP.h

**C**

```
void TCPClose(
    TCP_SOCKET hTCP
);
```

**Returns**

None

**Description**

Disconnects an open socket and destroys the socket handle, including server mode socket handles. This function performs identically to the TCPDisconnect (see page 442)() function, except that both client and server mode socket handles are relinquished to the TCP/IP stack upon return.

**Preconditions**

None

**Parameters**

Parameters	Description
hTCP	Handle to the socket to disconnect and close.

## 10.18.1.15 TCPDiscard Function

**File**

TCP.h

**C**

```
void TCPDiscard(
    TCP_SOCKET hTCP
);
```

**Returns**

None

**Description**

Discards any pending data in the TCP RX FIFO.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket whose RX FIFO is to be cleared.

## 10.18.1.16 TCPDisconnect Function

**File**

TCP.h

**C**

```
void TCPDisconnect(
    TCP_SOCKET hTCP
);
```

**Returns**

None

**Description**

This function closes a connection to a remote node by sending a FIN (if currently connected).

The function can be called a second time to force a socket closed by sending a RST packet. This is useful when the application knows that the remote node will not send an ACK (if it has crashed or lost its link), or when the application needs to reuse the socket immediately regardless of whether or not the remote node would like to transmit more data before closing.

For client mode sockets, upon return, the hTCP handle is relinquished to the TCP/IP stack and must no longer be used by the application (except for an immediate subsequent call to TCPDisconnect() to force a RST transmission, if needed).

For server mode sockets, upon return, the hTCP handle is NOT relinquished to the TCP/IP stack. After closing, the socket returns to the listening state allowing future connection requests to be serviced. This leaves the hTCP handle in a valid state and must be retained for future operations on the socket. If you want to close the server and relinquish the socket back to the TCP/IP stack, call the TCPClose (see page 441)() API instead of TCPDisconnect().

**Remarks**

If the socket is using SSL, a CLOSE\_NOTIFY record will be transmitted first to allow the SSL session to be resumed at a later time.

**Preconditions**

None

**Parameters**

Parameters	Description
hTCP	Handle of the socket to disconnect.

## 10.18.1.17 TCPFind Macro

**File**

TCP.h

**C**

```
#define TCPFind(a,b,c,d) TCPFindEx(a,b,c,0,d)
```

**Description**

This function is an alias to TCPFindEx (see page 444) with no length parameter. It is provided for backwards compatibility with an older API.

## 10.18.1.18 TCPFindArray Macro

**File**

TCP.h

**C**

```
#define TCPFindArray(a,b,c,d,e) TCPFindArrayEx(a,b,c,d,0,e)
```

**Description**

This function is an alias to TCPFindArrayEx (see page 443) with no length parameter. It is provided for backwards compatibility with an older API.

## 10.18.1.19 TCPFindArrayEx Function

**File**

TCP.h

**C**

```
WORD TCPFindArrayEx(
    TCP_SOCKET hTCP,
    BYTE* cFindArray,
    WORD wLen,
    WORD wStart,
    WORD wSearchLen,
    BOOL bTextCompare
);
```

**Description**

This function finds the first occurrence of an array of bytes in the TCP RX buffer. It can be used by an application to abstract searches out of their own application code. For increased efficiency, the function is capable of limiting the scope of search to a specific range of bytes. It can also perform a case-insensitive search if required.

For example, if the buffer contains "I love PIC MCUs!" and the search array is "love" with a length of 4, a value of 2 will be returned.

**Remarks**

Since this function usually must transfer data from external storage to internal RAM for comparison, its performance degrades significantly when the buffer is full and the array is not found. For better performance, try to search for characters that are expected to exist or limit the scope of the search as much as possible. The HTTP2 module, for example, uses this function to parse headers. However, it searches for newlines, then the separating colon, then reads the header name to RAM for final comparison. This has proven to be significantly faster than searching for full header name strings outright.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to search within.
cFindArray	The array of bytes to find in the buffer.
wLen	Length of cFindArray.
wStart	Zero-indexed starting position within the buffer.
wSearchLen	Length from wStart to search in the buffer.
bTextCompare	TRUE for case-insensitive text search, FALSE for binary search

**Return Values**

Return Values	Description
0xFFFF	Search array not found
Otherwise	Zero-indexed position of the first occurrence

## 10.18.1.20 TCPFindEx Function

**File**

TCP.h

**C**

```
WORD TCPFindEx(
    TCP_SOCKET hTCP,
    BYTE cFind,
    WORD wStart,
    WORD wSearchLen,
    BOOL bTextCompare
);
```

**Description**

This function finds the first occurrence of a byte in the TCP RX buffer. It can be used by an application to abstract searches out of their own application code. For increased efficiency, the function is capable of limiting the scope of search to a specific range of bytes. It can also perform a case-insensitive search if required.

For example, if the buffer contains "I love PIC MCUs!" and the cFind byte is ' ', a value of 1 will be returned.

**Remarks**

Since this function usually must transfer data from external storage to internal RAM for comparison, its performance degrades significantly when the buffer is full and the array is not found. For better performance, try to search for characters that are expected to exist or limit the scope of the search as much as possible. The HTTP2 module, for example, uses this function to parse headers. However, it searches for newlines, then the separating colon, then reads the header name to RAM for final comparison. This has proven to be significantly faster than searching for full header name strings outright.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to search within.
cFind	The byte to find in the buffer.
wStart	Zero-indexed starting position within the buffer.
wSearchLen	Length from wStart to search in the buffer.
bTextCompare	TRUE for case-insensitive text search, FALSE for binary search

**Return Values**

Return Values	Description
0xFFFF	Search array not found
Otherwise	Zero-indexed position of the first occurrence

## 10.18.1.21 TCPFindROMArray Macro

**File**

TCP.h

**C**

```
#define TCPFindROMArray(a,b,c,d,e) TCPFindArray(a,(BYTE*)b,c,d,e)
```

**Description**

This function is an alias to TCPFindROMArrayEx (see page 445) with no length parameter. It is provided for backwards compatibility with an older API.

## 10.18.1.22 TCPFindROMArrayEx Function

**File**

TCP.h

**C**

```
WORD TCPFindROMArrayEx(
    TCP_SOCKET hTCP,
    ROM BYTE* cFindArray,
    WORD wLen,
    WORD wStart,
    WORD wSearchLen,
    BOOL bTextCompare
);
```

**Description**

This function finds the first occurrence of an array of bytes in the TCP RX buffer. It can be used by an application to abstract searches out of their own application code. For increased efficiency, the function is capable of limiting the scope of search to a specific range of bytes. It can also perform a case-insensitive search if required.

For example, if the buffer contains "I love PIC MCUs!" and the search array is "love" with a length of 4, a value of 2 will be returned.

**Remarks**

Since this function usually must transfer data from external storage to internal RAM for comparison, its performance degrades significantly when the buffer is full and the array is not found. For better performance, try to search for characters that are expected to exist or limit the scope of the search as much as possible. The HTTP2 module, for example, uses this

function to parse headers. However, it searches for newlines, then the separating colon, then reads the header name to RAM for final comparison. This has proven to be significantly faster than searching for full header name strings outright.

This function is aliased to TCPFindArrayEx (see page 443) on non-PIC18 platforms.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to search within.
cFindArray	The array of bytes to find in the buffer.
wLen	Length of cFindArray.
wStart	Zero-indexed starting position within the buffer.
wSearchLen	Length from wStart to search in the buffer.
bTextCompare	TRUE for case-insensitive text search, FALSE for binary search

**Return Values**

Return Values	Description
0xFFFF	Search array not found
Otherwise	Zero-indexed position of the first occurrence

### 10.18.1.23 TCPFlush Function

**File**

TCP.h

**C**

```
void TCPFlush(
    TCP_SOCKET hTCP
);
```

**Returns**

None

**Description**

This function immediately transmits all pending TX data with a PSH flag. If this function is not called, data will automatically be sent when either a) the TX buffer is half full or b) the TCP\_AUTO\_TRANSMIT\_TIMEOUT\_VAL (see page 474) (default: 40ms) has elapsed.

**Remarks**

SSL application data is automatically flushed, so this function has no effect for SSL sockets.

**Preconditions**

TCP is initialized and the socket is connected.

**Parameters**

Parameters	Description
hTCP	The socket whose data is to be transmitted.

### 10.18.1.24 TCPGet Function

**File**

TCP.h

**C**

```

BOOL TCPGet(
    TCP_SOCKET hTCP,
    BYTE* byte
);

```

**Description**

Retrieves a single byte to a TCP socket.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket from which to read.
byte	Pointer to location in which the read byte should be stored.

**Return Values**

Return Values	Description
TRUE	A byte was read from the buffer.
FALSE	The buffer was empty, or the socket is not connected.

## 10.18.1.25 TCPGetArray Function

**File**

TCP.h

**C**

```

WORD TCPGetArray(
    TCP_SOCKET hTCP,
    BYTE* buffer,
    WORD count
);

```

**Returns**

The number of bytes read from the socket. If less than len, the RX FIFO buffer became empty or the socket is not connected.

**Description**

Reads an array of data bytes from a TCP socket's receive FIFO. The data is removed from the FIFO in the process.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket from which data is to be read.
buffer	Pointer to the array to store data that was read.
len	Number of bytes to be read.

## 10.18.1.26 TCPGetRemoteInfo Function

**File**

TCP.h

**C**

```
SOCKET_INFO* TCPGetRemoteInfo(
    TCP_SOCKET hTCP
);
```

**Returns**

The SOCKET\_INFO (see page 459) structure associated with this socket. This structure is allocated statically by the function and is valid only until the next time TCPGetRemoteInfo() is called.

**Description**

Returns the SOCKET\_INFO (see page 459) structure associated with this socket. This contains the NODE\_INFO structure with IP and MAC address (or gateway MAC) and the remote port.

**Preconditions**

TCP is initialized and the socket is connected.

**Parameters**

Parameters	Description
hTCP	The socket to check.

## 10.18.1.27 TCPGetRxFIFOFree Function

**File**

TCP.h

**C**

```
WORD TCPGetRxFIFOFree(
    TCP_SOCKET hTCP
);
```

**Returns**

The number of bytes free in the TCP RX FIFO. If zero, no additional data can be received until the application removes some data using one of the TCPGet (see page 446) family functions.

**Description**

Determines how many bytes are free in the RX FIFO.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to check.

## 10.18.1.28 TCPGetRxFIFOFull Macro

**File**

TCP.h

**C**

```
#define TCPGetRxFIFOFull(a) TCPIsGetReady(a)
```

**Description**

Alias to TCPIsGetReady (see page 450) provided for API completeness

## 10.18.1.29 TCPGetTxFIFOFree Macro

### File

TCP.h

### C

```
#define TCPGetTxFIFOFree(a) TCPIsPutReady(a)
```

### Description

Alias to TCPIsPutReady (see page 450) provided for API completeness

## 10.18.1.30 TCPGetTxFIFOFull Function

### File

TCP.h

### C

```
WORD TCPGetTxFIFOFull(  
    TCP_SOCKET hTCP  
);
```

### Returns

Number of bytes pending to be flushed in the TCP TX FIFO.

### Description

Determines how many bytes are pending in the TCP TX FIFO.

### Preconditions

TCP is initialized.

### Parameters

Parameters	Description
hTCP	The socket to check.

## 10.18.1.31 TCPIsConnected Function

### File

TCP.h

### C

```
BOOL TCPIsConnected(  
    TCP_SOCKET hTCP  
);
```

### Description

This function determines if a socket has an established connection to a remote node. Call this function after calling TCPOpen (see page 451) to determine when the connection is set up and ready for use. This function was historically used to check for disconnections, but TCPWasReset (see page 458) is now a more appropriate solution.

### Remarks

A socket is said to be connected only if it is in the TCP\_ESTABLISHED state. Sockets (see page 146) in the process of opening or closing will return FALSE.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to check.

**Return Values**

Return Values	Description
TRUE	The socket has an established connection to a remote node.
FALSE	The socket is not currently connected.

## 10.18.1.32 TCPIsGetReady Function

**File**

TCP.h

**C**

```
WORD TCPIsGetReady(
    TCP_SOCKET hTCP
);
```

**Returns**

The number of bytes available to be read from the TCP RX buffer.

**Description**

Call this function to determine how many bytes can be read from the TCP RX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to wait for more data to arrive.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to check.

## 10.18.1.33 TCPIsPutReady Function

**File**

TCP.h

**C**

```
WORD TCPIsPutReady(
    TCP_SOCKET hTCP
);
```

**Returns**

The number of bytes available to be written in the TCP TX buffer.

**Description**

Call this function to determine how many bytes can be written to the TCP TX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to transmit more data.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to check.

## 10.18.1.34 TCPListen Macro

**File**

TCP.h

**C**

```
#define TCPListen(port) TCPOpen(0, TCP_OPEN_SERVER, port, TCP_PURPOSE_DEFAULT)
```

**Description**

This function is an alias to TCPOpen (see page 451) for server sockets. It is provided for backwards compatibility with older versions of the stack. New applications should use the TCPOpen (see page 451) API instead.

## 10.18.1.35 TCPOpen Function

**File**

TCP.h

**C**

```
TCP_SOCKET TCPOpen(
    DWORD dwRemoteHost,
    BYTE vRemoteHostType,
    WORD wPort,
    BYTE vSocketPurpose
);
```

**Description**

Provides a unified method for opening TCP sockets. This function can open both client and server sockets. For client sockets, it can accept (see page 163) a host name string to query in DNS, an IP address as a string, an IP address in binary form, or a previously resolved NODE\_INFO structure containing the remote IP address and associated MAC address. When a host name or IP address only is provided, the TCP module will internally perform the necessary DNS and/or ARP resolution steps before reporting that the TCP socket is connected (via a call to TCPISConnected returning TRUE). Server sockets ignore this destination parameter and listen (see page 169) only on the indicated port.

The vSocketPurpose field allows sockets to be opened with varying buffer size parameters and memory storage mediums. This field corresponds to pre-defined sockets allocated in the TCPSocketInitializer[] array in TCPIPConfig.h. The TCPIPConfig.h file can be edited using the TCP/IP Configuration Wizard.

Sockets (see page 146) are statically allocated on boot, but can be claimed with this function and freed using TCPDisconnect (see page 442) or TCPClose (see page 441) (for client sockets). Server sockets can be freed using TCPClose (see page 441) only (calls to TCPDisconnect (see page 442) will return server sockets to the listening state, allowing reuse).

**Remarks**

This function replaces the old TCPConnect (see page 441) and TCPListen (see page 451) functions.

If TCP\_OPEN\_RAM\_HOST (see page 439) or TCP\_OPEN\_ROM\_HOST (see page 440) are used for the destination type, the DNS client module must also be enabled (STACK\_USE\_DNS must be defined in TCPIPConfig.h).

**Preconditions**

TCP is initialized.

**Example**

```
// Open a server socket
skt = TCPOpen(NULL, TCP_OPEN_SERVER, HTTP_PORT, TCP_PURPOSE_HTTP_SERVER);

// Open a client socket to www.microchip.com
// The double cast here prevents compiler warnings
skt = TCPOpen((DWORD)(PTR_BASE)"www.microchip.com",
              TCP_OPEN_ROM_HOST, 80, TCP_PURPOSE_DEFAULT);

// Reopen a client socket without repeating DNS or ARP
SOCKET_INFO cache = TCPGetSocketInfo(skt); // Call with the old socket
skt = TCPOpen((DWORD)(PTR_BASE)&cache.remote, TCP_OPEN_NODE_INFO,
              cache.remotePort.Val, TCP_PURPOSE_DEFAULT);
```

**Parameters**

Parameters	Description
dwRemoteHost	For client sockets only. Provide a pointer to a null-terminated string of the remote host name (ex: "www.microchip.com" or "192.168.1.123"), a literal destination IP address (ex: 0x7B01A8C0 or an IP_ADDR data type), or a pointer to a NODE_INFO structure with the remote IP address and remote node or gateway MAC address specified. If a string is provided, note that it must be statically allocated in memory and cannot be modified or deallocated until TCPIsConnected (see page 449) returns TRUE. This parameter is ignored for server sockets.
vRemoteHostType	Any one of the following flags to identify the meaning of the dwRemoteHost parameter: <ul style="list-style-type: none"> <li>TCP_OPEN_SERVER (see page 440) - Open a server socket and ignore the dwRemoteHost parameter.</li> <li>TCP_OPEN_RAM_HOST (see page 439) - Open a client socket and connect (see page 165) it to a remote host who's name is stored as a null terminated string in a RAM array. Ex: "www.microchip.com" or "192.168.0.123" (BYTE* type)</li> <li>TCP_OPEN_ROM_HOST (see page 440) - Open a client socket and connect (see page 165) it to a remote host who's name is stored as a null terminated string in a literal string or ROM array. Ex: "www.microchip.com" or "192.168.0.123" (ROM BYTE* type)</li> <li>TCP_OPEN_IP_ADDRESS (see page 439) - Open a client socket and connect (see page 165) it to a remote IP address. Ex: 0x7B01A8C0 for 192.168.1.123 (DWORD type). Note that the byte ordering is big endian.</li> <li>TCP_OPEN_NODE_INFO (see page 439) - Open a client socket and connect (see page 165) it to a remote IP and MAC addresses pair stored in a NODE_INFO structure. dwRemoteHost must be a pointer to the NODE_INFO structure. This option is provided for backwards compatibility with applications built against prior stack versions that only implemented the TCPConnect (see page 441)() function. It can also be used to skip DNS and ARP resolution steps if connecting to a remote node which you've already connected to and have cached addresses for.</li> </ul>
wPort	TCP port to listen (see page 169) on or connect (see page 165) to: <ul style="list-style-type: none"> <li>Client sockets - the remote TCP port to which a connection should be made. The local port for client sockets will be automatically picked by the TCP module.</li> <li>Server sockets - the local TCP port on which to listen (see page 169) for connections.</li> </ul>
vSocketPurpose	Any of the TCP_PURPOSE_* constants defined in TCPIPConfig.h or the TCPIPConfig utility (see TCPSocketInitializer[] array).

**Return Values**

Return Values	Description
INVALID_SOCKET (see page 437)	No sockets of the specified type were available to be opened.
Otherwise	A TCP_SOCKET (see page 462) handle. Save this handle and use it when calling all other TCP APIs.

## 10.18.1.36 TCPPeek Function

**File**

TCP.h

**C**

```

BYTE TCPPeek(
    TCP_SOCKET hTCP,
    WORD wStart
);

```

**Description**

Peeks at one byte in the TCP RX FIFO without removing it from the buffer.

**Remarks**

Use the TCPPeekArray (see page 453)() function to read more than one byte. It will perform better than calling TCPPeek() in a loop.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to peak from (read without removing from stream).
wStart	Zero-indexed starting position within the FIFO to peek from.

## 10.18.1.37 TCPPeekArray Function

**File**

TCP.h

**C**

```

WORD TCPPeekArray(
    TCP_SOCKET hTCP,
    BYTE * vBuffer,
    WORD wLen,
    WORD wStart
);

```

**Description**

Reads a specified number of data bytes from the TCP RX FIFO without removing them from the buffer. No TCP control actions are taken as a result of this function (ex: no window update is sent to the remote node).

**Remarks**

None

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to peak from (read without removing from stream).
vBuffer	Destination to write the peeked data bytes.
wLen	Length of bytes to peak from the RX FIFO and copy to vBuffer.
wStart	Zero-indexed starting position within the FIFO to start peeking from.

### 10.18.1.38 TCPPut Function

**File**

TCP.h

**C**

```

BOOL TCPPut(
    TCP_SOCKET hTCP,
    BYTE byte
);
    
```

**Description**

Writes a single byte to a TCP socket.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to which data is to be written.
byte	The byte to write.

**Return Values**

Return Values	Description
TRUE	The byte was written to the transmit buffer.
FALSE	The transmit buffer was full, or the socket is not connected.

### 10.18.1.39 TCPPutArray Function

**File**

TCP.h

**C**

```

WORD TCPPutArray(
    TCP_SOCKET hTCP,
    BYTE* Data,
    WORD Len
);
    
```

**Returns**

The number of bytes written to the socket. If less than len, the buffer became full or the socket is not connected.

**Description**

Writes an array from RAM to a TCP socket.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to which data is to be written.
data	Pointer to the array to be written.
len	Number of bytes to be written.

## 10.18.1.40 TCPPutROMArray Function

**File**

TCP.h

**C**

```
WORD TCPPutROMArray(
    TCP_SOCKET hTCP,
    ROM BYTE* Data,
    WORD Len
);
```

**Returns**

The number of bytes written to the socket. If less than len, the buffer became full or the socket is not connected.

**Description**

Writes an array from ROM to a TCP socket.

**Remarks**

This function is aliased to TCPPutArray (see page 454) on non-PIC18 platforms.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to which data is to be written.
data	Pointer to the array to be written.
len	Number of bytes to be written.

## 10.18.1.41 TCPPutROMString Function

**File**

TCP.h

**C**

```
ROM BYTE* TCPPutROMString(
    TCP_SOCKET hTCP,
    ROM BYTE* Data
);
```

**Returns**

Pointer to the byte following the last byte written to the socket. If this pointer does not dereference to a NUL byte, the buffer became full or the socket is not connected.

**Description**

Writes a null-terminated string from ROM to a TCP socket. The null-terminator is not copied to the socket.

**Remarks**

The return value of this function differs from that of TCPPutArray (see page 454). To write long strings in a single state, initialize the \*data pointer to the first byte, then call this function repeatedly (breaking to the main stack loop after each call) until the return value dereferences to a NUL byte. Save the return value as the new starting \*data pointer otherwise.

This function is aliased to TCPPutString (see page 456) on non-PIC18 platforms.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to which data is to be written.
data	Pointer to the string to be written.

## 10.18.1.42 TCPPutString Function

**File**

TCP.h

**C**

```
BYTE* TCPPutString(
    TCP_SOCKET hTCP,
    BYTE* Data
);
```

**Returns**

Pointer to the byte following the last byte written to the socket. If this pointer does not dereference to a NUL byte, the buffer became full or the socket is not connected.

**Description**

Writes a null-terminated string from RAM to a TCP socket. The null-terminator is not copied to the socket.

**Remarks**

The return value of this function differs from that of TCPPutArray (see page 454). To write long strings in a single state, initialize the \*data pointer to the first byte, then call this function repeatedly (breaking to the main stack loop after each call) until the return value dereferences to a NUL byte. Save the return value as the new starting \*data pointer otherwise.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to which data is to be written.
data	Pointer to the string to be written.

## 10.18.1.43 TCPRAMCopy Function

**File**

TCP.c

**C**

```
static void TCPRAMCopy(
    PTR_BASE wDest,
    BYTE vDestType,
```

```

PTR_BASE wSource,
BYTE vSourceType,
WORD wLength
);
    
```

**Returns**

None

**Description**

This function copies data between memory mediums (PIC RAM, SPI RAM, and Ethernet buffer RAM).

**Remarks**

Copying to a destination region that overlaps with the source address is supported only if the destination start address is at a lower memory address (closer to 0x0000) than the source pointer. However, if they do overlap there must be at least 4 bytes of non-overlap to ensure correct results due to hardware DMA requirements.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
ptrDest	Address (see page 141) to write to
vDestType	Destination medium (TCP_PIC_RAM, TCP_ETH_RAM, TCP_SPI_RAM)
ptrSource	Address (see page 141) to copy from
vSourceType	Source medium (TCP_PIC_RAM, TCP_ETH_RAM, or TCP_SPI_RAM)
wLength	Number of bytes to copy

**Section**

Function Prototypes

## 10.18.1.44 TCPRAMCopyROM Function

**File**

TCP.c

**C**

```

static void TCPRAMCopyROM(
    PTR_BASE wDest,
    BYTE wDestType,
    ROM BYTE* wSource,
    WORD wLength
);
    
```

**Returns**

None

**Description**

This function copies data between memory mediums (PIC RAM, SPI RAM, and Ethernet buffer RAM). This function is to be used when copying from ROM.

**Remarks**

Copying to a destination region that overlaps with the source address is supported only if the destination start address is at a lower memory address (closer to 0x0000) than the source pointer.

This function is aliased to TCPRAMCopy (see page 456) on non-PIC18 platforms.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
wDest	Address (see page 141) to write to
wDestType	Destination medium (TCP_PIC_RAM, TCP_ETH_RAM, TCP_SPI_RAM)
wSource	Address (see page 141) to copy from
wLength	Number of bytes to copy

**10.18.1.45 TCPWasReset Function****File**

TCP.h

**C**

```

BOOL TCPWasReset (
    TCP_SOCKET hTCP
);

```

**Description**

This function is a self-clearing semaphore indicating whether or not a socket has been disconnected since the previous call. This function works for all possible disconnections: a call to TCPDisconnect (see page 442), a FIN from the remote node, or an acknowledgement timeout caused by the loss of a network link. It also returns TRUE after the first call to TCPInit (see page 463). Applications should use this function to reset their state machines.

This function was added due to the possibility of an error when relying on TCPIsConnected (see page 449) returning FALSE to check for a condition requiring a state machine reset. If a socket is closed (due to a FIN ACK) and then immediately reopened (due to the arrival of a new SYN) in the same cycle of the stack, calls to TCPIsConnected (see page 449) by the application will never return FALSE even though the socket has been disconnected. This can cause errors for protocols such as HTTP in which a client will immediately open a new connection upon closing of a prior one. Relying on this function instead allows applications to trap those conditions and properly reset their internal state for the new connection.

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
hTCP	The socket to check.

**Return Values**

Return Values	Description
TRUE	The socket has been disconnected since the previous call.
FALSE	The socket has not been disconnected since the previous call.

**10.18.2 TCP Stack Members****Enumerations**

	Name	Description
	TCP_STATE (see page 462)	TCP States as defined by RFC 793

**Functions**

	<b>Name</b>	<b>Description</b>
	TCPIInit ( <a href="#">see page 463</a> )	Initializes the TCP module.
	TCPProcess ( <a href="#">see page 464</a> )	Handles incoming TCP segments.
	TCPTick ( <a href="#">see page 464</a> )	Performs periodic TCP tasks.
	TCPSSLDecryptMAC ( <a href="#">see page 465</a> )	Decrypts and MACs data arriving via SSL.
	TCPStartSSLClientEx ( <a href="#">see page 465</a> )	Begins an SSL client session.

**Module**

TCP ([see page 434](#))

**Structures**

	<b>Name</b>	<b>Description</b>
	SOCKET_INFO ( <a href="#">see page 459</a> )	Information about a socket
	TCB ( <a href="#">see page 460</a> )	Remainder of TCP Control Block data. The rest of the TCB is stored in Ethernet buffer RAM or elsewhere as defined by vMemoryMedium. Current size is 41 (PIC18), 42 (PIC24/dsPIC), or 48 bytes (PIC32)
	TCB_STUB ( <a href="#">see page 461</a> )	TCP Control Block (TCB) stub data storage. Stubs are stored in local PIC RAM for speed. Current size is 34 bytes (PIC18), 36 bytes (PIC24/dsPIC), or 56 (PIC32)

**Types**

	<b>Name</b>	<b>Description</b>
	TCP_SOCKET ( <a href="#">see page 462</a> )	A TCP_SOCKET is stored as a single BYTE

**Description**

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

## 10.18.2.1 SOCKET\_INFO Structure

**File**

TCP.h

**C**

```
typedef struct {
    NODE_INFO remote;
    WORD_VAL remotePort;
} SOCKET_INFO;
```

**Members**

<b>Members</b>	<b>Description</b>
NODE_INFO remote;	NODE_INFO structure for remote node
WORD_VAL remotePort;	Port number associated with remote node

**Description**

Information about a socket

## 10.18.2.2 TCB Structure

### File

TCP.h

### C

```
typedef struct {
    DWORD retryInterval;
    DWORD MySEQ;
    DWORD RemoteSEQ;
    PTR_BASE txUnackedTail;
    WORD_VAL remotePort;
    WORD_VAL localPort;
    WORD remoteWindow;
    WORD wFutureDataSize;
    union {
        NODE_INFO niRemoteMACIP;
        DWORD dwRemoteHost;
    } remote;
    SHORT sHoleSize;
    struct {
        unsigned char bFINSent : 1;
        unsigned char bSYNSent : 1;
        unsigned char bRemoteHostIsROM : 1;
        unsigned char bRXNoneACKed1 : 1;
        unsigned char bRXNoneACKed2 : 1;
        unsigned char filler : 3;
    } flags;
    WORD wRemoteMSS;
    WORD_VAL localSSLPort;
    BYTE retryCount;
    BYTE vSocketPurpose;
} TCB;
```

### Members

Members	Description
DWORD retryInterval;	How long to wait before retrying transmission
DWORD MySEQ;	Local sequence number
DWORD RemoteSEQ;	Remote sequence number
PTR_BASE txUnackedTail;	TX tail pointer for data that is not yet acked
WORD_VAL remotePort;	Remote port number
WORD_VAL localPort;	Local port number
WORD remoteWindow;	Remote window size
WORD wFutureDataSize;	How much out-of-order data has been received
NODE_INFO niRemoteMACIP;	10 bytes for MAC and IP address
DWORD dwRemoteHost;	RAM or ROM pointer to a hostname string (ex: "www.microchip.com")
SHORT sHoleSize;	Size of the hole, or -1 for none exists. (0 indicates hole has just been filled)
unsigned char bFINSent : 1;	A FIN has been sent
unsigned char bSYNSent : 1;	A SYN has been sent
unsigned char bRemoteHostIsROM : 1;	Remote host is stored in ROM
unsigned char bRXNoneACKed1 : 1;	A duplicate ACK was likely received
unsigned char bRXNoneACKed2 : 1;	A second duplicate ACK was likely received
unsigned char filler : 3;	future use
WORD wRemoteMSS;	Maximum Segment Size option advertised by the remote node during initial handshaking
WORD_VAL localSSLPort;	Local SSL port number (for listening sockets)
BYTE retryCount;	Counter for transmission retries

BYTE vSocketPurpose;	Purpose of socket (as defined in TCPIPConfig.h)
----------------------	---

**Description**

Remainder of TCP Control Block data. The rest of the TCB is stored in Ethernet buffer RAM or elsewhere as defined by vMemoryMedium. Current size is 41 (PIC18), 42 (PIC24/dsPIC), or 48 bytes (PIC32)

### 10.18.2.3 TCB\_STUB Structure

**File**

TCP.h

**C**

```
typedef struct {
    PTR_BASE bufferTxStart;
    PTR_BASE bufferRxStart;
    PTR_BASE bufferEnd;
    PTR_BASE txHead;
    PTR_BASE txTail;
    PTR_BASE rxHead;
    PTR_BASE rxTail;
    DWORD eventTime;
    WORD eventTime2;
    union {
        WORD delayedACKTime;
        WORD closeWaitTime;
    } OverlappedTimers;
    TCP_STATE smState;
    struct {
        unsigned char vUnackedKeepalives : 3;
        unsigned char bServer : 1;
        unsigned char bTimerEnabled : 1;
        unsigned char bTimer2Enabled : 1;
        unsigned char bDelayedACKTimerEnabled : 1;
        unsigned char bOneSegmentReceived : 1;
        unsigned char bHalfFullFlush : 1;
        unsigned char bTXASAP : 1;
        unsigned char bTXASAPWithoutTimerReset : 1;
        unsigned char bTXFIN : 1;
        unsigned char bSocketReset : 1;
        unsigned char bSSLHandshaking : 1;
        unsigned char filler : 2;
    } Flags;
    WORD_VAL remoteHash;
    PTR_BASE sslTxHead;
    PTR_BASE sslRxHead;
    BYTE sslStubID;
    BYTE sslReqMessage;
    BYTE vMemoryMedium;
} TCB_STUB;
```

**Members**

Members	Description
PTR_BASE bufferTxStart;	First byte of TX buffer
PTR_BASE bufferRxStart;	First byte of RX buffer. TX buffer ends 1 byte prior
PTR_BASE bufferEnd;	Last byte of RX buffer
PTR_BASE txHead;	Head pointer for TX
PTR_BASE txTail;	Tail pointer for TX
PTR_BASE rxHead;	Head pointer for RX
PTR_BASE rxTail;	Tail pointer for RX
DWORD eventTime;	Packet retransmissions, state changes
WORD eventTime2;	Window updates, automatic transmission

WORD delayedACKTime;	Delayed Acknowledgement timer
WORD closeWaitTime;	TCP_CLOSE_WAIT timeout timer
TCP_STATE smState;	State of this socket
unsigned char vUnackedKeepalives : 3;	Count of how many keepalives have been sent with no response
unsigned char bServer : 1;	Socket should return to listening state when closed
unsigned char bTimerEnabled : 1;	Timer is enabled
unsigned char bTimer2Enabled : 1;	Second timer is enabled
unsigned char bDelayedACKTimerEnabled : 1;	DelayedACK timer is enabled
unsigned char bOneSegmentReceived : 1;	A segment has been received
unsigned char bHalfFullFlush : 1;	Flush is for being half full
unsigned char bTXASAP : 1;	Transmit as soon as possible (for Flush)
unsigned char bTXASAPWithoutTimerReset : 1;	Transmit as soon as possible (for Flush), but do not reset retransmission timers
unsigned char bTXFIN : 1;	FIN needs to be transmitted
unsigned char bSocketReset : 1;	Socket has been reset (self-clearing semaphore)
unsigned char bSSLHandshaking : 1;	Socket is in an SSL handshake
unsigned char filler : 2;	Future expansion
WORD_VAL remoteHash;	Consists of remoteIP, remotePort, localPort for connected sockets. It is a localPort number only for listening server sockets.
PTR_BASE sslTxHead;	Position of data being written in next SSL application record Also serves as cache of localSSLPort when smState = TCP_LISTENING
PTR_BASE sslRxHead;	Position of incoming data not yet handled by SSL
BYTE sslStubID;	Which sslStub (see page 428) is associated with this connection
BYTE sslReqMessage;	Currently requested SSL message
BYTE vMemoryMedium;	Which memory medium the TCB is actually stored

**Description**

TCP Control Block (TCB) stub data storage. Stubs are stored in local PIC RAM for speed. Current size is 34 bytes (PIC18), 36 bytes (PIC24/dsPIC), or 56 (PIC32)

### 10.18.2.4 TCP\_SOCKET Type

**File**

TCP.h

**C**

```
typedef BYTE TCP_SOCKET;
```

**Description**

A TCP\_SOCKET is stored as a single BYTE

### 10.18.2.5 TCP\_STATE Enumeration

**File**

TCP.h

**C**

```
typedef enum {
    TCP_GET_DNS_MODULE,
    TCP_DNS_RESOLVE,
    TCP_GATEWAY_SEND_ARP,

```

```
TCP_GATEWAY_GET_ARP ,
TCP_LISTEN,
TCP_SYN_SENT ,
TCP_SYN_RECEIVED ,
TCP_ESTABLISHED ,
TCP_FIN_WAIT_1 ,
TCP_FIN_WAIT_2 ,
TCP_CLOSING ,
TCP_CLOSE_WAIT ,
TCP_LAST_ACK ,
TCP_CLOSED ,
TCP_CLOSED_BUT_RESERVED
} TCP_STATE ;
```

**Members**

Members	Description
TCP_GET_DNS_MODULE	Special state for TCP client mode sockets
TCP_DNS_RESOLVE	Special state for TCP client mode sockets
TCP_GATEWAY_SEND_ARP	Special state for TCP client mode sockets
TCP_GATEWAY_GET_ARP	Special state for TCP client mode sockets
TCP_LISTEN	Socket is listening for connections
TCP_SYN_SENT	A SYN has been sent, awaiting an SYN+ACK
TCP_SYN_RECEIVED	A SYN has been received, awaiting an ACK
TCP_ESTABLISHED	Socket is connected and connection is established
TCP_FIN_WAIT_1	FIN WAIT state 1
TCP_FIN_WAIT_2	FIN WAIT state 2
TCP_CLOSING	Socket is closing TCP_TIME_WAIT, state is not implemented
TCP_CLOSE_WAIT	Waiting to close the socket
TCP_LAST_ACK	The final ACK has been sent
TCP_CLOSED	Socket is idle and unallocated
TCP_CLOSED_BUT_RESERVED	Special state for TCP client mode sockets. Socket is idle, but still allocated pending application closure of the handle.

**Description**

TCP States as defined by RFC 793

## 10.18.2.6 TCPInit Function

**File**

TCP.h

**C**

```
void TCPInit();
```

**Returns**

None

**Description**

Initializes the TCP module. This function sets up the TCP buffers in memory and initializes each socket to the CLOSED state. If insufficient memory was allocated for the TCP sockets, the function will hang here to be captured by the debugger.

**Remarks**

This function is called only one during lifetime of the application.

**Preconditions**

None

**Section**

Function Declarations

## 10.18.2.7 TCPProcess Function

**File**

TCP.h

**C**

```

BOOL TCPProcess(
    NODE_INFO* remote,
    IP_ADDR* localIP,
    WORD len
);

```

**Description**

This function handles incoming TCP segments. When a segment arrives, it is compared to open sockets using a hash of the remote port and IP. On a match, the data is passed to HandleTCPseg (see page 469) for further processing.

**Preconditions**

TCP is initialized and a TCP segment is ready in the MAC buffer.

**Parameters**

Parameters	Description
remote	Remote NODE_INFO structure
localIP	This stack's IP address (for header checking)
len	Total length of the waiting TCP segment

**Return Values**

Return Values	Description
TRUE	the segment was properly handled.
FALSE	otherwise

## 10.18.2.8 TCPTick Function

**File**

TCP.h

**C**

```

void TCPTick();

```

**Returns**

None

**Description**

This function performs any required periodic TCP tasks. Each socket's state machine is checked, and any elapsed timeout periods are handled.

**Preconditions**

TCP is initialized.

## 10.18.2.9 TCPSSLDecryptMAC Function

### File

TCP.h

### C

```
void TCPSSLDecryptMAC(
    TCP_SOCKET hTCP,
    ARCFOUR_CTX* ctx,
    WORD len
);
```

### Returns

None

### Description

This function decrypts data in the TCP buffer and calculates the MAC over the data. All data is left in the exact same location in the TCP buffer. It is called to help process incoming SSL records.

### Remarks

This function should never be called by an application. It is used only by the SSL module itself.

### Preconditions

TCP is initialized, hTCP is connected, and ctx's Sbox is loaded.

### Parameters

Parameters	Description
hTCP	TCP connection to decrypt in
ctx	ARCFOUR encryption context to use
len	Number of bytes to crypt
inPlace	TRUE to write back in place, FALSE to write at end of currently visible data.

## 10.18.2.10 TCPStartSSLClientEx Function

### File

TCP.h

### C

```
BOOL TCPStartSSLClientEx(
    TCP_SOCKET hTCP,
    BYTE* host,
    void * buffer,
    BYTE suppDataType
);
```

### Description

This function escalates the current connection to an SSL secured connection by initiating an SSL client handshake.

### Remarks

The host parameter is currently ignored and is not validated.

### Preconditions

TCP is initialized and hTCP is already connected.

**Parameters**

Parameters	Description
hTCP	TCP connection to secure
host	Expected host name on certificate (currently ignored)
buffer	Buffer for supplementary data return
suppDataType	Type of supplementary data to copy

**Return Values**

Return Values	Description
TRUE	an SSL connection was initiated
FALSE	Insufficient SSL resources (stubs) were available

## 10.18.3 TCP Internal Members

**Functions**

	Name	Description
⇒	CloseSocket (see page 468)	Closes a TCP socket.
⇒	FindMatchingSocket (see page 469)	Finds a suitable socket for a TCP segment.
⇒	HandleTCPSEg (see page 469)	Processes an incoming TCP segment.
⇒	SendTCP (see page 471)	Transmits a TPC segment.
⇒	SwapTCPHeader (see page 472)	Swaps endian-ness of a TCP header.
⇒	SyncTCB (see page 473)	Flushes MyTCB cache and loads up the specified TCB. Does nothing on cache hit.

**Macros**

	Name	Description
⇒	ACK (see page 468)	Acknowledge Flag as defined in RFC
⇒	FIN (see page 468)	FIN Flag as defined in RFC
⇒	LOCAL_PORT_END_NUMBER (see page 470)	End port for client sockets
⇒	LOCAL_PORT_START_NUMBER (see page 470)	Starting port for client sockets
⇒	PSH (see page 471)	Push Flag as defined in RFC
⇒	RST (see page 471)	Reset Flag as defined in RFC
⇒	SENDTCP_KEEP_ALIVE (see page 472)	Instead of transmitting normal data, a garbage octet is transmitted according to RFC 1122 section 4.2.3.6
⇒	SENDTCP_RESET_TIMERS (see page 472)	Indicates if this packet is a retransmission (no reset) or a new packet (reset required)
⇒	SYN (see page 473)	SYN Flag as defined in RFC
⇒	SyncTCBStub (see page 473)	Flushes MyTCBStub (see page 471) cache and loads up the specified TCB_STUB (see page 461). Does nothing on cache hit.
⇒	TCP_AUTO_TRANSMIT_TIMEOUT_VAL (see page 474)	Timeout before automatically transmitting unflushed data
⇒	TCP_WINDOW_UPDATE_TIMEOUT_VAL (see page 474)	Timeout before automatically transmitting a window update due to a TCPGet (see page 446)() or TCPGetArray (see page 447)() function call

	TCP_CLOSE_WAIT_TIMEOUT ( <a href="#">see page 474</a> )	Timeout for the CLOSE_WAIT state
	TCP_DELAYED_ACK_TIMEOUT ( <a href="#">see page 474</a> )	Timeout for delayed-acknowledgement algorithm
	TCP_FIN_WAIT_2_TIMEOUT ( <a href="#">see page 475</a> )	Timeout for FIN WAIT 2 state
	TCP_KEEP_ALIVE_TIMEOUT ( <a href="#">see page 476</a> )	Timeout for keep-alive messages when no traffic is sent
	TCP_MAX_RETRIES ( <a href="#">see page 476</a> )	Maximum number of retransmission attempts
	TCP_MAX_SEG_SIZE_RX ( <a href="#">see page 476</a> )	TCP Maximum Segment Size for RX. This value is advertised during connection establishment and the remote node should obey it. This should be set to 536 to avoid IP layer fragmentation from causing packet loss. However, raising its value can enhance performance at the (small) risk of introducing incompatibility with certain special remote nodes (ex: ones connected via a slow dial up modem).
	TCP_MAX_SEG_SIZE_TX ( <a href="#">see page 477</a> )	TCP Maximum Segment Size for TX. The TX maximum segment size is actually governed by the remote node's MSS option advertised during connection establishment. However, if the remote node specifies an unhandlably large MSS (ex: > Ethernet MTU), this define sets a hard limit so that we don't cause any TX buffer overflows. If the remote node does not advertise a MSS option, all TX segments are fixed at 536 bytes maximum.
	TCP_MAX_SYN_RETRIES ( <a href="#">see page 477</a> )	Smaller than all other retries to reduce SYN flood DoS duration
	TCP_MAX_UNACKED_KEEP_ALIVES ( <a href="#">see page 477</a> )	Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection
	TCP_OPTIMIZE_FOR_SIZE ( <a href="#">see page 477</a> )	For smallest size and best throughput, TCP_OPTIMIZE_FOR_SIZE should always be enabled on PIC24/dsPIC products. On PIC32 products there is very little difference and depends on compiler optimization level
	TCP_OPTIONS_END_OF_LIST ( <a href="#">see page 478</a> )	End of List TCP Option Flag
	TCP_OPTIONS_MAX_SEG_SIZE ( <a href="#">see page 478</a> )	Maximum segment size TCP flag
	TCP_OPTIONS_NO_OP ( <a href="#">see page 478</a> )	No Op TCP Option
	TCP_SOCKET_COUNT ( <a href="#">see page 479</a> )	Determines the number of defined TCP sockets
	TCP_START_TIMEOUT_VAL ( <a href="#">see page 479</a> )	Timeout to retransmit unacked data
	TCP_SYN_QUEUE_MAX_ENTRIES ( <a href="#">see page 480</a> )	Number of TCP RX SYN packets to save if they cannot be serviced immediately
	TCP_SYN_QUEUE_TIMEOUT ( <a href="#">see page 480</a> )	Timeout for when SYN queue entries are deleted if unserviceable
	URG ( <a href="#">see page 480</a> )	Urgent Flag as defined in RFC

**Module**

TCP ([see page 434](#))

**Structures**

	Name	Description
	TCP_HEADER ( <a href="#">see page 475</a> )	TCP Header Data Structure
	TCP_OPTIONS ( <a href="#">see page 478</a> )	TCP Options data structure

	TCP_SYN_QUEUE (see page 479)	Structure containing all the important elements of an incoming SYN packet in order to establish a connection at a future time if all sockets on the listening port are already connected to someone
---	------------------------------	---

**Variables**

	Name	Description
	hCurrentTCP (see page 470)	Current TCP socket
	MyTCB (see page 470)	Currently loaded TCB
	MyTCBStub (see page 471)	Alias to current TCP stub.
	SYNQueue (see page 473)	Array of saved incoming SYN requests that need to be serviced later
	TCBStubs (see page 473)	This is variable TCBStubs.

**Description**

The following functions and variables are designated as internal to the TCP module.

### 10.18.3.1 ACK Macro

**File**

TCP.c

**C**

```
#define ACK (0x10) // Acknowledge Flag as defined in RFC
```

**Description**

Acknowledge Flag as defined in RFC

### 10.18.3.2 CloseSocket Function

**File**

TCP.c

**C**

```
static void CloseSocket();
```

**Returns**

None

**Description**

This function closes a TCP socket. All socket state information is reset, and any buffered bytes are discarded. The socket is no longer accessible by the application after this point.

**Preconditions**

The TCPStub corresponding to the socket to be closed is synced.

### 10.18.3.3 FIN Macro

**File**

TCP.c

**C**

```
#define FIN (0x01) // FIN Flag as defined in RFC
```

**Description**

FIN Flag as defined in RFC

## 10.18.3.4 FindMatchingSocket Function

**File**

TCP.c

**C**

```
static BOOL FindMatchingSocket(
    TCP_HEADER* h,
    NODE_INFO* remote
);
```

**Description**

This function searches through the sockets and attempts to match one with a given TCP header and NODE\_INFO structure. If a socket is found, its index is saved in hCurrentTCP (see page 470) and the associated MyTCBStub (see page 471) and MyTCB are loaded. Otherwise, INVALID\_SOCKET (see page 437) is placed in hCurrentTCP (see page 470).

**Preconditions**

TCP is initialized.

**Parameters**

Parameters	Description
h	TCP header to be matched against
remote	The remote node who sent this header

**Return Values**

Return Values	Description
TRUE	A match was found and is loaded in hCurrentTCP (see page 470)
FALSE	No suitable socket was found and hCurrentTCP (see page 470) is INVALID_SOCKET (see page 437)

## 10.18.3.5 HandleTCPSeg Function

**File**

TCP.c

**C**

```
static void HandleTCPSeg(
    TCP_HEADER* h,
    WORD len
);
```

**Returns**

None

**Description**

Once an incoming segment has been matched to a socket, this function performs the necessary processing with the data. Depending on the segment and the state, this may include copying data to the TCP buffer, re-assembling out-of order packets, continuing an initialization or closing handshake, or closing the socket altogether.

**Preconditions**

TCP is initialized and the current TCP stub is already synced.

**Parameters**

Parameters	Description
h	The TCP header for this packet
len	The total buffer length of this segment

### 10.18.3.6 hCurrentTCP Variable

**File**

TCP.c

**C**

```
TCP_SOCKET hCurrentTCP = INVALID_SOCKET;
```

**Description**

Current TCP socket

### 10.18.3.7 LOCAL\_PORT\_END\_NUMBER Macro

**File**

TCP.c

**C**

```
#define LOCAL_PORT_END_NUMBER (5000u)
```

**Description**

End port for client sockets

### 10.18.3.8 LOCAL\_PORT\_START\_NUMBER Macro

**File**

TCP.c

**C**

```
#define LOCAL_PORT_START_NUMBER (1024u)
```

**Description**

Starting port for client sockets

### 10.18.3.9 MyTCB Variable

**File**

TCP.c

**C**

```
TCB MyTCB;
```

**Description**

Currently loaded TCB

## 10.18.3.10 MyTCBStub Variable

### File

TCP.c

### C

```
TCB_STUB MyTCBStub;
```

### Description

Alias to current TCP stub.

## 10.18.3.11 PSH Macro

### File

TCP.c

### C

```
#define PSH (0x08) // Push Flag as defined in RFC
```

### Description

Push Flag as defined in RFC

## 10.18.3.12 RST Macro

### File

TCP.c

### C

```
#define RST (0x04) // Reset Flag as defined in RFC
```

### Description

Reset Flag as defined in RFC

## 10.18.3.13 SendTCP Function

### File

TCP.c

### C

```
static void SendTCP(  
    BYTE vTCPFlags,  
    BYTE vSendFlags  
);
```

### Returns

None

### Description

This function assembles and transmits a TCP segment, including any pending data. It also supports retransmissions, keep-alives, and other packet types.

### Preconditions

TCP is initialized.

**Parameters**

Parameters	Description
vTCPFlags	Additional TCP flags to include
vSendFlags	Any combinations of SENDTCP_* constants to modify the transmit behavior or contents.

**10.18.3.14 SENDTCP\_KEEP\_ALIVE Macro****File**

TCP.c

**C**

```
#define SENDTCP_KEEP_ALIVE 0x02
```

**Description**

Instead of transmitting normal data, a garbage octet is transmitted according to RFC 1122 section 4.2.3.6

**10.18.3.15 SENDTCP\_RESET\_TIMERS Macro****File**

TCP.c

**C**

```
#define SENDTCP_RESET_TIMERS 0x01
```

**Description**

Indicates if this packet is a retransmission (no reset) or a new packet (reset required)

**10.18.3.16 SwapTCPHeader Function****File**

TCP.c

**C**

```
static void SwapTCPHeader(
    TCP_HEADER* header
);
```

**Returns**

None

**Description**

This function swaps the endian-ness of a given TCP header for comparison.

**Preconditions**

None

**Parameters**

Parameters	Description
header	The TCP header that is to be swapped

## 10.18.3.17 SYN Macro

### File

TCP.c

### C

```
#define SYN (0x02) // SYN Flag as defined in RFC
```

### Description

SYN Flag as defined in RFC

## 10.18.3.18 SyncTCB Function

### File

TCP.c

### C

```
static void SyncTCB();
```

### Description

Flushes MyTCB cache and loads up the specified TCB. Does nothing on cache hit.

## 10.18.3.19 SyncTCBStub Macro

### File

TCP.c

### C

```
#define SyncTCBStub(a) hCurrentTCP = (a)
```

### Description

Flushes MyTCBStub (see page 471) cache and loads up the specified TCB\_STUB (see page 461). Does nothing on cache hit.

## 10.18.3.20 SYNQueue Variable

### File

TCP.c

### C

```
TCP_SYN_QUEUE SYNQueue[TCP_SYN_QUEUE_MAX_ENTRIES];
```

### Description

Array of saved incoming SYN requests that need to be serviced later

## 10.18.3.21 TCBStubs Variable

### File

TCP.c

**C**

```
TCB_STUB TCBStubs[TCP_SOCKET_COUNT];
```

**Description**

This is variable TCBStubs.

### 10.18.3.22 TCP\_AUTO\_TRANSMIT\_TIMEOUT\_VAL Macro

**File**

TCP.c

**C**

```
#define TCP_AUTO_TRANSMIT_TIMEOUT_VAL (TICK_SECOND/25ull) // Timeout before  
automatically transmitting unflushed data
```

**Description**

Timeout before automatically transmitting unflushed data

### 10.18.3.23 TCP\_WINDOW\_UPDATE\_TIMEOUT\_VAL Macro

**File**

TCP.c

**C**

```
#define TCP_WINDOW_UPDATE_TIMEOUT_VAL (TICK_SECOND/5ull) // Timeout before automatically  
transmitting a window update due to a TCPGet() or TCPGetArray() function call
```

**Description**

Timeout before automatically transmitting a window update due to a TCPGet (see page 446)() or TCPGetArray (see page 447)() function call

### 10.18.3.24 TCP\_CLOSE\_WAIT\_TIMEOUT Macro

**File**

TCP.c

**C**

```
#define TCP_CLOSE_WAIT_TIMEOUT ((DWORD)TICK_SECOND/5) // Timeout for the CLOSE_WAIT state
```

**Description**

Timeout for the CLOSE\_WAIT state

### 10.18.3.25 TCP\_DELAYED\_ACK\_TIMEOUT Macro

**File**

TCP.c

**C**

```
#define TCP_DELAYED_ACK_TIMEOUT ((DWORD)TICK_SECOND/10) // Timeout for  
delayed-acknowledgement algorithm
```

**Description**

Timeout for delayed-acknowledgement algorithm

### 10.18.3.26 TCP\_FIN\_WAIT\_2\_TIMEOUT Macro

**File**

TCP.c

**C**

```
#define TCP_FIN_WAIT_2_TIMEOUT ((DWORD)TICK_SECOND*5) // Timeout for FIN WAIT 2 state
```

**Description**

Timeout for FIN WAIT 2 state

### 10.18.3.27 TCP\_HEADER Structure

**File**

TCP.c

**C**

```
typedef struct {
    WORD SourcePort;
    WORD DestPort;
    DWORD SeqNumber;
    DWORD AckNumber;
    struct {
        unsigned char Reserved3 : 4;
        unsigned char Val : 4;
    } DataOffset;
    union {
        struct {
            unsigned char flagFIN : 1;
            unsigned char flagSYN : 1;
            unsigned char flagRST : 1;
            unsigned char flagPSH : 1;
            unsigned char flagACK : 1;
            unsigned char flagURG : 1;
            unsigned char Reserved2 : 2;
        } bits;
        BYTE byte;
    } Flags;
    WORD Window;
    WORD Checksum;
    WORD UrgentPointer;
} TCP_HEADER;
```

**Members**

Members	Description
WORD SourcePort;	Local port number
WORD DestPort;	Remote port number
DWORD SeqNumber;	Local sequence number
DWORD AckNumber;	Acknowledging remote sequence number
struct { unsigned char Reserved3 : 4; unsigned char Val : 4; } DataOffset;	Data offset flags nibble

<pre> union { struct { unsigned char flagFIN : 1; unsigned char flagSYN : 1; unsigned char flagRST : 1; unsigned char flagPSH : 1; unsigned char flagACK : 1; unsigned char flagURG : 1; unsigned char Reserved2 : 2; } bits; BYTE byte; } Flags; WORD Window; WORD Checksum; WORD UrgentPointer; </pre>	<p>TCP Flags as defined in RFC</p> <p>Local free RX buffer window</p> <p>Data payload checksum</p> <p>Urgent pointer</p>
--	--

**Description**

TCP Header Data Structure

### 10.18.3.28 TCP\_KEEP\_ALIVE\_TIMEOUT Macro

**File**

TCP.c

**C**

```
#define TCP_KEEP_ALIVE_TIMEOUT ((DWORD)TICK_SECOND*10) // Timeout for keep-alive
messages when no traffic is sent
```

**Description**

Timeout for keep-alive messages when no traffic is sent

### 10.18.3.29 TCP\_MAX\_RETRIES Macro

**File**

TCP.c

**C**

```
#define TCP_MAX_RETRIES (5u) // Maximum number of retransmission attempts
```

**Description**

Maximum number of retransmission attempts

### 10.18.3.30 TCP\_MAX\_SEG\_SIZE\_RX Macro

**File**

TCP.c

**C**

```
#define TCP_MAX_SEG_SIZE_RX (536u)
```

**Description**

TCP Maximum Segment Size for RX. This value is advertised during connection establishment and the remote node should obey it. This should be set to 536 to avoid IP layer fragmentation from causing packet loss. However, raising its value can

enhance performance at the (small) risk of introducing incompatibility with certain special remote nodes (ex: ones connected via a slow dial up modem).

### 10.18.3.31 TCP\_MAX\_SEG\_SIZE\_TX Macro

#### File

TCP.c

#### C

```
#define TCP_MAX_SEG_SIZE_TX (1460u)
```

#### Description

TCP Maximum Segment Size for TX. The TX maximum segment size is actually governed by the remote node's MSS option advertised during connection establishment. However, if the remote node specifies an unhandably large MSS (ex: > Ethernet MTU), this define sets a hard limit so that we don't cause any TX buffer overflows. If the remote node does not advertise a MSS option, all TX segments are fixed at 536 bytes maximum.

### 10.18.3.32 TCP\_MAX\_SYN\_RETRIES Macro

#### File

TCP.c

#### C

```
#define TCP_MAX_SYN_RETRIES (2u) // Smaller than all other retries to reduce SYN flood DoS duration
```

#### Description

Smaller than all other retries to reduce SYN flood DoS duration

### 10.18.3.33 TCP\_MAX\_UNACKED\_KEEP\_ALIVES Macro

#### File

TCP.c

#### C

```
#define TCP_MAX_UNACKED_KEEP_ALIVES (6u) // Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection
```

#### Description

Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection

### 10.18.3.34 TCP\_OPTIMIZE\_FOR\_SIZE Macro

#### File

TCP.c

#### C

```
#define TCP_OPTIMIZE_FOR_SIZE
```

#### Description

For smallest size and best throughput, TCP\_OPTIMIZE\_FOR\_SIZE should always be enabled on PIC24/dsPIC products.

On PIC32 products there is very little difference and depends on compiler optimization level

### 10.18.3.35 TCP\_OPTIONS Structure

#### File

TCP.c

#### C

```
typedef struct {
    BYTE Kind;
    BYTE Length;
    WORD_VAL MaxSegSize;
} TCP_OPTIONS;
```

#### Members

Members	Description
BYTE Kind;	Type of option
BYTE Length;	Length
WORD_VAL MaxSegSize;	Maximum segment size

#### Description

TCP Options data structure

### 10.18.3.36 TCP\_OPTIONS\_END\_OF\_LIST Macro

#### File

TCP.c

#### C

```
#define TCP_OPTIONS_END_OF_LIST (0x00u) // End of List TCP Option Flag
```

#### Description

End of List TCP Option Flag

### 10.18.3.37 TCP\_OPTIONS\_MAX\_SEG\_SIZE Macro

#### File

TCP.c

#### C

```
#define TCP_OPTIONS_MAX_SEG_SIZE (0x02u) // Maximum segment size TCP flag
```

#### Description

Maximum segment size TCP flag

### 10.18.3.38 TCP\_OPTIONS\_NO\_OP Macro

#### File

TCP.c

#### C

```
#define TCP_OPTIONS_NO_OP (0x01u) // No Op TCP Option
```

**Description**

No Op TCP Option

### 10.18.3.39 TCP\_SOCKET\_COUNT Macro

**File**

TCP.c

**C**

```
#define TCP_SOCKET_COUNT (sizeof(TCPsocketInitializer)/sizeof(TCPsocketInitializer[0]))
```

**Description**

Determines the number of defined TCP sockets

### 10.18.3.40 TCP\_START\_TIMEOUT\_VAL Macro

**File**

TCP.c

**C**

```
#define TCP_START_TIMEOUT_VAL ((DWORD)TICK_SECOND*1) // Timeout to retransmit unacked data
```

**Description**

Timeout to retransmit unacked data

### 10.18.3.41 TCP\_SYN\_QUEUE Structure

**File**

TCP.c

**C**

```
typedef struct {
    NODE_INFO niSourceAddress;
    WORD wSourcePort;
    DWORD dwSourceSEQ;
    WORD wDestPort;
    WORD wTimestamp;
} TCP_SYN_QUEUE;
```

**Members**

Members	Description
NODE_INFO niSourceAddress;	Remote IP address and MAC address
WORD wSourcePort;	Remote TCP port number that the response SYN needs to be sent to
DWORD dwSourceSEQ;	Remote TCP SEquence number that must be ACKnowledged when we send our response SYN
WORD wDestPort;	Local TCP port which the original SYN was destined for
WORD wTimestamp;	Timer to expire old SYN packets that can't be serviced at all

**Description**

Structure containing all the important elements of an incoming SYN packet in order to establish a connection at a future time if all sockets on the listening port are already connected to someone

### 10.18.3.42 TCP\_SYN\_QUEUE\_MAX\_ENTRIES Macro

#### File

TCP.c

#### C

```
#define TCP_SYN_QUEUE_MAX_ENTRIES (3u) // Number of TCP RX SYN packets
to save if they cannot be serviced immediately
```

#### Description

Number of TCP RX SYN packets to save if they cannot be serviced immediately

### 10.18.3.43 TCP\_SYN\_QUEUE\_TIMEOUT Macro

#### File

TCP.c

#### C

```
#define TCP_SYN_QUEUE_TIMEOUT ((DWORD)TICK_SECOND*3) // Timeout for when SYN queue
entries are deleted if unserviceable
```

#### Description

Timeout for when SYN queue entries are deleted if unserviceable

### 10.18.3.44 URG Macro

#### File

TCP.c

#### C

```
#define URG (0x20) // Urgent Flag as defined in RFC
```

#### Description

Urgent Flag as defined in RFC

---

## 10.18.4 Variables

#### Module

TCP ([see page 434](#))

#### Variables

	Name	Description
	NextPort ( <a href="#">see page 480</a> )	Tracking variable for next local client port number

### 10.18.4.1 NextPort Variable

#### File

TCP.c

**C**WORD **NextPort** ;**Description**

Tracking variable for next local client port number

## 10.19 Telnet

Telnet provides bidirectional, interactive communication between two nodes on the Internet or on a Local Area Network. The Telnet code included with Microchip's TCP/IP stack is a demonstration of the structure of a Telnet application. This demo begins by listening for a Telnet connection. When a client attempts to make one, the demo will prompt the client for a username and password, and if the correct one is provided, will output and periodically refresh several values obtained from the demo board.

There are several changes that you may need to make to `Telnet.c` and/or `Telnet.h` to suit your application. All of the Telnet Public members can be re-defined in the application-specific section of `TCPIPConfig.h`. You may also wish to change some of the Telnet Internal Member strings, located in `Telnet.c`, to more accurately reflect your application. You will also need to modify the `TelnetTask` (see page 483) function to include the functionality you'd like. You may insert or change states in `TelnetTask` (see page 483) as needed.

### 10.19.1 Telnet Public Members

**Macros**

	Name	Description
↔	MAX_TELNET_CONNECTIONS (see page 481)	Number of simultaneously allowed Telnet (see page 481) sessions. Note that you must have an equal number of TCP_PURPOSE_TELNET type TCP sockets declared in the <code>TCPSocketInitializer[]</code> array above for multiple connections to work. If fewer sockets are available than this definition, then the the lesser of the two quantities will be the actual limit.
↔	TELNET_PASSWORD (see page 482)	Default Telnet (see page 481) password
↔	TELNET_PORT (see page 482)	Default local listening port for the Telnet (see page 481) server. Port 23 is the protocol default.
↔	TELNETS_PORT (see page 482)	Default local listening port for the Telnet (see page 481) server when SSL secured. Port 992 is the telnets protocol default.
↔	TELNET_USERNAME (see page 482)	Default username and password required to login to the Telnet (see page 481) server.

**Module**

Telnet (see page 481)

**Description**

The following functions and variables are available to the stack application.

#### 10.19.1.1 MAX\_TELNET\_CONNECTIONS Macro

**File**

TCPIP MRF24WB.h

**C**

```
#define MAX_TELNET_CONNECTIONS (1u)
```

**Description**

Number of simultaneously allowed Telnet (see page 481) sessions. Note that you must have an equal number of TCP\_PURPOSE\_TELNET type TCP sockets declared in the TCPSocketInitializer[] array above for multiple connections to work. If fewer sockets are available than this definition, then the lesser of the two quantities will be the actual limit.

## 10.19.1.2 TELNET\_PASSWORD Macro

**File**

TCPIP MRF24WB.h

**C**

```
#define TELNET_PASSWORD "microchip"
```

**Description**

Default Telnet (see page 481) password

## 10.19.1.3 TELNET\_PORT Macro

**File**

TCPIP MRF24WB.h

**C**

```
#define TELNET_PORT 23
```

**Description**

Default local listening port for the Telnet (see page 481) server. Port 23 is the protocol default.

## 10.19.1.4 TELNETS\_PORT Macro

**File**

TCPIP MRF24WB.h

**C**

```
#define TELNETS_PORT 992
```

**Description**

Default local listening port for the Telnet (see page 481) server when SSL secured. Port 992 is the telnets protocol default.

## 10.19.1.5 TELNET\_USERNAME Macro

**File**

TCPIP MRF24WB.h

**C**

```
#define TELNET_USERNAME "admin"
```

**Description**

Default username and password required to login to the Telnet (see page 481) server.

## 10.19.2 Telnet Stack Members

### Functions

	Name	Description
	TelnetTask ( <a href="#">see page 483</a> )	Performs Telnet ( <a href="#">see page 481</a> ) Server related tasks. Contains the Telnet ( <a href="#">see page 481</a> ) state machine and state tracking variables.

### Module

Telnet ([see page 481](#))

### Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.19.2.1 TelnetTask Function

#### File

Telnet.h

#### C

```
void TelnetTask();
```

#### Side Effects

None

#### Returns

None

#### Description

Performs Telnet ([see page 481](#)) Server related tasks. Contains the Telnet ([see page 481](#)) state machine and state tracking variables.

#### Remarks

None

#### Preconditions

Stack is initialized()

## 10.19.3 Telnet Internal Members

### Module

Telnet ([see page 481](#))

### Variables

	Name	Description
	strSpaces ( <a href="#">see page 484</a> )	String with extra spaces, for Demo
	strAuthenticated ( <a href="#">see page 484</a> )	Successful authentication message
	strDisplay ( <a href="#">see page 484</a> )	Demo output string

	strGoodBye ( <a href="#">see page 484</a> )	Demo disconnection message
	strPassword ( <a href="#">see page 485</a> )	DO Suppress Local Echo (stop telnet client from printing typed characters) Access denied message
	strTitle ( <a href="#">see page 485</a> )	Demo title string

**Description**

The following functions and variables are designated as internal to the Telnet ([see page 481](#)) module.

## 10.19.3.1 strSpaces Variable

**File**

Telnet.c

**C**

```
ROM BYTE strSpaces[] = "          ";
```

**Description**

String with extra spaces, for Demo

## 10.19.3.2 strAuthenticated Variable

**File**

Telnet.c

**C**

```
ROM BYTE strAuthenticated[] = "\r\nLogged in successfully\r\n\r\n" "\r\nPress 'q' to quit\r\n";
```

**Description**

Successful authentication message

## 10.19.3.3 strDisplay Variable

**File**

Telnet.c

**C**

```
ROM BYTE strDisplay[] = "\r\nSNTP Time:      (disabled)" "\r\nAnalog:          1023"
"\r\nButtons:      3 2 1 0" "\r\nLEDs:          7 6 5 4 3 2 1 0";
```

**Description**

Demo output string

## 10.19.3.4 strGoodBye Variable

**File**

Telnet.c

**C**

```
ROM BYTE strGoodBye[] = "\r\n\r\nGoodbye!\r\n";
```

**Description**

Demo disconnection message

### 10.19.3.5 strPassword Variable

**File**

Telnet.c

**C**

```
ROM BYTE strPassword[] = "Password: \xff\xfd\x2d";
```

**Description**

DO Suppress Local Echo (stop telnet client from printing typed characters) Access denied message

### 10.19.3.6 strTitle Variable

**File**

Telnet.c

**C**

```
ROM BYTE strTitle[] = "\x1b[2J\x1b[31m\x1b[1m" "Microchip Telnet Server 1.1\x1b[0m\r\n"  
"(for this demo, type 'admin' for the login and 'microchip' for the password.)\r\n" "Login:"  
";
```

**Description**

Demo title string

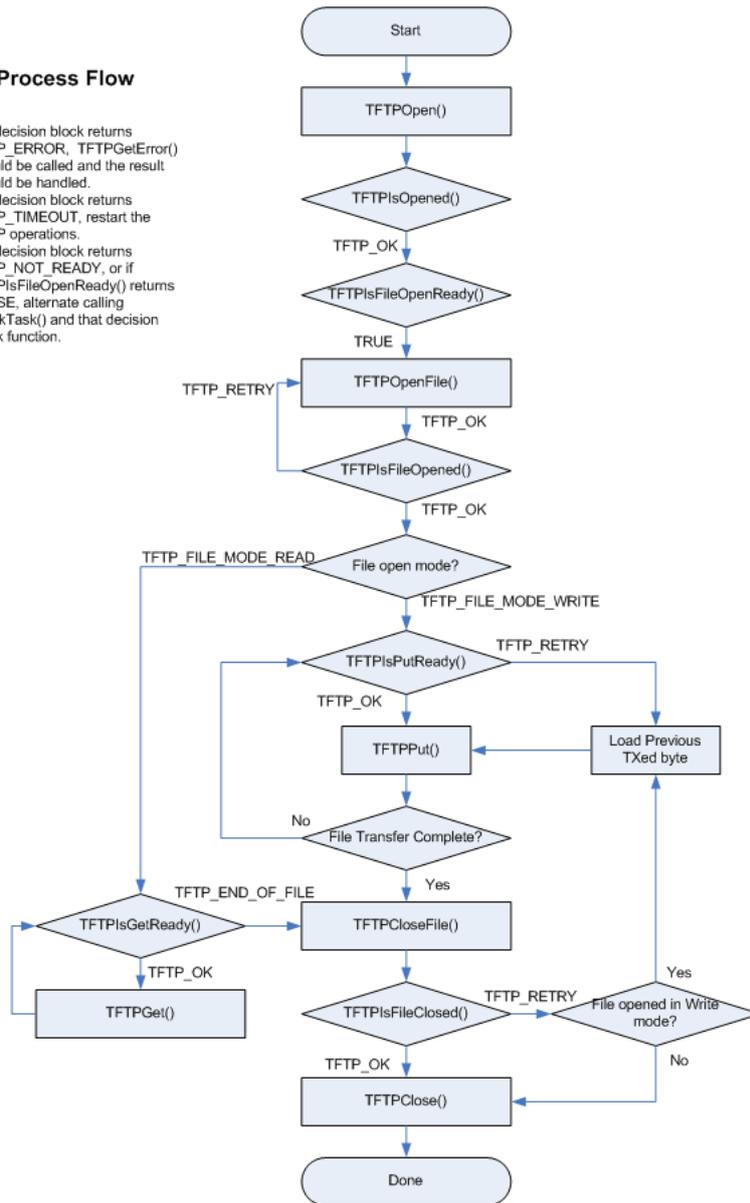
---

## 10.20 TFTP

The Trivial File Transfer Protocol provides unreliable upload and download services to applications connected to the UDP-based TFTP server.

**TFTP Process Flow Notes**

- If a decision block returns TFTP\_ERROR, TFTPGetError() should be called and the result should be handled.
- If a decision block returns TFTP\_TIMEOUT, restart the TFTP operations.
- If a decision block returns TFTP\_NOT\_READY, or if TFTPisFileOpenReady() returns FALSE, alternate calling StackTask() and that decision block function.



## 10.20.1 TFTP Public Members

**Enumerations**

	Name	Description
	TFTP_ACCESS_ERROR (see page 495)	Standard error codes as defined by TFTP spec. Use to decode value returned by TFTPGetError (see page 489).
	_TFTP_ACCESS_ERROR (see page 495)	Standard error codes as defined by TFTP spec. Use to decode value returned by TFTPGetError (see page 489).
	TFTP_FILE_MODE (see page 495)	File open mode as used by TFTPFileOpen().
	_TFTP_FILE_MODE (see page 495)	File open mode as used by TFTPFileOpen().
	TFTP_RESULT (see page 496)	Enum. of results returned by most of the TFTP functions.

	<code>_TFTP_RESULT</code> ( <a href="#">see page 496</a> )	Enum. of results returned by most of the TFTP functions.
---	--	--

**Functions**

	Name	Description
	<code>TFTPCloseFile</code> ( <a href="#">see page 488</a> )	Sends file closing messages.
	<code>TFTPGet</code> ( <a href="#">see page 489</a> )	Gets a data byte from data that was read.
	<code>TFTPIsFileClosed</code> ( <a href="#">see page 490</a> )	Determines if the file was closed.
	<code>TFTPIsFileOpened</code> ( <a href="#">see page 490</a> )	Determines if file has been opened.
	<code>TFTPIsGetReady</code> ( <a href="#">see page 491</a> )	Determines if a data block is ready to be read.
	<code>TFTPIsOpened</code> ( <a href="#">see page 492</a> )	Determines if the TFTP connection is open.
	<code>TFTPIsPutReady</code> ( <a href="#">see page 492</a> )	Determines if data can be written to a file.
	<code>TFTPOpen</code> ( <a href="#">see page 493</a> )	Initializes TFTP module.
	<code>TFTPOpenFile</code> ( <a href="#">see page 494</a> )	Prepares and sends TFTP file name and mode packet.
	<code>TFTPOpenROMFile</code> ( <a href="#">see page 494</a> )	PIC18 ROM argument implementation of <code>TFTPOpenFile</code> ( <a href="#">see page 494</a> )
	<code>TFTPPut</code> ( <a href="#">see page 495</a> )	Write a byte to a file.
	<code>TFTPGetUploadStatus</code> ( <a href="#">see page 496</a> )	Returns the TFTP file upload status started by calling the <code>TFTPUploadRAMFileToHost</code> ( <a href="#">see page 498</a> ()) or <code>TFTPUploadFragmentedRAMFileToHost</code> ( <a href="#">see page 497</a> ()) functions.
	<code>TFTPUploadFragmentedRAMFileToHost</code> ( <a href="#">see page 497</a> )	Uploads an random, potentially non-contiguous, array of RAM bytes as a file to a remote TFTP server.
	<code>TFTPUploadRAMFileToHost</code> ( <a href="#">see page 498</a> )	Uploads a contiguous array of RAM bytes as a file to a remote TFTP server.

**Macros**

	Name	Description
	<code>TFTPClose</code> ( <a href="#">see page 488</a> )	Macro: void <code>TFTPClose(void)</code> Closes TFTP client socket.
	<code>TFTPGetError</code> ( <a href="#">see page 489</a> )	Macro: WORD <code>TFTPGetError(void)</code> Returns previously saved error code.
	<code>TFTPIsFileOpenReady</code> ( <a href="#">see page 491</a> )	Macro: BOOL <code>TFTPIsFileOpenReady(void)</code> Checks to see if it is okay to send TFTP file open request to remote server.
	<code>TFTP_UPLOAD_COMPLETE</code> ( <a href="#">see page 499</a> )	Status codes for <code>TFTPGetUploadStatus</code> ( <a href="#">see page 496</a> ()) function. Zero means upload success, >0 means working and <0 means fatal error.
	<code>TFTP_UPLOAD_CONNECT</code> ( <a href="#">see page 499</a> )	This is macro <code>TFTP_UPLOAD_CONNECT</code> .
	<code>TFTP_UPLOAD_CONNECT_TIMEOUT</code> ( <a href="#">see page 499</a> )	This is macro <code>TFTP_UPLOAD_CONNECT_TIMEOUT</code> .
	<code>TFTP_UPLOAD_GET_DNS</code> ( <a href="#">see page 499</a> )	This is macro <code>TFTP_UPLOAD_GET_DNS</code> .
	<code>TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT</code> ( <a href="#">see page 500</a> )	This is macro <code>TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT</code> .
	<code>TFTP_UPLOAD_RESOLVE_HOST</code> ( <a href="#">see page 500</a> )	This is macro <code>TFTP_UPLOAD_RESOLVE_HOST</code> .

	TFTP_UPLOAD_SEND_DATA ( <a href="#">see page 500</a> )	This is macro TFTP_UPLOAD_SEND_DATA.
	TFTP_UPLOAD_SEND_FILENAME ( <a href="#">see page 500</a> )	This is macro TFTP_UPLOAD_SEND_FILENAME.
	TFTP_UPLOAD_SERVER_ERROR ( <a href="#">see page 500</a> )	This is macro TFTP_UPLOAD_SERVER_ERROR.
	TFTP_UPLOAD_WAIT_FOR_CLOSURE ( <a href="#">see page 501</a> )	This is macro TFTP_UPLOAD_WAIT_FOR_CLOSURE.

**Module**

TFTP ([see page 485](#))

**Structures**

	Name	Description
	TFTP_CHUNK_DESCRIPTOR ( <a href="#">see page 498</a> )	This is type TFTP_CHUNK_DESCRIPTOR.

**Description**

The following functions and variables are available to the stack application.

## 10.20.1.1 TFTPclose Macro

**File**

TFTPC.h

**C**

```
#define TFTPclose(void) UDPClose(_tftpSocket)
```

**Side Effects**

None

**Returns**

None

**Description**

Macro: void TFTPclose(void)

Closes TFTP client socket.

**Remarks**

Once closed, application must do TFTPOpen ([see page 493](#)) to perform any new TFTP operations.

If TFTP server does not change during application life-time, one may not need to call TFTPclose and keep TFTP socket open.

**Preconditions**

TFTPOpen ([see page 493](#)) is already called and TFTPisOpened ([see page 492](#)()) returned TFTP\_OK.

## 10.20.1.2 TFTPcloseFile Function

**File**

TFTPC.h

**C**

```
void TFTPcloseFile();
```

**Side Effects**

None

**Returns**

None

**Description**

If file is opened in read mode, it makes sure that last ACK is sent to server. If file is opened in write mode, it makes sure that last block is sent out to server and waits for server to respond with ACK.

**Remarks**

TFTP\_IsFileClosed (see page 490)() must be called to confirm if file was really closed.

**Preconditions**

TFTPOpenFile (see page 494)() was called and TFTP\_IsFileOpened (see page 490)() had returned with TFTP\_OK.

## 10.20.1.3 TFTPGet Function

**File**

TFTPc.h

**C**

```
BYTE TFTPGet();
```

**Side Effects**

None

**Returns**

data byte as received from remote server.

**Description**

Fetches next data byte from TFTP socket. If end of data block is reached, it issues ack to server so that next data block can be received.

**Remarks**

Use this function to read file from server.

**Preconditions**

TFTPOpenFile (see page 494)() is called with TFTP\_FILE\_MODE\_READ and TFTP\_IsGetReady (see page 491)() = TRUE

## 10.20.1.4 TFTPGetError Macro

**File**

TFTPc.h

**C**

```
#define TFTPGetError (_tftpError)
```

**Side Effects**

None

**Returns**

Error code as returned by remote server. Application may use TFTP\_ACCESS\_ERROR (see page 495) enum. to decode standard error code.

**Description**

Macro: WORD TFTPGetError(void)

Returns previously saved error code.

**Remarks**

None

**Preconditions**

One of the TFTP function returned with TFTP\_ERROR result.

## 10.20.1.5 TFTP\_IsFileClosed Function

**File**

TFTPC.h

**C**

```
TFTP_RESULT TFTP_IsFileClosed();
```

**Side Effects**

None

**Returns**

TFTP\_OK if file was successfully closed

TFTP\_RETRY if file mode was Write and remote server did not receive last packet. Application must retry with last block.

TFTP\_TIMEOUT if all attempts were exhausted in closing file.

TFTP\_ERROR if remote server sent an error in response to last block. Actual error code may be read by calling TFTPGetError (see page 489)

TFTP\_NOT\_READY if file is not closed yet.

**Description**

If file mode is Read, it simply makes that last block is acknowledged. If file mode is Write, it waits for server ack. If no ack was received within specified timeout instructs application to resend last block. It keeps track of retries and declares timeout all attempts were exhausted.

**Remarks**

None

**Preconditions**

TFTPCloseFile (see page 488)() is already called.

## 10.20.1.6 TFTP\_IsFileOpened Function

**File**

TFTPC.h

**C**

```
TFTP_RESULT TFTP_IsFileOpened();
```

**Side Effects**

None

**Returns**

TFTP\_OK if file is ready to be read or written  
TFTP\_RETRY if previous attempt was timed out needs to be retried.  
TFTP\_TIMEOUT if all attempts were exhausted.  
TFTP\_ERROR if remote server responded with error  
TFTP\_NOT\_READY if file is not yet opened.

**Description**

Waits for remote server response regarding previous attempt to open file. If no response is received within specified timeout, function returns with TFTP\_RETRY and application logic must issue another TFTPFileOpen().

**Remarks**

None

**Preconditions**

TFTPOpenFile (see page 494)() is called.

## 10.20.1.7 TFTP\_IsFileOpenReady Macro

**File**

TFTPc.h

**C**

```
#define TFTP_IsFileOpenReady UDPIsPutReady(_tftpSocket)
```

**Side Effects**

None

**Returns**

TRUE, if it is ok to call TFTPOpenFile (see page 494)() FALSE, if otherwise.

**Description**

Macro: BOOL TFTP\_IsFileOpenReady(void)

Checks to see if it is okay to send TFTP file open request to remote server.

**Remarks**

None

**Preconditions**

TFTPOpen (see page 493) is already called and TFTP\_IsOpened (see page 492)() returned TFTP\_OK.

## 10.20.1.8 TFTP\_IsGetReady Function

**File**

TFTPc.h

**C**

```
TFTP_RESULT TFTP_IsGetReady();
```

**Side Effects**

None

**Returns**

TFTP\_OK if there is more data byte available to read

TFTP\_TIMEOUT if timeout occurred waiting for new data.

TFTP\_END\_OF\_FILE if end of file has reached.

TFTP\_ERROR if remote server returned ERROR. Actual error code may be read by calling TFTPGetError (see page 489)()

TFTP\_NOT\_READY if still waiting for new data.

**Description**

Waits for data block. If data block does not arrive within specified timeout, it automatically sends out ack for previous block to remind server to send next data block. If all attempts are exhausted, it returns with TFTP\_TIMEOUT.

**Remarks**

By default, this function uses "octet" or binary mode of file transfer.

**Preconditions**

TFTPOpenFile (see page 494)() is called with TFTP\_FILE\_MODE\_READ and TFTPisFileOpened (see page 490)() returned with TRUE.

## 10.20.1.9 TFTPisOpened Function

**File**

TFTPc.h

**C**

```
TFTP_RESULT TFTPisOpened( );
```

**Side Effects**

None

**Returns**

TFTP\_OK if previous call to TFTPOpen (see page 493) is complete

TFTP\_TIMEOUT if remote host did not respond to previous ARP request.

TFTP\_NOT\_READY if remote has still not responded and timeout has not expired.

**Description**

Waits for ARP reply and opens a UDP socket to perform further TFTP operations.

**Remarks**

Once opened, application may keep TFTP socket open and future TFTP operations. If TFTPclose (see page 488)() is called to close the connection TFTPOpen (see page 493)() must be called again before performing any other TFTP operations.

**Preconditions**

TFTPOpen (see page 493)() is already called.

## 10.20.1.10 TFTPisPutReady Function

**File**

TFTPc.h

**C**

```
TFTP_RESULT TFTPisPutReady( );
```

**Side Effects**

None

**Returns**

TFTP\_OK if it is okay to write more data byte.

TFTP\_TIMEOUT if timeout occurred waiting for ack from server

TFTP\_RETRY if all server did not send ack on time and application needs to resend last block.

TFTP\_ERROR if remote server returned ERROR. Actual error code may be read by calling TFTPGetError (see page 489)()

TFTP\_NOT\_READY if still waiting...

**Description**

Waits for ack from server. If ack does not arrive within specified timeout, it instructs application to retry last block by returning TFTP\_RETRY.

If all attempts are exhausted, it returns with TFTP\_TIMEOUT.

**Remarks**

None

**Preconditions**

TFTPOpenFile (see page 494)() is called with TFTP\_FILE\_MODE\_WRITE and TFTPisFileOpened (see page 490)() returned with TRUE.

## 10.20.1.11 TFTPOpen Function

**File**

TFTPc.h

**C**

```
void TFTPOpen(  
    IP_ADDR * host  
);
```

**Side Effects**

None

**Returns**

None

**Description**

Initiates ARP for given host and prepares TFTP module for next sequence of function calls.

**Remarks**

Use TFTPisOpened (see page 492)() to check if a connection was successfully opened or not.

**Preconditions**

UDP module is already initialized and at least one UDP socket is available.

**Parameters**

Parameters	Description
host	IP address of remote TFTP server

## 10.20.1.12 TFTPOpenFile Function

**File**

TFTPC.h

**C**

```
void TFTPOpenFile(
    BYTE * fileName,
    TFTP_FILE_MODE mode
);
```

**Side Effects**

None

**Returns**

None

**Description**

Prepares and sends TFTP file name and mode packet.

**Remarks**

By default, this function uses "octet" or binary mode of file transfer. Use `TFTPIsFileOpened` (see page 490) to check if file is ready to be read or written.

**Preconditions**

`TFTPIsFileOpenReady() = TRUE`

**Parameters**

Parameters	Description
fileName	File name that is to be opened.
mode	Mode of file access Must be <code>TFTP_FILE_MODE_READ</code> for read <code>TFTP_FILE_MODE_WRITE</code> for write

## 10.20.1.13 TFTPOpenROMFile Function

**File**

TFTPC.h

**C**

```
void TFTPOpenROMFile(
    ROM BYTE * fileName,
    TFTP_FILE_MODE mode
);
```

**Description**

PIC18 ROM argument implementation of `TFTPOpenFile` (see page 494)

## 10.20.1.14 TFTPPut Function

### File

TFTPC.h

### C

```
void TFTPput(
    BYTE c
);
```

### Side Effects

None

### Returns

None

### Description

Puts given data byte into TFTP socket. If end of data block is reached, it transmits entire block.

### Remarks

Use this function to write file to server.

### Preconditions

TFTPOpenFile (see page 494)() is called with TFTP\_FILE\_MODE\_WRITE and TFTPisPutReady (see page 492)() = TRUE

### Parameters

Parameters	Description
c	Data byte that is to be written

## 10.20.1.15 TFTP\_ACCESS\_ERROR Enumeration

### File

TFTPC.h

### C

```
typedef enum _TFTP_ACCESS_ERROR {
    TFTP_ERROR_NOT_DEFINED = 0,
    TFTP_ERROR_FILE_NOT_FOUND,
    TFTP_ERROR_ACCESS_VIOLATION,
    TFTP_ERROR_DISK_FULL,
    TFTP_ERROR_INVALID_OPERATION,
    TFTP_ERROR_UNKNOWN_TID,
    TFTP_ERROR_FILE_EXISTS,
    TFTP_ERROR_NO_SUCH_USE
} TFTP_ACCESS_ERROR;
```

### Description

Standard error codes as defined by TFTP spec. Use to decode value returned by TFTPGetError (see page 489)().

## 10.20.1.16 TFTP\_FILE\_MODE Enumeration

### File

TFTPC.h

**C**

```
typedef enum _TFTP_FILE_MODE {  
    TFTP_FILE_MODE_READ = 1,  
    TFTP_FILE_MODE_WRITE = 2  
} TFTP_FILE_MODE;
```

**Description**

File open mode as used by TFTPFileOpen().

## 10.20.1.17 TFTP\_RESULT Enumeration

**File**

TFTPC.h

**C**

```
typedef enum _TFTP_RESULT {  
    TFTP_OK = 0,  
    TFTP_NOT_READY,  
    TFTP_END_OF_FILE,  
    TFTP_ERROR,  
    TFTP_RETRY,  
    TFTP_TIMEOUT  
} TFTP_RESULT;
```

**Description**

Enum. of results returned by most of the TFTP functions.

## 10.20.1.18 TFTPGetUploadStatus Function

**File**

TFTPC.h

**C**

```
CHAR TFTPGetUploadStatus();
```

**Returns**

A status code. Negative results are fatal errors. Positive results indicate the TFTP upload operation is still being processed. A zero result indicates successful file upload completion (TFTP API is now idle and available for further calls). Specific return values are as follows: 0 (TFTP\_UPLOAD\_COMPLETE (see page 499)): Upload completed successfully 1 (TFTP\_UPLOAD\_GET\_DNS (see page 499)): Attempting to obtain DNS client module 2 (TFTP\_UPLOAD\_RESOLVE\_HOST (see page 500)): Attempting to resolve TFTP hostname 3 (TFTP\_UPLOAD\_CONNECT (see page 499)): Attempting to ARP and contact the TFTP server 4 (TFTP\_UPLOAD\_SEND\_FILENAME (see page 500)): Attempting to send the filename and receive acknowledgement. 5 (TFTP\_UPLOAD\_SEND\_DATA (see page 500)): Attempting to send the file contents and receive acknowledgement. 6 (TFTP\_UPLOAD\_WAIT\_FOR\_CLOSURE (see page 501)): Attempting to send the final packet of file contents and receive acknowledgement. -1 (TFTP\_UPLOAD\_HOST\_RESOLVE\_TIMEOUT (see page 500)): Couldn't resolve hostname -2 (TFTP\_UPLOAD\_CONNECT\_TIMEOUT (see page 499)): Couldn't finish ARP and reach server -3 (TFTP\_UPLOAD\_SERVER\_ERROR (see page 500)): TFTP server returned an error (ex: access denial) or file upload failed due to a timeout (partial file may have been uploaded).

**Description**

Returns the TFTP file upload status started by calling the TFTPUploadRAMFileToHost (see page 498)() or TFTPUploadFragmentedRAMFileToHost (see page 497)() functions.

**Remarks**

The DNS client module must be enabled to use this function. i.e. STACK\_USE\_DNS must be defined in TCPIPConfig.h.

**Preconditions**

None

## 10.20.1.19 TFTPUploadFragmentedRAMFileToHost Function

**File**

TFTPC.h

**C**

```
void TFTPUploadFragmentedRAMFileToHost(
    ROM BYTE * vRemoteHost,
    ROM BYTE * vFilename,
    TFTP_CHUNK_DESCRIPTOR * vFirstChunkDescriptor
);
```

**Returns**

None

**Description**

Uploads an random, potentially non-contiguous, array of RAM bytes as a file to a remote TFTP server.

**Remarks**

The DNS client module must be enabled to use this function. i.e. STACK\_USE\_DNS must be defined in TCPIPConfig.h.

Call the TFTPGetUploadStatus (see page 496)() function to determine the status of the file upload.

It is only possible to have one TFTP operation active at any given time. After starting a TFTP operation by calling TFTPUploadRAMFileToHost (see page 498)() or TFTPUploadFragmentedRAMFileToHost(), you must wait until TFTPGetUploadStatus (see page 496)() returns a completion status code (<=0) before calling any other TFTP API functions.

**Preconditions**

None

**Parameters**

Parameters	Description
vRemoteHost	ROM string of the remote TFTP server to upload to (ex: "www.myserver.com"). For device architectures that make no distinction between RAM and ROM pointers (PIC24, dsPIC and PIC32), this string must remain allocated and unmodified in RAM until the TFTP upload process completes (as indicated by TFTPGetUploadStatus (see page 496)()).
vFilename	ROM string of the remote file to create/overwrite (ex: "status.txt"). For device architectures that make no distinction between RAM and ROM pointers (PIC24, dsPIC and PIC32), this string must remain allocated and unmodified in RAM until the TFTP upload process completes (as indicated by TFTPGetUploadStatus (see page 496)()).
vFirstChunkDescriptor	Pointer to a static or global (persistent) array of TFTP_CHUNK_DESCRIPTOR (see page 498) structures describing what RAM memory addresses the file contents should be obtained from. The TFTP_CHUNK_DESCRIPTOR.vDataPointer field should be set to the memory address of the data to transmit, and the TFTP_CHUNK_DESCRIPTOR.wDataLength field should be set to the number of bytes to transmit from the given pointer. The TFTP_CHUNK_DESCRIPTOR (see page 498) array must be terminated by a dummy descriptor whos TFTP_CHUNK_DESCRIPTOR.vDataPointer pointer is set to NULL. Refer to the TFTPUploadRAMFileToHost (see page 498)() API for an example calling sequence since it merely a wrapper to this TFTPUploadFragmentedRAMFileToHost() function.

## 10.20.1.20 TFTPUploadRAMFileToHost Function

**File**

TFTPC.h

**C**

```
void TFTPUploadRAMFileToHost (
    ROM BYTE * vRemoteHost,
    ROM BYTE * vFilename,
    BYTE * vData,
    WORD wDataLength
);
```

**Returns**

None

**Description**

Uploads a contiguous array of RAM bytes as a file to a remote TFTP server.

**Remarks**

The DNS client module must be enabled to use this function. i.e. STACK\_USE\_DNS must be defined in TCPIPConfig.h.

Call the TFTPGetUploadStatus (see page 496)() function to determine the status of the file upload.

It is only possible to have one TFTP operation active at any given time. After starting a TFTP operation by calling TFTPUploadRAMFileToHost() or TFTPUploadFragmentedRAMFileToHost (see page 497)(), you must wait until TFTPGetUploadStatus (see page 496)() returns a completion status code (<=0) before calling any other TFTP API functions.

**Preconditions**

None

**Parameters**

Parameters	Description
vRemoteHost	ROM string of the remote TFTP server to upload to (ex: "www.myserver.com"). For device architectures that make no distinction between RAM and ROM pointers (PIC24, dsPIC and PIC32), this string must remain allocated and unmodified in RAM until the TFTP upload process completes (as indicated by TFTPGetUploadStatus (see page 496)()).
vFilename	ROM string of the remote file to create/overwrite (ex: "status.txt"). For device architectures that make no distinction between RAM and ROM pointers (PIC24, dsPIC and PIC32), this string must remain allocated and unmodified in RAM until the TFTP upload process completes (as indicated by TFTPGetUploadStatus (see page 496)()).
vData	Pointer to a RAM array of data to write to the file.
wDataLength	Number of bytes pointed to by vData. This will be the final file size of the uploaded file. Note that since this is defined as a WORD type, the maximum possible file size is 65535 bytes. For longer files, call the TFTPUploadFragmentedRAMFileToHost (see page 497)() function instead.

## 10.20.1.21 TFTP\_CHUNK\_DESCRIPTOR Structure

**File**

TFTPC.h

**C**

```
typedef struct {
```

```
    BYTE * vDataPointer;  
    WORD  wDataLength;  
} TFTP_CHUNK_DESCRIPTOR;
```

**Description**

This is type TFTP\_CHUNK\_DESCRIPTOR.

## 10.20.1.22 TFTP\_UPLOAD\_COMPLETE Macro

**File**

TFTPC.h

**C**

```
#define TFTP_UPLOAD_COMPLETE 0
```

**Description**

Status codes for TFTPGetUploadStatus (see page 496)() function. Zero means upload success, >0 means working and <0 means fatal error.

## 10.20.1.23 TFTP\_UPLOAD\_CONNECT Macro

**File**

TFTPC.h

**C**

```
#define TFTP_UPLOAD_CONNECT 3
```

**Description**

This is macro TFTP\_UPLOAD\_CONNECT.

## 10.20.1.24 TFTP\_UPLOAD\_CONNECT\_TIMEOUT Macro

**File**

TFTPC.h

**C**

```
#define TFTP_UPLOAD_CONNECT_TIMEOUT -2
```

**Description**

This is macro TFTP\_UPLOAD\_CONNECT\_TIMEOUT.

## 10.20.1.25 TFTP\_UPLOAD\_GET\_DNS Macro

**File**

TFTPC.h

**C**

```
#define TFTP_UPLOAD_GET_DNS 1
```

**Description**

This is macro TFTP\_UPLOAD\_GET\_DNS.

## 10.20.1.26 TFTP\_UPLOAD\_HOST\_RESOLVE\_TIMEOUT Macro

### File

TFTPC.h

### C

```
#define TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT -1
```

### Description

This is macro TFTP\_UPLOAD\_HOST\_RESOLVE\_TIMEOUT.

## 10.20.1.27 TFTP\_UPLOAD\_RESOLVE\_HOST Macro

### File

TFTPC.h

### C

```
#define TFTP_UPLOAD_RESOLVE_HOST 2
```

### Description

This is macro TFTP\_UPLOAD\_RESOLVE\_HOST.

## 10.20.1.28 TFTP\_UPLOAD\_SEND\_DATA Macro

### File

TFTPC.h

### C

```
#define TFTP_UPLOAD_SEND_DATA 5
```

### Description

This is macro TFTP\_UPLOAD\_SEND\_DATA.

## 10.20.1.29 TFTP\_UPLOAD\_SEND\_FILENAME Macro

### File

TFTPC.h

### C

```
#define TFTP_UPLOAD_SEND_FILENAME 4
```

### Description

This is macro TFTP\_UPLOAD\_SEND\_FILENAME.

## 10.20.1.30 TFTP\_UPLOAD\_SERVER\_ERROR Macro

### File

TFTPC.h

**C**

```
#define TFTP_UPLOAD_SERVER_ERROR -3
```

**Description**

This is macro TFTP\_UPLOAD\_SERVER\_ERROR.

## 10.20.1.31 TFTP\_UPLOAD\_WAIT\_FOR\_CLOSURE Macro

**File**

TFTPC.h

**C**

```
#define TFTP_UPLOAD_WAIT_FOR_CLOSURE 6
```

**Description**

This is macro TFTP\_UPLOAD\_WAIT\_FOR\_CLOSURE.

## 10.20.2 TFTP Stack Members

**Macros**

	Name	Description
	TFTP_ARP_TIMEOUT_VAL (  see page 501)	Number of seconds to wait before declaring TIMEOUT error on Put
	TFTP_GET_TIMEOUT_VAL (  see page 502)	Number of seconds to wait before declaring TIMEOUT error on Get.
	TFTP_MAX_RETRIES (  see page 502)	Number of attempts before declaring TIMEOUT error.

**Module**

TFTP ( see page 485)

**Description**

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.20.2.1 TFTP\_ARP\_TIMEOUT\_VAL Macro

**File**

TFTPC.h

**C**

```
#define TFTP_ARP_TIMEOUT_VAL (3u * TICKS_PER_SECOND)
```

**Description**

Number of seconds to wait before declaring TIMEOUT error on Put

### 10.20.2.2 TFTP\_GET\_TIMEOUT\_VAL Macro

**File**

TFTPC.h

**C**

```
#define TFTP_GET_TIMEOUT_VAL (3u * TICKS_PER_SECOND)
```

**Description**

Number of seconds to wait before declaring TIMEOUT error on Get.

### 10.20.2.3 TFTP\_MAX\_RETRIES Macro

**File**

TFTPC.h

**C**

```
#define TFTP_MAX_RETRIES (3u)
```

**Description**

Number of attempts before declaring TIMEOUT error.

## 10.20.3 TFTP Internal Members

**Enumerations**

	Name	Description
	TFTP_OPCODE ( <a href="#">see page 504</a> )	Enumeration of TFTP opcodes
	TFTP_STATE ( <a href="#">see page 505</a> )	The TFTP state machine

**Functions**

	Name	Description
	_TFTPSendAck ( <a href="#">see page 506</a> )	Private helper function
	_TFTPSendFileName ( <a href="#">see page 506</a> )	Private helper function
	_TFTPSendROMFileName ( <a href="#">see page 507</a> )	PIC18 ROM variable argument implementation of _TFTPSendFileName ( <a href="#">see page 506</a> )

**Macros**

	Name	Description
	TFTP_BLOCK_SIZE ( <a href="#">see page 504</a> )	The size of a TFTP block - 512 bytes
	TFTP_BLOCK_SIZE_MSB ( <a href="#">see page 504</a> )	The MSB of the TFTP_BLOCK_SIZE ( <a href="#">see page 504</a> )
	TFTP_CLIENT_PORT ( <a href="#">see page 504</a> )	The TFTP Client port - a unique port on this device
	TFTP_SERVER_PORT ( <a href="#">see page 505</a> )	The TFTP Server Port

**Module**

TFTP (see page 485)

**Variables**

	Name	Description
	MutexVar (see page 503)	Mutually Exclusive variable groups to conserve RAM.
	_tftpError (see page 505)	Variable to preserve error condition causes for later transmission
	_tftpFlags (see page 505)	TFTP status flags
	_tftpRetries (see page 506)	Tracker variable for the number of TFTP retries
	_tftpSocket (see page 507)	TFTP Socket for TFTP server link
	_tftpStartTick (see page 507)	Timing variable used to detect timeout conditions
	_tftpState (see page 507)	TFTP state machine tracker variable
	smUpload (see page 507)	This is variable smUpload.
	uploadChunkDescriptor (see page 508)	This is variable uploadChunkDescriptor.
	uploadChunkDescriptorForRetransmit (see page 508)	This is variable uploadChunkDescriptorForRetransmit.
	vUploadFilename (see page 508)	This is variable vUploadFilename.
	vUploadRemoteHost (see page 508)	TFTPUploadRAMFileToHost (see page 498)(), TFTPUploadFragmentedRAMFileToHost (see page 497)() and TFTPGetUploadStatus (see page 496)() functions require the DNS client module to be enabled for them to work. The RAM and ROM resources for these functions can be preserved if the DNS client module isn't enabled.
	wUploadChunkOffset (see page 509)	This is variable wUploadChunkOffset.
	wUploadChunkOffsetForRetransmit (see page 509)	This is variable wUploadChunkOffsetForRetransmit.

**Description**

The following functions and variables are designated as internal to the TFTP module.

### 10.20.3.1 MutexVar Variable

**File**

TFTPC.c

**C**

```

union {
    struct {
        NODE_INFO _hostInfo;
    } group1;
    struct {
        WORD_VAL _tftpBlockNumber;
        WORD_VAL _tftpDuplicateBlock;
        WORD_VAL _tftpBlockLength;
    } group2;
} MutexVar;
    
```

**Description**

Mutually Exclusive variable groups to conserve RAM.

## 10.20.3.2 TFTP\_BLOCK\_SIZE Macro

### File

TFTPc.c

### C

```
#define TFTP_BLOCK_SIZE (0x200L)
```

### Description

The size of a TFTP block - 512 bytes

## 10.20.3.3 TFTP\_BLOCK\_SIZE\_MSB Macro

### File

TFTPc.c

### C

```
#define TFTP_BLOCK_SIZE_MSB (0x02u)
```

### Description

The MSB of the TFTP\_BLOCK\_SIZE (see page 504)

## 10.20.3.4 TFTP\_CLIENT\_PORT Macro

### File

TFTPc.c

### C

```
#define TFTP_CLIENT_PORT 65352L
```

### Description

The TFTP Client port - a unique port on this device

## 10.20.3.5 TFTP\_OPCODE Enumeration

### File

TFTPc.c

### C

```
typedef enum {
    TFTP_OPCODE_RRQ = 1,
    TFTP_OPCODE_WRQ,
    TFTP_OPCODE_DATA,
    TFTP_OPCODE_ACK,
    TFTP_OPCODE_ERROR
} TFTP_OPCODE;
```

### Members

Members	Description
TFTP_OPCODE_RRQ = 1	Get
TFTP_OPCODE_WRQ	Put
TFTP_OPCODE_DATA	Actual data

TFTP_OPCODE_ACK	Ack for Get/Put
TFTP_OPCODE_ERROR	Error

**Description**

Enumeration of TFTP opcodes

**10.20.3.6 TFTP\_SERVER\_PORT Macro****File**

TFTPC.c

**C**

```
#define TFTP_SERVER_PORT (69L)
```

**Description**

The TFTP Server Port

**10.20.3.7 TFTP\_STATE Enumeration****File**

TFTPC.c

**C**

```
typedef enum {
    SM_TFTP_WAIT = 0,
    SM_TFTP_READY,
    SM_TFTP_WAIT_FOR_DATA,
    SM_TFTP_WAIT_FOR_ACK,
    SM_TFTP_DUPLICATE_ACK,
    SM_TFTP_SEND_ACK,
    SM_TFTP_SEND_LAST_ACK
} TFTP_STATE;
```

**Description**

The TFTP state machine

**10.20.3.8 \_tftpError Variable****File**

TFTPC.c

**C**

```
WORD _tftpError;
```

**Description**

Variable to preserve error condition causes for later transmission

**10.20.3.9 \_tftpFlags Variable****File**

TFTPC.c

**C**

```
union {
  struct {
    unsigned int bIsFlushed : 1;
    unsigned int bIsAked : 1;
    unsigned int bIsClosed : 1;
    unsigned int bIsClosing : 1;
    unsigned int bIsReading : 1;
  } bits;
  BYTE Val;
} _tftpFlags;
```

**Description**

TFTP status flags

### 10.20.3.10 \_tftpRetries Variable

**File**

TFTPC.c

**C**

```
BYTE _tftpRetries;
```

**Description**

Tracker variable for the number of TFTP retries

### 10.20.3.11 \_TFTPSendAck Function

**File**

TFTPC.c

**C**

```
static void _TFTPSendAck(
    WORD_VAL blockNumber
);
```

**Description**

Private helper function

### 10.20.3.12 \_TFTPSendFileName Function

**File**

TFTPC.c

**C**

```
static void _TFTPSendFileName(
    TFTP_OPCODE command,
    BYTE * fileName
);
```

**Description**

Private helper function

### 10.20.3.13 **\_TFTPSendROMFileName** Function

**File**

TFTPC.c

**C**

```
static void _TFTPSendROMFileName(  
    TFTP_OPCODE opcode,  
    ROM BYTE * fileName  
);
```

**Description**

PIC18 ROM variable argument implementation of `_TFTPSendFileName` ([↗](#) see page 506)

### 10.20.3.14 **\_tftpSocket** Variable

**File**

TFTPC.c

**C**

```
UDP_SOCKET _tftpSocket;
```

**Description**

TFTP Socket for TFTP server link

### 10.20.3.15 **\_tftpStartTick** Variable

**File**

TFTPC.c

**C**

```
DWORD _tftpStartTick;
```

**Description**

Timing variable used to detect timeout conditions

### 10.20.3.16 **\_tftpState** Variable

**File**

TFTPC.c

**C**

```
TFTP_STATE _tftpState;
```

**Description**

TFTP state machine tracker variable

### 10.20.3.17 **smUpload** Variable

**File**

TFTPC.c

**C**

```
CHAR smUpload = TFTP_UPLOAD_COMPLETE;
```

**Description**

This is variable smUpload.

### 10.20.3.18 uploadChunkDescriptor Variable

**File**

TFTPC.c

**C**

```
TFTP_CHUNK_DESCRIPTOR * uploadChunkDescriptor;
```

**Description**

This is variable uploadChunkDescriptor.

### 10.20.3.19 uploadChunkDescriptorForRetransmit Variable

**File**

TFTPC.c

**C**

```
TFTP_CHUNK_DESCRIPTOR * uploadChunkDescriptorForRetransmit;
```

**Description**

This is variable uploadChunkDescriptorForRetransmit.

### 10.20.3.20 vUploadFilename Variable

**File**

TFTPC.c

**C**

```
ROM BYTE * vUploadFilename;
```

**Description**

This is variable vUploadFilename.

### 10.20.3.21 vUploadRemoteHost Variable

**File**

TFTPC.c

**C**

```
ROM BYTE * vUploadRemoteHost;
```

**Description**

TFTPUploadRAMFileToHost (see page 498()), TFTPUploadFragmentedRAMFileToHost (see page 497()) and TFTPGetUploadStatus (see page 496()) functions require the DNS client module to be enabled for them to work. The RAM and ROM resources for these functions can be preserved if the DNS client module isn't enabled.

### 10.20.3.22 wUploadChunkOffset Variable

#### File

TFTPc.c

#### C

```
WORD wUploadChunkOffset;
```

#### Description

This is variable wUploadChunkOffset.

### 10.20.3.23 wUploadChunkOffsetForRetransmit Variable

#### File

TFTPc.c

#### C

```
WORD wUploadChunkOffsetForRetransmit;
```

#### Description

This is variable wUploadChunkOffsetForRetransmit.

## 10.21 Tick Module

The Tick module provides accurate time-keeping capabilities based on the hardware clock. By default, it uses Timer 0 on 8-bit parts and Timer 1 on 16- and 32-bit families. The module is interrupt driven, which makes the timing stable and accurate. As such, it is also suitable for a real-time clock.

The Tick module exists to assist with the implementation of non-blocking delays and timeouts. Rather than using a loop to count to a specific number, use the Tick module and compare a previous time with the current time. In this fashion applications can return its unused cycles to the stack during long delays, which increases the overall efficiency of the system.

Tick works best in conjunction with a state machine. In general, call TickGet (see page 512) and store the result. Return to the main stack application, and on future calls compare the current Tick value to the stored one. The constants TICK\_SECOND (see page 511), TICK\_MINUTE (see page 511), and TICK\_HOUR (see page 511) can be used to compare against logical time increments.

The following example implements a delay of 0.5 seconds using the Tick module:

```
TICK startTime;

// ...state machine and other states

case SM_SET_DELAY:
    startTime = TickGet();
    sm = SM_DELAY_WAIT;
    return;

case SM_DELAY_WAIT:
    if((LONG)(TickGet() - startTime) < TICK_SECOND/2)
        return;

case SM_DELAY_DONE:
    // This state is entered only after 0.5 second elapses.
```

Ticks are stored internally as 48-bit integers. Using the various TickGet (see page 512), TickGetDiv256 (see page 512),

and TickGetDiv64K (see page 513) functions the Tick is suitable for measuring time increments from a few microseconds to a few years.

If absolute timestamps are required, the SNTP Client module may be more appropriate.

## 10.21.1 Tick Public Members

### Functions

	Name	Description
⇒	TickConvertToMilliseconds (see page 511)	Converts a Tick value or difference to milliseconds.
⇒	TickGet (see page 512)	Obtains the current Tick value.
⇒	TickGetDiv256 (see page 512)	Obtains the current Tick value divided by 256.
⇒	TickGetDiv64K (see page 513)	Obtains the current Tick value divided by 64K.

### Macros

	Name	Description
↔	TICK_HOUR (see page 511)	Represents one hour in Ticks
↔	TICK_MINUTE (see page 511)	Represents one minute in Ticks
↔	TICK_SECOND (see page 511)	Represents one second in Ticks

### Module

Tick Module (see page 509)

### Types

	Name	Description
🔍	TICK (see page 510)	All TICKS are stored as 32-bit unsigned integers. This is deprecated since it conflicts with other TICK definitions used in other Microchip software libraries and therefore poses a merge and maintenance problem. Instead of using the TICK data type, just use the base DWORD data type instead.

### Description

The following functions and variables are available to the stack application.

### 10.21.1.1 TICK Type

#### File

Tick.h

#### C

```
typedef DWORD TICK;
```

#### Description

All TICKS are stored as 32-bit unsigned integers. This is deprecated since it conflicts with other TICK definitions used in other Microchip software libraries and therefore poses a merge and maintenance problem. Instead of using the TICK data type, just use the base DWORD data type instead.

## 10.21.1.2 TICK\_HOUR Macro

### File

Tick.h

### C

```
#define TICK_HOUR ((QWORD)TICKS_PER_SECOND*3600ull)
```

### Description

Represents one hour in Ticks

## 10.21.1.3 TICK\_MINUTE Macro

### File

Tick.h

### C

```
#define TICK_MINUTE ((QWORD)TICKS_PER_SECOND*60ull)
```

### Description

Represents one minute in Ticks

## 10.21.1.4 TICK\_SECOND Macro

### File

Tick.h

### C

```
#define TICK_SECOND ((QWORD)TICKS_PER_SECOND)
```

### Description

Represents one second in Ticks

## 10.21.1.5 TickConvertToMilliseconds Function

### File

Tick.h

### C

```
DWORD TickConvertToMilliseconds(  
    DWORD dwTickValue  
);
```

### Returns

Input value expressed in milliseconds.

### Description

This function converts a Tick value or difference to milliseconds. For example, `TickConvertToMilliseconds(32768)` returns 1000 when a 32.768kHz clock with no prescaler drives the Tick module interrupt.

### Remarks

This function performs division on DWORDs, which is slow. Avoid using it unless you absolutely must (such as displaying

data to a user). For timeout comparisons, compare the current value to a multiple or fraction of TICK\_SECOND (see page 511), which will be calculated only once at compile time.

**Preconditions**

None

**Parameters**

Parameters	Description
dwTickValue	Value to convert to milliseconds

## 10.21.1.6 TickGet Function

**File**

Tick.h

**C**

```
DWORD TickGet();
```

**Returns**

Lower 32 bits of the current Tick value.

**Description**

This function retrieves the current Tick value, allowing timing and measurement code to be written in a non-blocking fashion. This function retrieves the least significant 32 bits of the internal tick counter, and is useful for measuring time increments ranging from a few microseconds to a few hours. Use TickGetDiv256 (see page 512) or TickGetDiv64K (see page 513) for longer periods of time.

**Preconditions**

None

## 10.21.1.7 TickGetDiv256 Function

**File**

Tick.h

**C**

```
DWORD TickGetDiv256();
```

**Returns**

Middle 32 bits of the current Tick value.

**Description**

This function retrieves the current Tick value, allowing timing and measurement code to be written in a non-blocking fashion. This function retrieves the middle 32 bits of the internal tick counter, and is useful for measuring time increments ranging from a few minutes to a few weeks. Use TickGet (see page 512) for shorter periods or TickGetDiv64K (see page 513) for longer ones.

**Preconditions**

None

## 10.21.1.8 TickGetDiv64K Function

### File

Tick.h

### C

```
DWORD TickGetDiv64K();
```

### Returns

Upper 32 bits of the current Tick value.

### Description

This function retrieves the current Tick value, allowing timing and measurement code to be written in a non-blocking fashion. This function retrieves the most significant 32 bits of the internal tick counter, and is useful for measuring time increments ranging from a few days to a few years, or for absolute time measurements. Use TickGet (see page 512) or TickGetDiv256 (see page 512) for shorter periods of time.

### Preconditions

None

---

## 10.21.2 Tick Stack Functions

### Functions

	Name	Description
◆	TickInit (see page 513)	Initializes the Tick manager module.
◆	TickUpdate (see page 514)	Updates the tick value when an interrupt occurs.

### Module

Tick Module (see page 509)

### Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

### 10.21.2.1 TickInit Function

#### File

Tick.h

#### C

```
void TickInit();
```

#### Returns

None

#### Description

Configures the Tick module and any necessary hardware resources.

#### Remarks

This function is called only one during lifetime of the application.

**Preconditions**

None

## 10.21.2.2 TickUpdate Function

**File**

Tick.h

**C**

```
void TickUpdate();
```

**Returns**

None

**Description**

Updates the tick value when an interrupt occurs.

**Preconditions**

None

## 10.21.3 Tick Internal Members

**Functions**

	Name	Description
	GetTickCopy ( <a href="#">see page 515</a> )	Reads the tick value.

**Macros**

	Name	Description
	TICKS_PER_SECOND ( <a href="#">see page 515</a> )	Internal core clock drives timer with 1:256 prescaler #define TICKS_PER_SECOND (32768ul) // 32kHz crystal drives timer with no scalar

**Module**Tick Module ([see page 509](#))**Variables**

	Name	Description
	dwInternalTicks ( <a href="#">see page 514</a> )	Internal counter to store Ticks. This variable is incremented in an ISR and therefore must be marked volatile to prevent the compiler optimizer from reordering code to use this value in the main context while interrupts are disabled.
	vTickReading ( <a href="#">see page 515</a> )	6-byte value to store Ticks. Allows for use over longer periods of time.

**Description**

The following functions and variables are designated as internal to the Tick module.

### 10.21.3.1 dwInternalTicks Variable

**File**

Tick.c

**C**

```
volatile DWORD dwInternalTicks = 0;
```

**Description**

Internal counter to store Ticks. This variable is incremented in an ISR and therefore must be marked volatile to prevent the compiler optimizer from reordering code to use this value in the main context while interrupts are disabled.

## 10.21.3.2 GetTickCopy Function

**File**

Tick.c

**C**

```
static void GetTickCopy();
```

**Returns**

None

**Description**

This function performs an interrupt-safe and synchronized read of the 48-bit Tick value.

**Preconditions**

None

## 10.21.3.3 TICKS\_PER\_SECOND Macro

**File**

Tick.h

**C**

```
#define TICKS_PER_SECOND ((GetPeripheralClock()+128u11)/256u11) // Internal core clock  
drives timer with 1:256 prescaler
```

**Description**

Internal core clock drives timer with 1:256 prescaler #define TICKS\_PER\_SECOND (32768ul) // 32kHz crystal drives timer with no scalar

## 10.21.3.4 vTickReading Variable

**File**

Tick.c

**C**

```
BYTE vTickReading[6];
```

**Description**

6-byte value to store Ticks. Allows for use over longer periods of time.

# 10.22 UDP

## Types

	Name	Description
	UDP_STATE ( <a href="#">see page 536</a> )	UDP States

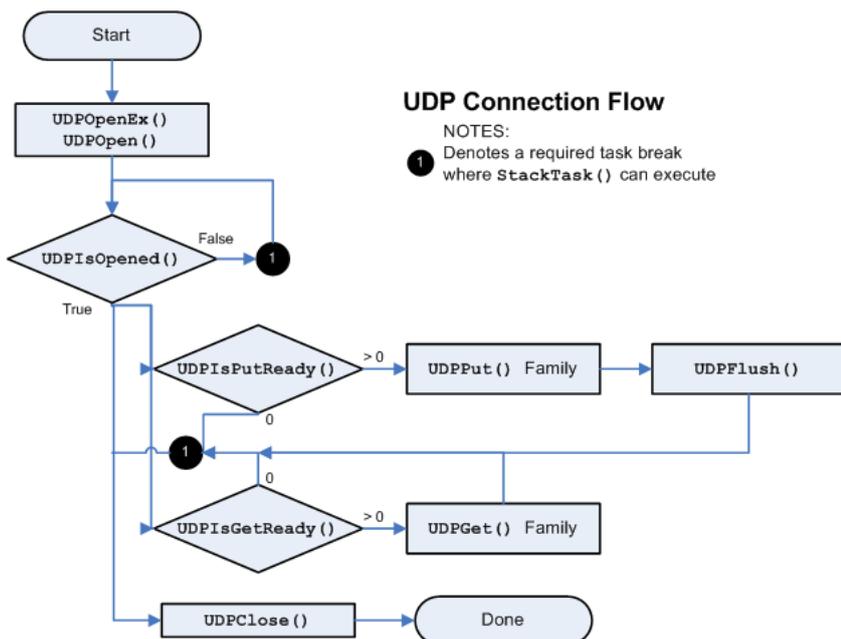
## Description

UDP is a standard transport layer protocol described in RFC 768. It provides fast but unreliable data-gram based transfers over networks, and forms the foundation SNTP, SNMP, DNS, and many other protocol standards.

Connections over UDP should be thought of as data-gram based transfers. Each packet is a separate entity, the application should expect some packets to arrive out-of-order or even fail to reach the destination node. This is in contrast to TCP, in which the connection is thought of as a stream and network errors are automatically corrected. These tradeoffs in reliability are made for an increase in throughput. In general, UDP transfers operate 2 to 3 times faster than those made over TCP.

Since UDP is packet-oriented, each packet must be dealt with in its entirety by your application before returning to the main stack loop. When a packet is received, your application will be called to handle it. This packet will no longer be available the next time your application is called, so you must either perform all necessary processing or copy the data elsewhere before returning. When transmitting a packet, your application must build and transmit the complete packet in one cycle.

The UDP flow diagram below provides an overview for the use of the UDP module:



Sockets ([see page 146](#)) are opened using UDPOpen ([see page 520](#)). This function can either open a listening socket to wait for incoming segments, or can make a client connection to a remote node. When making a client connection, you will need to perform any required DNS and/or ARP resolution using those modules directly before invoking UDPOpen ([see page 520](#)).

Once the socket is opened, you can immediately begin transmitting data. To transmit a segment, call UDPIsPutReady ([see page 524](#)) to determine how many bytes can be written and to designate a currently active socket. Then, use any of the UDPPut ([see page 524](#)) family of functions to write data to the socket. Once all data has been written, call UDPFlush ([see page 522](#)) to build and transmit the packet. This sequence must be accomplished all in one step. If your application returns to the main stack loop after calling UDPPut ([see page 524](#)) but before calling UDPFlush ([see page 522](#)), the data may be lost or the module may behave unpredictably.

To check for received segments, call `UDPIsGetReady` (see page 523). If the return value is non-zero, your application must consume the segment by reading data with the `UDPGet` (see page 522) family. Once all data has been read, return to the main stack loop to wait for an additional segment. UDP segments are only stored for one iteration of the cooperative multi-tasking loop, so your application must complete its processing on a segment or copy it elsewhere before returning. Note that this behavior differs from TCP, which buffers incoming data through multiple stack cycles.

When a socket is no longer needed, call `UDPClose` (see page 521) to release it back to the pool for future use.

## 10.22.1 UDP Public Members

### Functions

	Name	Description
⇒	<code>UDPOpenEx</code> (see page 519)	Opens a UDP socket for a client.
⇒	<code>UDPClose</code> (see page 521)	Closes a UDP socket and frees the handle.
⇒	<code>UDPDiscard</code> (see page 521)	Discards any remaining RX data from a UDP socket.
⇒	<code>UDPFlush</code> (see page 522)	Transmits all pending data in a UDP socket.
⇒	<code>UDPGet</code> (see page 522)	Reads a byte from the currently active socket.
⇒	<code>UDPGetArray</code> (see page 523)	Reads an array of bytes from the currently active socket.
⇒	<code>UDPIsGetReady</code> (see page 523)	Determines how many bytes can be read from the UDP socket.
⇒	<code>UDPIsPutReady</code> (see page 524)	Determines how many bytes can be written to the UDP socket.
⇒	<code>UDPPut</code> (see page 524)	Writes a byte to the currently active socket.
⇒	<code>UDPPutArray</code> (see page 525)	Writes an array of bytes to the currently active socket.
⇒	<code>UDPPutROMArray</code> (see page 525)	Writes an array of bytes from ROM to the currently active socket.
⇒	<code>UDPPutROMString</code> (see page 526)	Writes null-terminated string from ROM to the currently active socket.
⇒	<code>UDPPutString</code> (see page 526)	Writes null-terminated string to the currently active socket.
⇒	<code>UDPSetRxBuffer</code> (see page 527)	Moves the pointer within the RX buffer.
⇒	<code>UDPSetTxBuffer</code> (see page 527)	Moves the pointer within the TX buffer.
⇒	<code>UDPIsOpened</code> (see page 528)	Determines if a socket has an established connection.

### Macros

	Name	Description
⇒	<code>INVALID_UDP_PORT</code> (see page 518)	Indicates a UDP port that is not valid
⇒	<code>INVALID_UDP_SOCKET</code> (see page 518)	Indicates a UDP socket that is not valid
⇒	<code>UDPOpen</code> (see page 520)	Macro of the legacy version of <code>UDPOpen</code> .
⇒	<code>UDP_OPEN_IP_ADDRESS</code> (see page 528)	Create a client socket and use <code>dwRemoteHost</code> as a literal IP address.
⇒	<code>UDP_OPEN_NODE_INFO</code> (see page 528)	Create a client socket and use <code>dwRemoteHost</code> as a pointer to a <code>NODE_INFO</code> structure containing the exact remote IP address and MAC address to use.

	UDP_OPEN_RAM_HOST ( <a href="#">↗</a> see page 529)	Emit an undeclared identifier diagnostic if code tries to use UDP_OPEN_RAM_HOST while the DNS client module is not enabled.
	UDP_OPEN_ROM_HOST ( <a href="#">↗</a> see page 529)	Emit an undeclared identifier diagnostic if code tries to use UDP_OPEN_ROM_HOST while the DNS client module is not enabled.
	UDP_OPEN_SERVER ( <a href="#">↗</a> see page 529)	Create a server socket and ignore dwRemoteHost.

**Module**

UDP ([↗](#) see page 516)

**Types**

	Name	Description
	UDP_SOCKET ( <a href="#">↗</a> see page 518)	Provides a handle to a UDP Socket

**Description**

The following functions and variables are available to the stack application.

### 10.22.1.1 INVALID\_UDP\_PORT Macro

**File**

UDP.h

**C**

```
#define INVALID_UDP_PORT (0u1) // Indicates a UDP port that is not valid
```

**Description**

Indicates a UDP port that is not valid

### 10.22.1.2 INVALID\_UDP\_SOCKET Macro

**File**

UDP.h

**C**

```
#define INVALID_UDP_SOCKET (0xffu) // Indicates a UDP socket that is not valid
```

**Description**

Indicates a UDP socket that is not valid

### 10.22.1.3 UDP\_SOCKET Type

**File**

UDP.h

**C**

```
typedef BYTE UDP_SOCKET;
```

**Description**

Provides a handle to a UDP Socket

## 10.22.1.4 UDPOpenEx Function

### File

UDP.h

### C

```
UDP_SOCKET UDPOpenEx(
    DWORD remoteHost,
    BYTE remoteHostType,
    UDP_PORT localPort,
    UDP_PORT remotePort
);
```

### Description

Provides a unified method for opening UDP sockets. This function can open both client and server sockets. For client sockets, it can accept (see page 163) a host name string to query in DNS, an IP address as a string, an IP address in binary form, or a previously resolved NODE\_INFO structure containing the remote IP address and associated MAC address. When a host name or IP address only is provided, UDP module will internally perform the necessary DNSResolve (see page 180) and/or ARP resolution steps before reporting that the UDP socket is connected (via a call to UDPIsOpen returning TRUE). Server sockets ignore this destination parameter and listen (see page 169) only on the indicated port. Sockets (see page 146) are statically allocated on boot, but can be claimed with this function and freed using UDPClose (see page 521).

### Remarks

When finished using the UDP socket handle, call the UDPClose (see page 521)() function to free the socket and delete the handle.

### Preconditions

UDPInit (see page 530) should be called.

### Parameters

Parameters	Description
remoteHost	Pointer to remote node info (MAC and IP address) for this connection. If this is a server socket (receives the first packet) or the destination is the broadcast address, then this parameter should be NULL. For client sockets only. Provide a pointer to a null-terminated string of the remote host name (ex:"www.microchip.com" or "192.168.1.123"), a literal destination IP address (ex: 0x7B01A8C0 or an IP_ADDR data type), or a pointer to a NODE_INFO structure with the remote IP address and remote node or gateway MAC address specified, If a string is provided.

remoteHostType	<p>Any one of the following flags to identify the meaning of the remoteHost parameter:</p> <ul style="list-style-type: none"> <li>• UDP_OPEN_SERVER (see page 529) = Open a server socket and ignore the remoteHost parameter. (e.g. - SNMP agent, DHCP server, Announce (see page 149))</li> <li>• UDP_OPEN_IP_ADDRESS (see page 528) = Open a client socket and connect (see page 165) it to a remote IP address. Ex: 0x7B01A8C0 for 192.168.1.123 (DWORD type). Note that the byte ordering is big endian.</li> <li>• UDP_OPEN_NODE_INFO (see page 528) = Open a client socket and connect (see page 165) it to a remote IP and MAC addresses pair stored in a NODE_INFO structure.</li> <li>• UDP_OPEN_RAM_HOST (see page 529) = Open a client socket and connect (see page 165) it to a remote host who's name is stored as a null terminated string in a RAM array. Ex:"www.microchip.com" or "192.168.0.123"</li> <li>• UDP_OPEN_ROM_HOST (see page 529) = Open a client socket and connect (see page 165) it to a remote host who's name is stored as a null terminated string in a literal string or ROM array. Ex: "www.microchip.com" or "192.168.0.123"</li> </ul>
localPort	UDP port number to listen (see page 169) on. If 0, stack will dynamically assign a unique port number to use.
remotePort	For client sockets, the remote port number.

**Return Values**

Return Values	Description
Success	A UDP socket handle that can be used for subsequent UDP API calls.
Failure	INVALID_UDP_SOCKET (see page 518). This function fails when no more UDP socket handles are available. Increase MAX_UDP_SOCKETS to make more sockets available.

## 10.22.1.5 UDPOpen Macro

**File**

UDP.h

**C**

```
#define UDPOpen(localPort, remoteNode, remotePort)
UDPOpenEx( (DWORD)remoteNode, UDP_OPEN_NODE_INFO, localPort, remotePort)
```

**Description**

UDPOpen is a macro replacement of the legacy implementation of UDPOpen. Creates a UDP socket handle for transmitting or receiving UDP packets. Call this function to obtain a handle required by other UDP function.

**Remarks**

When finished using the UDP socket handle, call the UDPClose (see page 521)() function to free the socket and delete the handle.

**Preconditions**

UDPInit (see page 530)() must have been previously called.

**Parameters**

Parameters	Description
localPort	UDP port number to listen (see page 169) on. If 0, stack will dynamically assign a unique port number to use.

remoteNode	Pointer to remote node info (MAC and IP address) for this connection. If this is a server socket (receives the first packet) or the destination is the broadcast address, then this parameter should be NULL.
remotePort	For client sockets, the remote port number.

**Return Values**

Return Values	Description
Success	A UDP socket handle that can be used for subsequent UDP API calls.
Failure	INVALID_UDP_SOCKET (see page 518). This function fails when no more UDP socket handles are available. Increase MAX_UDP_SOCKETS to make more sockets available.

## 10.22.1.6 UDPClose Function

**File**

UDP.h

**C**

```
void UDPClose(
    UDP_SOCKET s
);
```

**Returns**

None

**Description**

UDP\_SOCKET (see page 518) UDPOpen (see page 520)(UDP\_PORT (see page 534) localPort, NODE\_INFO \*remoteNode, UDP\_PORT (see page 534) remotePort);

Closes a UDP socket and frees the handle. Call this function to release a socket and return it to the pool for use by future communications.

**Remarks**

This function does not affect the previously designated active socket.

**Preconditions**

UDPInit (see page 530)() must have been previously called.

**Parameters**

Parameters	Description
s	The socket handle to be released. If an illegal handle value is provided, the function safely does nothing.

## 10.22.1.7 UDPDiscard Function

**File**

UDP.h

**C**

```
void UDPDiscard();
```

**Returns**

None

**Description**

This function discards any remaining received data in the currently active UDP socket.

**Remarks**

It is safe to call this function more than is necessary. If no data is available, this function does nothing.

**Preconditions**

UDPIsGetReady (see page 523)() was previously called to select the currently active socket.

## 10.22.1.8 UDPFlush Function

**File**

UDP.h

**C**

```
void UDPFlush();
```

**Returns**

None

**Description**

This function builds a UDP packet with the pending TX data and marks it for transmission over the network interface. Since UDP is a frame-based protocol, this function must be called before returning to the main stack loop whenever any data is written.

**Remarks**

Note that unlike TCPFlush (see page 446), UDPFlush must be called before returning to the main stack loop. There is no auto transmit for UDP segments.

**Preconditions**

UDPIsPutReady (see page 524)() was previously called to specify the current socket, and data has been written to the socket using the UDPPut (see page 524) family of functions.

## 10.22.1.9 UDPGet Function

**File**

UDP.h

**C**

```
BOOL UDPGet(
    BYTE * v
);
```

**Description**

This function reads a single byte from the currently active UDP socket, while decrementing the remaining buffer length. UDPIsGetReady (see page 523) should be used before calling this function to specify the currently active socket.

**Preconditions**

UDPIsGetReady (see page 523)() was previously called to specify the current socket.

**Parameters**

Parameters	Description
v	The buffer to receive the data being read.

**Return Values**

Return Values	Description
TRUE	A byte was successfully read

FALSE	No data remained in the read buffer
-------	-------------------------------------

### 10.22.1.10 UDPGetArray Function

**File**

UDP.h

**C**

```
WORD UDPGetArray(
    BYTE * cData,
    WORD wDataLen
);
```

**Returns**

The number of bytes successfully read from the UDP buffer. If this value is less than wDataLen, then the buffer was emptied and no more data is available.

**Description**

This function reads an array of bytes from the currently active UDP socket, while decrementing the remaining bytes available. UDPsGetReady (see page 523) should be used before calling this function to specify the currently active socket.

**Preconditions**

UDPsGetReady (see page 523)() was previously called to specify the current socket.

**Parameters**

Parameters	Description
cData	The buffer to receive the bytes being read. If NULL, the bytes are simply discarded without being written anywhere (effectively skips over the bytes in the RX buffer, although if you need to skip a lot of data, seeking using the UDPSetRxBuffer (see page 527)() will be more efficient).
wDataLen	Number of bytes to be read from the socket.

### 10.22.1.11 UDPsGetReady Function

**File**

UDP.h

**C**

```
WORD UDPsGetReady(
    UDP_SOCKET s
);
```

**Returns**

The number of bytes that can be read from this socket.

**Description**

This function determines if bytes can be read from the specified UDP socket. It also prepares the UDP module for reading by setting the indicated socket as the currently active connection.

**Preconditions**

UDPInit (see page 530)() must have been previously called.

**Parameters**

Parameters	Description
s	The socket to be made active (which has already been opened or is listening)

## 10.22.1.12 UDPIsPutReady Function

### File

UDP.h

### C

```
WORD UDPIsPutReady(
    UDP_SOCKET s
);
```

### Returns

The number of bytes that can be written to this socket.

### Description

This function determines if bytes can be written to the specified UDP socket. It also prepares the UDP module for writing by setting the indicated socket as the currently active connection.

### Preconditions

UDPIInit (see page 530)() must have been previously called.

### Parameters

Parameters	Description
s	The socket to be made active

## 10.22.1.13 UDPPut Function

### File

UDP.h

### C

```
BOOL UDPPut(
    BYTE v
);
```

### Description

This function writes a single byte to the currently active UDP socket, while incrementing the buffer length. UDPIsPutReady (see page 524) should be used before calling this function to specify the currently active socket.

### Preconditions

UDPIsPutReady (see page 524)() was previously called to specify the current socket.

### Parameters

Parameters	Description
v	The byte to be loaded into the transmit buffer.

### Return Values

Return Values	Description
TRUE	The byte was successfully written to the socket.
FALSE	The transmit buffer is already full and so the write failed.

## 10.22.1.14 UDPPutArray Function

### File

UDP.h

### C

```
WORD UDPPutArray(
    BYTE * cData,
    WORD wDataLen
);
```

### Returns

The number of bytes successfully placed in the UDP transmit buffer. If this value is less than wDataLen, then the buffer became full and the input was truncated.

### Description

This function writes an array of bytes to the currently active UDP socket, while incrementing the buffer length. UDPIsPutReady (see page 524) should be used before calling this function to specify the currently active socket.

### Preconditions

UDPIsPutReady (see page 524)() was previously called to specify the current socket.

### Parameters

Parameters	Description
cData	The array to write to the socket.
wDataLen	Number of bytes from cData to be written.

## 10.22.1.15 UDPPutROMArray Function

### File

UDP.h

### C

```
WORD UDPPutROMArray(
    ROM BYTE * cData,
    WORD wDataLen
);
```

### Returns

The number of bytes successfully placed in the UDP transmit buffer. If this value is less than wDataLen, then the buffer became full and the input was truncated.

### Description

ROM function variants for PIC18

This function writes an array of bytes from ROM to the currently active UDP socket, while incrementing the buffer length. UDPIsPutReady (see page 524) should be used before calling this function to specify the currently active socket.

### Remarks

This function is aliased to UDPPutArray (see page 525) on non-PIC18 platforms.

### Preconditions

UDPIsPutReady (see page 524)() was previously called to specify the current socket.

**Parameters**

Parameters	Description
cData	The array to write to the socket.
wDateLen	Number of bytes from cData to be written.

## 10.22.1.16 UDPPutROMString Function

**File**

UDP.h

**C**

```
ROM BYTE* UDPPutROMString(
    ROM BYTE * strData
);
```

**Returns**

A pointer to the byte following the last byte written. Note that this is different than the UDPPutArray (see page 525) functions. If this pointer does not dereference to a NULL byte, then the buffer became full and the input data was truncated.

**Description**

This function writes a null-terminated string from ROM to the currently active UDP socket, while incrementing the buffer length. UDPIsPutReady (see page 524) should be used before calling this function to specify the currently active socket.

**Remarks**

This function is aliased to UDPPutString (see page 526) on non-PIC18 platforms.

**Preconditions**

UDPIsPutReady (see page 524)() was previously called to specify the current socket.

**Parameters**

Parameters	Description
cData	Pointer to the string to be written to the socket.

## 10.22.1.17 UDPPutString Function

**File**

UDP.h

**C**

```
BYTE* UDPPutString(
    BYTE * strData
);
```

**Returns**

A pointer to the byte following the last byte written. Note that this is different than the UDPPutArray (see page 525) functions. If this pointer does not dereference to a NULL byte, then the buffer became full and the input data was truncated.

**Description**

This function writes a null-terminated string to the currently active UDP socket, while incrementing the buffer length. UDPIsPutReady (see page 524) should be used before calling this function to specify the currently active socket.

**Preconditions**

UDPIsPutReady (see page 524)() was previously called to specify the current socket.

**Parameters**

Parameters	Description
cData	Pointer to the string to be written to the socket.

**10.22.1.18 UDPSetRxBuffer Function****File**

UDP.h

**C**

```
void UDPSetRxBuffer(
    WORD wOffset
);
```

**Returns**

None

**Description**

This function allows the read location within the RX buffer to be specified. Future calls to [UDPGet](#) (see page 522) and [UDPGetArray](#) (see page 523) will read data from the indicated location forward.

**Preconditions**

[UDPInit](#) (see page 530)() must have been previously called and a socket is currently active.

**Parameters**

Parameters	Description
wOffset	Offset from beginning of UDP packet data payload to place the read pointer.

**10.22.1.19 UDPSetTxBuffer Function****File**

UDP.h

**C**

```
void UDPSetTxBuffer(
    WORD wOffset
);
```

**Returns**

None

**Description**

This function allows the write location within the TX buffer to be specified. Future calls to [UDPPut](#) (see page 524), [UDPPutArray](#) (see page 525), [UDPPutString](#) (see page 526), etc will write data from the indicated location.

**Preconditions**

[UDPInit](#) (see page 530)() must have been previously called and a socket is currently active.

**Parameters**

Parameters	Description
wOffset	Offset from beginning of UDP packet data payload to place the write pointer.

## 10.22.1.20 UDPIsOpened Function

### File

UDP.h

### C

```

BOOL UDPIsOpened(
    UDP_SOCKET socket
);

```

### Description

This function determines if a socket has an established connection to a remote node . Call this function after calling UDPOpen (see page 520) to determine when the connection is set up and ready for use.

### Remarks

None

### Preconditions

UDP is initialized.

### Parameters

Parameters	Description
socket (see page 174)	The socket to check.

### Return Values

Return Values	Description
TRUE	The socket has been opened and ARP has been resolved.
FALSE	The socket is not currently connected.

## 10.22.1.21 UDP\_OPEN\_IP\_ADDRESS Macro

### File

UDP.h

### C

```
#define UDP_OPEN_IP_ADDRESS 3u
```

### Description

Create a client socket and use dwRemoteHost as a literal IP address.

## 10.22.1.22 UDP\_OPEN\_NODE\_INFO Macro

### File

UDP.h

### C

```
#define UDP_OPEN_NODE_INFO 4u
```

### Description

Create a client socket and use dwRemoteHost as a pointer to a NODE\_INFO structure containing the exact remote IP address and MAC address to use.

### 10.22.1.23 UDP\_OPEN\_RAM\_HOST Macro

**File**

UDP.h

**C**

```
#define UDP_OPEN_RAM_HOST You_need_to_enable_STACK_USE_DNS_to_use_UDP_OPEN_RAM_HOST
```

**Description**

Emit an undeclared identifier diagnostic if code tries to use UDP\_OPEN\_RAM\_HOST while the DNS client module is not enabled.

### 10.22.1.24 UDP\_OPEN\_ROM\_HOST Macro

**File**

UDP.h

**C**

```
#define UDP_OPEN_ROM_HOST You_need_to_enable_STACK_USE_DNS_to_use_UDP_OPEN_ROM_HOST
```

**Description**

Emit an undeclared identifier diagnostic if code tries to use UDP\_OPEN\_ROM\_HOST while the DNS client module is not enabled.

### 10.22.1.25 UDP\_OPEN\_SERVER Macro

**File**

UDP.h

**C**

```
#define UDP_OPEN_SERVER 0u
```

**Description**

Create a server socket and ignore dwRemoteHost.

## 10.22.2 UDP Stack Members

**Functions**

	Name	Description
☞	UDPInit (see page 530)	Initializes the UDP module.
☞	UDPProcess (see page 530)	Handles an incoming UDP segment.
☞	UDPTask (see page 531)	Performs state management and housekeeping for UDP.

**Module**

UDP (see page 516)

**Description**

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

## 10.22.2.1 UDPInit Function

### File

UDP.h

### C

```
void UDPInit();
```

### Returns

None

### Description

Initializes the UDP module. This function initializes all the UDP sockets to the closed state.

### Remarks

This function is called only one during lifetime of the application.

### Preconditions

None

### Section

Function Prototypes

## 10.22.2.2 UDPProcess Function

### File

UDP.h

### C

```
BOOL UDPProcess(
    NODE_INFO * remoteNode,
    IP_ADDR * localIP,
    WORD len
);
```

### Description

This function handles an incoming UDP segment to determine if it is acceptable and should be handed to one of the stack applications for processing.

### Preconditions

UDPInit (see page 530)() has been called and a UDP segment is ready in the MAC buffer.

### Parameters

Parameters	Description
remoteNode	The remote node that sent this segment.
localIP	The destination IP address for this segment.
len	Total length of the UDP segment.

### Return Values

Return Values	Description
TRUE	A valid packet is waiting and the stack applications should be called to handle it.
FALSE	The packet was discarded.

### 10.22.2.3 UDPTask Function

**File**

UDP.h

**C**

```
void UDPTask( );
```

**Description**

Performs state management and housekeeping for UDP. This is an internal function meant to be called by StackTask() (not a user API).

**Remarks**

UDPTask() is called once per StackTask() iteration to ensure that calls to UDPisPutReady (see page 524)() always update the Ethernet Write pointer location between StackTask() iterations.

**Preconditions**

None

## 10.22.3 UDP Internal Members

**Functions**

	Name	Description
	FindMatchingSocket (see page 532)	Matches an incoming UDP segment to a currently active socket.

**Macros**

	Name	Description
	LOCAL_UDP_PORT_END_NUMBER (see page 533)	Last port number for randomized local port number selection
	LOCAL_UDP_PORT_START_NUMBER (see page 533)	First port number for randomized local port number selection

**Module**

UDP (see page 516)

**Structures**

	Name	Description
	UDP_HEADER (see page 534)	Stores the header of a UDP packet
	UDP_SOCKET_INFO (see page 534)	Stores information about a current UDP socket

**Types**

	Name	Description
	UDP_PORT (see page 534)	Stores a UDP Port Number

**Variables**

	Name	Description
	activeUDPSocket (see page 532)	Indicates which UDP socket is currently active

	LastPutSocket ( <a href="#">see page 533</a> )	Indicates the last socket to which data was written
	SocketWithRxData ( <a href="#">see page 533</a> )	Indicates which socket has currently received data for this loop
	UDPRxCount ( <a href="#">see page 535</a> )	Number of bytes read from this UDP segment
	UDPSocketInfo ( <a href="#">see page 535</a> )	Stores an array of information pertaining to each UDP socket
	UDPTxCount ( <a href="#">see page 535</a> )	Number of bytes written to this UDP segment
	wGetOffset ( <a href="#">see page 535</a> )	Offset from beginning of payload from where data is to be read.
	wPutOffset ( <a href="#">see page 536</a> )	Offset from beginning of payload where data is to be written.

**Description**

The following functions and variables are designated as internal to the UDP module.

### 10.22.3.1 activeUDPSocket Variable

**File**

UDP.c

**C**

```
UDP_SOCKET activeUDPSocket;
```

**Description**

Indicates which UDP socket is currently active

### 10.22.3.2 FindMatchingSocket Function

**File**

UDP.c

**C**

```
static UDP_SOCKET FindMatchingSocket(
    UDP_HEADER * h,
    NODE_INFO * remoteNode,
    IP_ADDR * localIP
);
```

**Returns**

A UDP\_SOCKET ([see page 518](#)) handle of a matching socket, or INVALID\_UDP\_SOCKET ([see page 518](#)) when no match could be made.

**Description**

This function attempts to match an incoming UDP segment to a currently active socket for processing.

**Preconditions**

UDP segment header and IP header have both been retrieved.

**Parameters**

Parameters	Description
h	The UDP header that was received.
remoteNode	IP and MAC of the remote node that sent this segment.
localIP	IP address that this segment was destined for.

**Section**

Function Prototypes

### 10.22.3.3 LastPutSocket Variable

**File**

UDP.c

**C**

```
UDP_SOCKET LastPutSocket = INVALID_UDP_SOCKET;
```

**Description**

Indicates the last socket to which data was written

### 10.22.3.4 LOCAL\_UDP\_PORT\_END\_NUMBER Macro

**File**

UDP.c

**C**

```
#define LOCAL_UDP_PORT_END_NUMBER (8192u)
```

**Description**

Last port number for randomized local port number selection

### 10.22.3.5 LOCAL\_UDP\_PORT\_START\_NUMBER Macro

**File**

UDP.c

**C**

```
#define LOCAL_UDP_PORT_START_NUMBER (4096u)
```

**Description**

First port number for randomized local port number selection

### 10.22.3.6 SocketWithRxData Variable

**File**

UDP.c

**C**

```
UDP_SOCKET SocketWithRxData = INVALID_UDP_SOCKET;
```

**Description**

Indicates which socket has currently received data for this loop

## 10.22.3.7 UDP\_HEADER Structure

### File

UDP.h

### C

```
typedef struct {
    UDP_PORT SourcePort;
    UDP_PORT DestinationPort;
    WORD Length;
    WORD Checksum;
} UDP_HEADER;
```

### Members

Members	Description
UDP_PORT SourcePort;	Source UDP port
UDP_PORT DestinationPort;	Destination UDP port
WORD Length;	Length of data
WORD Checksum;	UDP checksum of the data

### Description

Stores the header of a UDP packet

## 10.22.3.8 UDP\_PORT Type

### File

UDP.h

### C

```
typedef WORD UDP_PORT;
```

### Description

Stores a UDP Port Number

## 10.22.3.9 UDP\_SOCKET\_INFO Structure

### File

UDP.h

### C

```
typedef struct {
    union {
        NODE_INFO remoteNode;
        DWORD remoteHost;
    } remote;
    UDP_PORT remotePort;
    UDP_PORT localPort;
    UDP_STATE smState;
    DWORD retryInterval;
    BYTE retryCount;
    struct {
        unsigned char bRemoteHostIsROM : 1;
    } flags;
    WORD eventTime;
} UDP_SOCKET_INFO;
```

**Members**

Members	Description
NODE_INFO remoteNode;	10 bytes for MAC and IP address
DWORD remoteHost;	RAM or ROM pointer to a hostname string (ex: "www.microchip.com")
UDP_PORT remotePort;	Remote node's UDP port number
UDP_PORT localPort;	Local UDP port number, or INVALID_UDP_PORT (see page 518) when free
UDP_STATE smState;	State of this socket
unsigned char bRemoteHostIsROM : 1;	Remote host is stored in ROM

**Description**

Stores information about a current UDP socket

## 10.22.3.10 UDPRxCount Variable

**File**

UDP.c

**C**

```
WORD UDPRxCount ;
```

**Description**

Number of bytes read from this UDP segment

## 10.22.3.11 UDPSocketInfo Variable

**File**

UDP.c

**C**

```
UDP_SOCKET_INFO UDPSocketInfo[MAX_UDP_SOCKETS];
```

**Description**

Stores an array of information pertaining to each UDP socket

## 10.22.3.12 UDPTxCount Variable

**File**

UDP.c

**C**

```
WORD UDPTxCount ;
```

**Description**

Number of bytes written to this UDP segment

## 10.22.3.13 wGetOffset Variable

**File**

UDP.c

**C**WORD `wGetOffset` ;**Description**

Offset from beginning of payload from where data is to be read.

## 10.22.3.14 wPutOffset Variable

**File**

UDP.c

**C**WORD `wPutOffset` ;**Description**

Offset from beginning of payload where data is to be written.

## 10.22.4 Types

**Enumerations**

	Name	Description
	UDP_STATE ( <a href="#">see page 536</a> )	UDP States

**Module**UDP ([see page 516](#))

### 10.22.4.1 UDP\_STATE Enumeration

**File**

UDP.h

**C**

```
typedef enum {
    UDP_DNS_IS_RESOLVED,
    UDP_DNS_RESOLVE,
    UDP_GATEWAY_SEND_ARP,
    UDP_GATEWAY_GET_ARP,
    UDP_CLOSED,
    UDP_OPENED
} UDP_STATE;
```

**Members**

Members	Description
UDP_DNS_IS_RESOLVED	Special state for UDP client mode sockets
UDP_DNS_RESOLVE	Special state for UDP client mode sockets
UDP_GATEWAY_SEND_ARP	Special state for UDP client mode sockets
UDP_GATEWAY_GET_ARP	Special state for UDP client mode sockets
UDP_CLOSED	Socket is idle and unallocated

**Description**

UDP States

# 11 Wi-Fi API

## Modules

Name	Description
Wi-Fi Connection Profile (see page 540)	Functions to setup, use, and teardown connection profiles
Wi-Fi Connection Algorithm (see page 554)	Functions to alter the behavior of the connection process
Wi-Fi Connection Manager (see page 576)	Functions to manage the connection process
Wi-Fi Scan (see page 578)	Functions to direct the MRF24WB0M to initiate a site survey
Wi-Fi Tx Power Control (see page 580)	API to control the Tx power of the MRF24WB0M
Wi-Fi Power Save (see page 582)	Functions to alter the power savings features of the MRF24WB0M
Wi-Fi Miscellaneous (see page 587)	Functions for controlling miscellaneous features of the MRF24WB0M

## Description

Unlike Ethernet, a WiFi application needs to initiate a connection to an access point or an ad hoc network) before data communications can commence. In order to initiate an connection there is a sequence of steps that should be followed.

1) A connection profile must be created (see `WF_CPCreate` (see page 541)()). The connection profile contains information directing the WiFi driver about the nature of the connection that will be established. The connection profile defines:

- a. SSID (name of Access Point)
- b. Security (open, WEP, WPA, etc.)
- c. Network type (infrastructure or ad hoc).

The Connection Profile functions are used to create and define an connection profile. These functions all begin with `WF_CP...`

2) The connection algorithm must be defined, and applies to all connection profiles. For most applications the defaults will be sufficient. For example, the default connection algorithm channel list for scanning is 1, 6, and 11. However, if, in your application you know the Access Point will always be on channel 6 you could change this setting, thus making the scan process more efficient. Functions pertaining to the connection algorithm all begin with `WF_CA...`

3) Once a connection profile and the connection algorithm are customized for an application, the `WF_CMConnect` (see page 576)() function must be called to initiate the connection process.

4) After `WF_Connect()` is called the host application will be notified when the MRF24WB0M has succeeded (or failed) in establishing a connection via the event mechanism. The `WF_Config.c` file has a function, `WF_ProcessEvent` (see page 595)(), that is a template for processing MRF24WB0M events. In the WiFi demos it simply prints to the console (if the UART is enabled) that the event occurred. This file can be modified to suit the needs of an application – for example, an application could pend on a global flag that would be set in `WF_ProcessEvent` (see page 595)() when the connection succeeded. Please refer to `WF_ProcessEvent` (see page 595) for more information on WiFi event handling.

The MRF2WB0M demos (under the Demo App, WiFi Console, and WiFi EZ Config demo directories) contain a function, `WF_Connect()`, in `MainDemo.c` that executes the above steps and can be referred to as an example of how to initiate a WiFi

connection. The `WF_Config.h` file has several compile-time constants that can be customized (e.g. `MY_DEFAULT_SSID_NAME`) as needed.

This help file book describes the host API to the MRF24WB0M on-chip connection manager which creates and maintains Wi-Fi connections. The API is divided into these major sections:

API Section	Description
Initialization (see page 132)	Functions to initialize the host API and MRF24WB0M
Connection Profile	Functions to create and maintain one or more connection profiles
Connection Algorithm	Functions to fine tune the connection algorithm
Connection Manager	Functions to start and stop an 802.11 connection
Scan	Functions to scan for wireless networks
Tx Power Control	Functions to control the MRF24WB0M Tx power
Power Save	Functions to save power consumption by the MRF24WB0M
Multicast	Functions to create multicast filters
Miscellaneous	Functions to set a custom MAC address, get device information, etc.
MRF24WB0M Events	Functions to handle events from the MRF24WB0M

## SPI

The `WF_Spi.c` file contains functions that the Wi-Fi Driver will use to initialize, send, and receive SPI messages between the host CPU and the MRF24WB0M. To communicate with the MRF24WB0M, which is always an SPI slave, the host CPU SPI controller needs to be configured as follows:

- Mode = 0
- CPOL (clock polarity) = 0
- CPHA (clock phase) = 0
- Host CPU set as master
- Clock idles high
- 8-bit transfer length
- Data changes on falling edge
- Data sampled on rising edge

Below is a list of functions in `WF_Spi.c` that must be customized for the specific host CPU architecture:

Function	Description
<code>WF_SpiInit()</code>	Initializes the host CPU SPI controller for usage by the Wi-Fi driver. Called by the Wi-Fi driver during initialization.
<code>WF_SpiTxRx()</code>	Transmits and/or receives SPI data from the MRF24WB0M.
<code>WF_SpiEnableChipSelect()</code>	Set slave select line on MRF24WB0M low (start SPI transfer).  If SPI bus is shared with any other devices then this function also needs to save the current SPI context and then configure the MRF24WB0M SPI context.
<code>WF_SpiDisableChipSelect()</code>	Set slave select line on MRF24WB0M high (end SPI transfer).  If SPI bus is shared with any other devices then this function also needs to restore the SPI context (saved during <code>WF_SpiEnableChipSelect()</code> ).

## External Interrupt

The `WF_Eint.c` file contains functions that the Wi-Fi Driver will use to enable and disable the MRF24WB0M external interrupt as well as get interrupt status. The functions in this module need to be customized for the specific host CPU architecture.

The MRF24WB0M asserts its EXINT (external interrupt) line (active low) when specific events occur, such as a data message being received. Note that the host CPU has a choice to either configure the EXINT line to generate an actual interrupt, or, it can be polled. Below is a list of the Wi-Fi Driver functions within `WF_Eint.c` that must be customized for the specific Host CPU architecture.

Function	Description
<code>WF_EintInit()</code>	Configures the interrupt for use and leaves it in a disabled state. Will be called by the Wi-Fi driver during initialization. If polling the EXINT pin then this function won't have any work to do except leave the interrupt in a logically disabled state.
<code>WF_EintEnable()</code>	Enables the MRF24WB0M external interrupt. If using real interrupts then enable the interrupt. If polling the EXINT pin then this function enables polling of the pin.
<code>WF_EintDisable()</code>	Disables the MRF24WB0M external interrupt. If using real interrupts then disable the interrupt. If polling the EXINT pin then this function disables polling of the pin.
<code>WF_EintIsr()</code>	This is the interrupt service routine invoked when the EXINT line goes low. It should perform any necessary housekeeping, such as clearing the interrupt. The interrupt must remain disabled until the Wi-Fi Driver calls <code>WF_EintEnable()</code> . The Wi-Fi driver function, <code>WFEintHandler()</code> must be called.
<code>WF_EintIsDisabled()</code>	Returns true if the external interrupt is disabled, else returns false.
<code>WFEintHandler()</code>	This function does not need to be customized – it is part of the Wi-Fi driver. However, it is added to this list because it must be called each time the MRF24WB0M interrupt service routine (ISR) occurs.

## WF\_Config

The `WF_Config` module (`WF_Config.h/WF_Config.c`) is used to control several aspects of the WiFi Driver behavior. Most of the customization of the Wi-Fi module is done from the context of this module.

## Removal of Unused Driver Functions

In `WF_Customize.h` there is a block of defines that can be commented out to remove those sections of the Wi-Fi host driver that are not needed by the application. This allows the saving of code and data space.

#define	Controlling Functions
<code>WF_USE_SCAN_FUNCTIONS</code>	Scan API
<code>WF_USE_TX_POWER_CONTROL_FUNCTIONS</code>	Tx power control API
<code>WF_USE_POWER_SAVE_FUNCTIONS</code>	Power save API
<code>WF_USE_MULTICAST_FUNCTIONS</code>	Multicast API
<code>WF_USE_INDIVIDUAL_SET_GETS</code>	Affects all get and set functions, except the following: <code>WF_CPSetElements (see page 549)()</code> <code>WF_CPGetElements (see page 544)()</code> <code>WF_CASetElements (see page 566)()</code> <code>WF_CAGetElements (see page 558)()</code>
<code>WF_USE_GROUP_SET_GETS</code>	Affects the following functions: <code>WF_CPSetElements (see page 549)()</code> <code>WF_CPGetElements (see page 544)()</code> <code>WF_CASetElements (see page 566)()</code> <code>WF_CAGetElements (see page 558)()</code>

**WF\_DEBUG**

This define enables the `WF_ASSERT` macro in the Wi-Fi driver. Customer code is free to use this macro. The `WF_ASSERT` macro can be compiled in or out via the `WF_DEBUG` define. See the comment above the `WF_DEBUG` define in `WF_Customize.h` for details.

**WF\_CONSOLE**

The Wi-Fi driver has a UART console application built in that allows one to type in command lines and has them parsed. If this functionality is not needed than it can be compiled out by commenting out the `WF_CONSOLE` define.

**WF\_ProcessEvent()**

This function is called by the Wi-Fi Driver when an event occurs that the host CPU needs to be notified of. There are several Wi-Fi connection related events that the application can choose whether to be notified or not. And, there are several events the application will always be notified of.

The function `WF_ProcessEvent` (see page 595)() can be customized to support desired handling of events.

---

## 11.1 Wi-Fi Connection Profile

**Module**

Wi-Fi API (see page 537)

**Description**

This section describes the API functions related to creating and using connection profiles. At least one connection profile must be created. The connection profile defines elements required by the MRF24WB0M to establish a connection to a Wi-Fi network.

**Modifying Connection Profile after Connection is Established**

A connection profile can be updated while it is being used for an active connection. However, the updates do not take effect until one of the following occurs:

- Connection is disabled and re-enabled by the host application
- Connection algorithm loses the connection, exhausts all retries, and then reloads the connection profile.

To ensure that connection profile updates take place at a known point in time it is recommended that the host application call `WF_CMDDisconnect` (see page 577)(), update the connection profile, then call `WF_CMConnect` (see page 576)().

---

### 11.1.1 Connection Profile Public Members

**Functions**

	Name	Description
☰	<code>WF_CPCreate</code> (see page 541)	Creates a Connection Profile on the MRF24WB0M.
☰	<code>WF_CPDelete</code> (see page 542)	Deletes a Connection Profile on the MRF24WB0M.
☰	<code>WF_CPGetAdHocBehavior</code> (see page 542)	Gets the desired Ad Hoc behavior

	WF_CPGetBssid ( <a href="#">see page 543</a> )	Gets the BSSID for the specified Connection Profile ID.
	WF_CPGetDefaultWepKeyIndex ( <a href="#">see page 543</a> )	Gets the value of the active WEP keys to use.
	WF_CPGetElements ( <a href="#">see page 544</a> )	Reads the Connection Profile elements for the specified ID.
	WF_CPGetIds ( <a href="#">see page 544</a> )	Retrieves the CP ID bit mask.
	WF_CPGetNetworkType ( <a href="#">see page 545</a> )	Gets the network for the specified Connection Profile ID.
	WF_CPGetSecurity ( <a href="#">see page 546</a> )	Gets the security for the specified Connection Profile.
	WF_CPGetSsid ( <a href="#">see page 547</a> )	Gets the SSID for the specified Connection Profile ID.
	WF_CPSetAdHocBehavior ( <a href="#">see page 547</a> )	Selects the desired Ad Hoc behavior
	WF_CPSetBssid ( <a href="#">see page 548</a> )	Sets the BSSID for the specified Connection Profile ID.
	WF_CPSetDefaultWepKeyIndex ( <a href="#">see page 548</a> )	Selects one of the 4 WEP keys to use.
	WF_CPSetElements ( <a href="#">see page 549</a> )	Writes out data for a specific connection profile element.
	WF_CPSetNetworkType ( <a href="#">see page 549</a> )	Sets the network for the specified Connection Profile ID.
	WF_CPSetSecurity ( <a href="#">see page 550</a> )	Sets the security for the specified Connection Profile.
	WF_CPSetSsid ( <a href="#">see page 551</a> )	Sets the SSID for the specified Connection Profile ID.

**Module**

Wi-Fi Connection Profile ([see page 540](#))

**Structures**

	Name	Description
	WFCPElementsStruct ( <a href="#">see page 551</a> )	Connection profile elements structure

**Description**

## 11.1.1.1 WF\_CPCreate Function

**File**

WFApi.h

**C**

```
void WF_CPCreate(
    UINT8 * p_CpId
);
```

**Returns**

None.

**Description**

Connection Profile Functions

Requests the MRF24WB0M to create a Connection Profile (CP), assign it an ID, and set all the elements to default values.

The ID returned by this function is used in other connection profile functions. A maximum of 2 Connection Profiles can exist on the MRF24WB0M.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_CpId	Pointer to where Connection Profile ID will be written. If function fails, the CP ID will be set to 0xff.

## 11.1.1.2 WF\_CPDelete Function

**File**

WFApi.h

**C**

```
void WF_CPDelete(
    UINT8 CpId
);
```

**Returns**

None.

**Description**

Deletes the specified Connection Profile. If the Connection Profile was in FLASH it will be erased from FLASH.

**Remarks**

First release of this code will not support FLASH, only the two CP's in memory.

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	Connection Profile to delete test case.

## 11.1.1.3 WF\_CPGetAdHocBehavior Function

**File**

WFApi.h

**C**

```
void WF_CPGetAdHocBehavior(
    UINT8 CpId,
    UINT8 * p_adHocBehavior
);
```

**Returns**

None.

**Description**

Gets the AdHoc behavior within a Connection Profile. Allowable values are:

- WF\_ADHOC\_CONNECT\_THEN\_START
- WF\_ADHOC\_CONNECT\_ONLY
- WF\_ADHOC\_START\_ONLY

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	Connection Profile ID
adhocBehavior	Pointer to location of the adhoc behavior value for this connection profile.

## 11.1.1.4 WF\_CPGetBssid Function

**File**

WFApi.h

**C**

```
void WF_CPGetBssid(
    UINT8 CpId,
    UINT8 * p_bssid
);
```

**Returns**

None.

**Description**

Gets the BSSID element in a Connection Profile.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	Connection Profile ID
p_bssid	Pointer to the BSSID

## 11.1.1.5 WF\_CPGetDefaultWepKeyIndex Function

**File**

WFApi.h

**C**

```
void WF_CPGetDefaultWepKeyIndex(
    UINT8 CpId,
    UINT8 * p_defaultWepKeyIndex
);
```

```
);
```

**Returns**

None.

**Description**

Only applicable if the Connection Profile security type is either WF\_SECURITY\_WEP\_40 or WF\_SECURITY\_WEP\_104. Selects which of the four WEP keys to use.

**Remarks**

Note that only key 0 amongst AP manufacturers is typically used. Using any of the other three keys may be unpredictable from brand to brand.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	Connection Profile ID
p_defaultWepKeyIndex	Pointer to index of WEP key to use (0 - 3)

## 11.1.1.6 WF\_CPGetElements Function

**File**

WFApi.h

**C**

```
void WF_CPGetElements(
    UINT8 CpId,
    tWFCPElements * p_elements
);
```

**Returns**

None.

**Description**

Gets all Connection Profile elements for the specified CP ID. If the Host CPU does not have enough memory to create a structure of this size then call the individual get functions.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	Connectino Profile ID.
p_elements	Pointer to Connection Profile elements structure.

## 11.1.1.7 WF\_CPGetIds Function

**File**

WFApi.h

**C**

```
void WF_CPGetIds(
    UINT8 * cpIdList
);
```

**Returns**

None.

**Description**

Returns a list of all Connection Profile ID's that have been created on the MRF24WB0M. This is not to be confused with the Connection Algorithm's connectionProfileList. This function returns a bit mask corresponding to a list of all Connection Profiles that have been created (whether they are in the connectionProfileList or not). Any Connection Profiles that have been saved to FLASH will be included.

**Remarks**

the first release will only support two Connection Profiles in memory. Saving CP's to FLASH will not be supported.

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_cpIdList	Pointer to value representing the bit mask where each bit index (plus 1) corresponds to a Connection Profile ID that has been created. For example, if this value is 0x03, then Connection Profile ID's 1 and 2 have been created.

### 11.1.1.8 WF\_CPGetNetworkType Function

**File**

WFApi.h

**C**

```
void WF_CPGetNetworkType(
    UINT8 CpId,
    UINT8 * p_networkType
);
```

**Returns**

None.

**Description**

Gets the Network Type element a Connection Profile. Allowable values are:

- WF\_INFRASTRUCTURE
- WF\_ADHOC

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	Connection Profile ID

networkType	Type of network to create (infrastructure or adhoc)
-------------	---

### 11.1.1.9 WF\_CPGetSecurity Function

#### File

WFApi.h

#### C

```
void WF_CPGetSecurity(
    UINT8 CpId,
    UINT8 * p_securityType,
    UINT8 * p_wepKeyIndex,
    UINT8 * p_securityKey,
    UINT8 * p_securityKeyLength
);
```

#### Returns

None.

#### Description

Configures security for a Connection Profile.

Security	Key	Length
WF_SECURITY_OPEN	N/A	N/A
WF_SECURITY_WEP_40	hex	4, 5 byte keys
WF_SECURITY_WEP_104	hex	4, 13 byte keys
WF_SECURITY_WPA_WITH_KEY	hex	32 bytes
WF_SECURITY_WPA_WITH_PASS_PHRASE	ascii	8-63 ascii characters
WF_SECURITY_WPA2_WITH_KEY	hex	32 bytes
WF_SECURITY_WPA2_WITH_PASS_PHRASE	ascii	8-63 ascii characters
WF_SECURITY_WPA_AUTO_WITH_KEY	hex	32 bytes
WF_SECURITY_WPA_AUTO_WITH_PASS_PHRASE	ascii	8-63 ascii characters

#### Remarks

None.

#### Preconditions

MACInit must be called first.

#### Parameters

Parameters	Description
CpId	Connection Profile ID
securityType	Value corresponding to the security type desired.
wepKeyIndex	0 thru 3 (only used if security type is WF_SECURITY_WEP_40 or WF_SECURITY_WEP_104)
p_securityKey	Binary key or passphrase (not used if security is WF_SECURITY_OPEN)
securityKeyLength	Number of bytes in p_securityKey (not used if security is WF_SECURITY_OPEN)

## 11.1.1.10 WF\_CPGetSsid Function

### File

WFApi.h

### C

```
void WF_CPGetSsid(
    UINT8 CpId,
    UINT8 * p_ssid,
    UINT8 * p_ssidLength
);
```

### Returns

None.

### Description

Gets the SSID and SSID Length elements in the Connection Profile.

### Remarks

None.

### Preconditions

MACInit must be called first.

### Parameters

Parameters	Description
CpId	Connection Profile ID
p_ssid	Pointer to the SSID string
ssidLength	Number of bytes in the SSID

## 11.1.1.11 WF\_CPSetAdHocBehavior Function

### File

WFApi.h

### C

```
void WF_CPSetAdHocBehavior(
    UINT8 CpId,
    UINT8 adHocBehavior
);
```

### Returns

None.

### Description

Sets the AdHoc behavior within a Connection Profile. Allowable values are:

- WF\_ADHOC\_CONNECT\_THEN\_START
- WF\_ADHOC\_CONNECT\_ONLY
- WF\_ADHOC\_START\_ONLY

### Remarks

None.

### Preconditions

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	Connection Profile ID
adHocBehavior	Value of the adhoc behavior for this connection profile.

**11.1.1.12 WF\_CPSetBssid Function****File**

WFApi.h

**C**

```
void WF_CPSetBssid(
    UINT8 CpId,
    UINT8 * p_bssid
);
```

**Returns**

None.

**Description**

Sets the BSSID element in a Connection Profile.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	Connection Profile ID
p_bssid	Pointer to the BSSID

**11.1.1.13 WF\_CPSetDefaultWepKeyIndex Function****File**

WFApi.h

**C**

```
void WF_CPSetDefaultWepKeyIndex(
    UINT8 CpId,
    UINT8 defaultWepKeyIndex
);
```

**Returns**

None.

**Description**

Only applicable if the Connection Profile security type is either WF\_SECURITY\_WEP\_40 or WF\_SECURITY\_WEP\_104. Selects which of the four WEP keys to use.

**Remarks**

Note that only key 0 amongst AP manufacturers is typically used. Using any of the other three keys may be unpredictable from brand to brand.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	Connection Profile ID
defaultWepKeyIndex	Index of WEP key to use (0 - 3)

**11.1.1.14 WF\_CPSetElements Function****File**

WFApi.h

**C**

```
void WF_CPSetElements(
    UINT8 CpId,
    tWFCPElements * p_elements
);
```

**Returns**

None.

**Description**

Sets all Connection Profile elements. If the Host CPU does not have enough memory to create a structure of this size then call the individual set functions.

**Remarks**

None.

**Preconditions**

MACInit must be called.

**Parameters**

Parameters	Description
CpId	Connectino Profile ID.
p_elements	Pointer to Connection Profile elements structure.

**11.1.1.15 WF\_CPSetNetworkType Function****File**

WFApi.h

**C**

```
void WF_CPSetNetworkType(
    UINT8 CpId,
    UINT8 networkType
);
```

**Returns**

None.

**Description**

Sets the Network Type element a Connection Profile. Allowable values are:

- WF\_INFRASTRUCTURE

- WF\_ADHOC

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	Connection Profile ID
networkType	Type of network to create (infrastructure or adhoc)

## 11.1.1.16 WF\_CPSetSecurity Function

**File**

WFApi.h

**C**

```
void WF_CPSetSecurity(
    UINT8 CpId,
    UINT8 securityType,
    UINT8 wepKeyIndex,
    UINT8 * p_securityKey,
    UINT8 securityKeyLength
);
```

**Returns**

None.

**Description**

Configures security for a Connection Profile.

Security	Key	Length
WF_SECURITY_OPEN	N/A	N/A
WF_SECURITY_WEP_40	hex	4, 5 byte keys
WF_SECURITY_WEP_104	hex	4, 13 byte keys
WF_SECURITY_WPA_WITH_KEY	hex	32 bytes
WF_SECURITY_WPA_WITH_PASS_PHRASE	ascii	8-63 ascii characters
WF_SECURITY_WPA2_WITH_KEY	hex	32 bytes
WF_SECURITY_WPA2_WITH_PASS_PHRASE	ascii	8-63 ascii characters
WF_SECURITY_WPA_AUTO_WITH_KEY	hex	32 bytes
WF_SECURITY_WPA_AUTO_WITH_PASS_PHRASE	ascii	8-63 ascii characters

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	Connection Profile ID
securityType	Value corresponding to the security type desired.
wepKeyIndex	0 thru 3 (only used if security type is WF_SECURITY_WEP_40 or WF_SECURITY_WEP_104)
p_securityKey	Binary key or passphrase (not used if security is WF_SECURITY_OPEN)
securityKeyLength	Number of bytes in p_securityKey (not used if security is WF_SECURITY_OPEN)

**11.1.1.17 WF\_CPSetSsid Function****File**

WFApi.h

**C**

```
void WF_CPSetSsid(
    UINT8 CpId,
    UINT8 * p_ssid,
    UINT8 ssidLength
);
```

**Returns**

None.

**Description**

Sets the SSID and SSID Length elements in the Connection Profile. Note that if an Access Point can have either a visible or hidden SSID. If an Access Point uses a hidden SSID then an active scan must be used (see scanType field in the Connection Algorithm).

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	Connection Profile ID
p_ssid	Pointer to the SSID string
ssidLength	Number of bytes in the SSID

**11.1.1.18 WFCPElementsStruct Structure****File**

WFApi.h

**C**

```
struct WFCPElementsStruct {
    UINT8 ssid[WF_MAX_SSID_LENGTH];
    UINT8 bssid[WF_BSSID_LENGTH];
    UINT8 ssidLength;
    UINT8 securityType;
    UINT8 securityKey[WF_MAX_SECURITY_KEY_LENGTH];
    UINT8 securityKeyLength;
};
```

```

UINT8 wepDefaultKeyId;
UINT8 networkType;
UINT8 adHocBehavior;
};
    
```

**Members**

Members	Description
UINT8 ssid[WF_MAX_SSID_LENGTH];	SSID, which must be less than or equal to 32 characters. Set to all 0's if not being used. If ssidLength is 0 this field is ignored. If SSID is not defined then the MRF24WB0M, when using this profile to connect (see page 165), will scan all channels within its regional domain. Default: SSID not used.
UINT8 bssid[WF_BSSID_LENGTH];	Basic Service Set Identifier, always 6 bytes. This is the 48-bit MAC of the SSID. It is an optional field that can be used to specify a specific SSID if more than one AP exists with the same SSID. This field can also be used in lieu of the SSID. Set each byte to 0xFF if BSSID is not going to be used. Default: BSSID not used (all FF's)
UINT8 ssidLength;	Number of ASCII bytes in ssid. Set to 0 is SSID is not going to be used. Default: 0
UINT8 securityType;	Designates the desired security level for the connection. Choices are:
UINT8 securityKey[WF_MAX_SECURITY_KEY_LENGTH];	Set to NULL if securityType is WF_SECURITY_OPEN. If securityKeyLength is 0 this field is ignored.
UINT8 securityKeyLength;	Number of bytes used in the securityKey. Set to 0 if securityType is WF_SECURITY_OPEN.
UINT8 wepDefaultKeyId;	This field is only used if securityType is WF_SECURITY_WEP. This field designates which of the four WEP keys defined in securityKey to use when connecting to a WiFi network. The range is 0 thru 3, with the default being 0.
UINT8 networkType;	WF_INFRASTRUCTURE or WF_ADHOC Default: WF_INFRASTRUCTURE
UINT8 adHocBehavior;	Only applicable if networkType is WF_ADHOC. Configures Adhoc behavior. Choices are:

**Description**

Connection profile elements structure

## 11.1.2 Connection Profile Internal Members

**Functions**

	Name	Description
	LowLevel_CPGetElement (see page 553)	Get an element of the connection profile on the MRF24WB0M.
	LowLevel_CPSetElement (see page 553)	Set an element of the connection profile on the MRF24WB0M.

**Module**

Wi-Fi Connection Profile (see page 540)

**Description**

## 11.1.2.1 LowLevel\_CPGetElement Function

### File

WFConnectionProfile.c

### C

```
static void LowLevel_CPGetElement(
    UINT8 CpId,
    UINT8 elementId,
    UINT8 * p_elementData,
    UINT8 elementDataLength,
    UINT8 dataReadAction
);
```

### Returns

None.

### Description

All Connection Profile 'Get Element' functions call this function to construct the management message. The caller must fix up any endian issues prior to calling this function.

### Remarks

None.

### Preconditions

MACInit must be called first.

### Parameters

Parameters	Description
CpId	Connection Profile ID
elementId	Element that is being read
p_elementData	Pointer to where element data will be written
elementDataLength	Number of element data bytes that will be read
dataReadAction	If TRUE then read data per paramters and free mgmt response buffer. If FALSE then return after response received, do not read any data as the caller will do that, and don't free buffer, as caller will do that as well.

## 11.1.2.2 LowLevel\_CPSetElement Function

### File

WFConnectionProfile.c

### C

```
static void LowLevel_CPSetElement(
    UINT8 CpId,
    UINT8 elementId,
    UINT8 * p_elementData,
    UINT8 elementDataLength
);
```

### Returns

None.

### Description

LOCAL FUNCTION PROTOTYPES

All Connection Profile 'Set Element' functions call this function to construct the management message. The caller must fix up

any endian issues prior to calling this function.

#### Remarks

None.

#### Preconditions

MACInit must be called first.

#### Parameters

Parameters	Description
CpId	Connection Profile ID
elementId	Element that is being set
p_elementData	Pointer to element data
elementDataLength	Number of bytes pointed to by p_elementData

## 11.2 Wi-Fi Connection Algorithm

### Module

Wi-Fi API (see page 537)

### Description

The connection algorithm is used to fine-tune the MRF24WB0M algorithm used in the connection process. The connection algorithm can only be changed when the MRF24WB0M is not connected to an 802.11 network.

### 11.2.1 Connection Algorithm Public Members

#### Functions

	Name	Description
≡	WF_CAGetBeaconTimeout (see page 555)	Reads the beacon timeout value.
≡	WF_CAGetBeaconTimeoutAction (see page 556)	Reads the Connection Algorithm beacon timeout action.
≡	WF_CAGetChannelList (see page 557)	Gets the channel list.
≡	WF_CAGetConnectionProfileList (see page 557)	Not currently supported
≡	WF_CAGetDeathAction (see page 558)	Reads the Connection Algorithm death action.
≡	WF_CAGetElements (see page 558)	Reads all Connection Algorithm elements.
≡	WF_CAGetEventNotificationAction (see page 559)	Reads the Connection Algorithm event notification action.
≡	WF_CAGetListenInterval (see page 559)	Gets the listen (see page 169) interval.
≡	WF_CAGetListRetryCount (see page 560)	Gets the list retry count
≡	WF_CAGetMaxChannelTime (see page 560)	Gets the Max Channel Time (in milliseconds)

	WF_CAGetMinChannelTime ( <a href="#">see page 561</a> )	Gets the current Connection Algorithm minimum channel time.
	WF_CAGetProbeDelay ( <a href="#">see page 561</a> )	Gets the Probe Delay (in microseconds)
	WF_CAGetRssi ( <a href="#">see page 562</a> )	Gets the RSSI threshold
	WF_CAGetScanCount ( <a href="#">see page 562</a> )	Gets the scan count
	WF_CAGetScanType ( <a href="#">see page 563</a> )	Gets the Connection Algorithm scan type
	WF_CASetBeaconTimeout ( <a href="#">see page 563</a> )	Sets the beacon timeout value.
	WF_CASetBeaconTimeoutAction ( <a href="#">see page 564</a> )	Action to take if a connection is lost due to a beacon timeout.
	WF_CASetChannelList ( <a href="#">see page 565</a> )	Sets the channel list.
	WF_CASetConnectionProfileList ( <a href="#">see page 565</a> )	Not currently supported
	WF_CASetDeauthAction ( <a href="#">see page 566</a> )	Sets the DeauthAction used by the Connection Algorithm.
	WF_CASetElements ( <a href="#">see page 566</a> )	Writes all Connection Algorithm elements.
	WF_CASetEventNotificationAction ( <a href="#">see page 567</a> )	Sets the WiFi events that the host wishes to be notified of.
	WF_CASetListenInterval ( <a href="#">see page 567</a> )	Sets the listen ( <a href="#">see page 169</a> ) interval.
	WF_CASetListRetryCount ( <a href="#">see page 568</a> )	Sets the list retry count
	WF_CASetMaxChannelTime ( <a href="#">see page 569</a> )	Sets the maximum channel time (in milliseconds)
	WF_CASetMinChannelTime ( <a href="#">see page 569</a> )	Sets the minimum channel time (in milliseconds)
	WF_CASetProbeDelay ( <a href="#">see page 570</a> )	Sets the Probe Delay (in microseconds)
	WF_CASetRssi ( <a href="#">see page 570</a> )	Sets the RSSI threshold
	WF_CASetScanCount ( <a href="#">see page 571</a> )	Sets the scan count
	WF_CASetScanType ( <a href="#">see page 571</a> )	Sets the Connection Algorithm scan type

**Module**

Wi-Fi Connection Algorithm ([see page 554](#))

**Structures**

	Name	Description
	WFCAElementsStruct ( <a href="#">see page 572</a> )	Connection Algorithm Elements

**Description**

The following functions and variables are available to the stack application.

## 11.2.1.1 WF\_CAGetBeaconTimeout Function

**File**

WFApi.h

**C**

```
void WF_CAGetBeaconTimeout(
    UINT8 * p_beaconTimeout
);
```

**Returns**

None.

**Description**

Gets the Beacon Timeout used by the Connection Algorithm.

Value	Description
0	No monitoring of the beacon timeout condition. The host will not be notified of this event.
1-255	Number of beacons missed before disconnect event occurs and beaconTimeoutAction occurs. If enabled, host will receive an event message indicating connection temporarily or permanently lost, and if retrying, a connection successful event.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_beaconTimeout	Pointer where beacon timeout value is written

## 11.2.1.2 WF\_CAGetBeaconTimeoutAction Function

**File**

WFApi.h

**C**

```
void WF_CAGetBeaconTimeoutAction(
    UINT8 * p_beaconTimeoutAction
);
```

**Returns**

None.

**Description**

Gets the Beacon Timeout Action used by the Connection Algorithm.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_beaconTimeoutAction	Pointer where returned value is written. The value

will be either

- WF\_ATTEMPT\_TO\_RECONNECT
- WF\_DO\_NOT\_ATTEMPT\_TO\_RECONNECT

### 11.2.1.3 WF\_CAGetChannelList Function

#### File

WFApi.h

#### C

```
void WF_CAGetChannelList(
    UINT8 * p_channelList,
    UINT8 * p_numChannels
);
```

#### Returns

None.

#### Description

Gets the Channel List used by the Connection Algorithm.

#### Remarks

None.

#### Preconditions

MACInit must be called first.

#### Parameters

Parameters	Description
p_channelList	Pointer to where channel list will be returned
p_numChannels	Pointer to where number of channels in list will be returned

### 11.2.1.4 WF\_CAGetConnectionProfileList Function

#### File

WFApi.h

#### C

```
void WF_CAGetConnectionProfileList(
    UINT8 cpList[WF_CP_LIST_LENGTH]
);
```

#### Returns

None

#### Description

Not currently supported

#### Remarks

Not currently supported. The list size is always WF\_CP\_LIST\_SIZE.

#### Preconditions

MACInit must be called first.

**Parameters**

Parameters	Description
cpList	Array of connection profile ID's used to create CP list

## 11.2.1.5 WF\_CAGetDeauthAction Function

**File**

WFApi.h

**C**

```
void WF_CAGetDeauthAction(
    UINT8 * p_deauthAction
);
```

**Returns**

None.

**Description**

Gets the DeauthAction used by the Connection Algorithm.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_deauthAction	Pointer where returned value is written. The value will
be either	<ul style="list-style-type: none"> <li>WF_ATTEMPT_TO_RECONNECT</li> <li>WF_DO_NOT_ATTEMPT_TO_RECONNECT</li> </ul>

## 11.2.1.6 WF\_CAGetElements Function

**File**

WFApi.h

**C**

```
void WF_CAGetElements(
    tWFCAElements * p_elements
);
```

**Returns**

None

**Description**

Sends a message to the MRF24WB0M which requests all the Connection Algorithm elements.

**Remarks**

None

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_elements	Pointer to the output structure (tWFCAElements) where the connection algorithm elements are written.

## 11.2.1.7 WF\_CAGetEventNotificationAction Function

**File**

WFApi.h

**C**

```
void WF_CAGetEventNotificationAction(
    UINT8 * p_eventNotificationAction
);
```

**Returns**

None.

**Description**

Gets the Event Notification Action used by the Connection Algorithm. The value read back will be a bit mask that corresponds to the following table:

Bit	Event
0	WF_NOTIFY_CONNECTION_ATTEMPT_SUCCESSFUL
1	WF_NOTIFY_CONNECTION_ATTEMPT_FAILED
2	WF_NOTIFY_CONNECTION_TEMPORARILY_LOST
3	WF_NOTIFY_CONNECTION_PERMANENTLY_LOST
4	WF_NOTIFY_CONNECTION_REESTABLISHED

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_eventNotificationAction	Pointer to where returned value is written.

## 11.2.1.8 WF\_CAGetListenInterval Function

**File**

WFApi.h

**C**

```
void WF_CAGetListenInterval(
    UINT16 * p_listenInterval
);
```

**Returns**

None.

**Description**

Gets the Listen Interval used by the Connection Algorithm. This value is measured in 100ms intervals, the default beacon period of APs.

Value	Description
1	MRF24WB0M wakes up every 100ms to receive buffered messages.
2	MRF24WB0M wakes up every 200ms to receive buffered messages.
...	...
65535	MRF24WB0M wakes up every 65535.5 seconds (~109 minutes) to receive buffered messages.

**Remarks**

None.

**Preconditions**

MACInit must be called first. Only used when PS Poll mode is enabled.

**Parameters**

Parameters	Description
p_listenInterval	Pointer to where listen (see page 169) interval is returned

## 11.2.1.9 WF\_CAGetListRetryCount Function

**File**

WFApi.h

**C**

```
void WF_CAGetListRetryCount(
    UINT8 * p_listRetryCount
);
```

**Returns**

None

**Description**

See description in WF\_CASetListRetryCount (see page 568)()

**Remarks**

None

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_listRetryCount	Pointer to where list retry count is written.

## 11.2.1.10 WF\_CAGetMaxChannelTime Function

**File**

WFApi.h

**C**

```
void WF_CAGetMaxChannelTime(
    UINT16 * p_minChannelTime
);
```

**Returns**

None

**Description**

Gets the maximum time the connection manager waits for a probe response after sending a probe request.

**Remarks**

Default is 400ms

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_maxChannelTime	Pointer where maximum channel time is written

## 11.2.1.11 WF\_CAGetMinChannelTime Function

**File**

WFApi.h

**C**

```
void WF_CAGetMinChannelTime(
    UINT16 * p_minChannelTime
);
```

**Returns**

None

**Description**

Gets the minimum time the connection manager waits for a probe response after sending a probe request.

**Remarks**

None

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_minChannelTime	Pointer where minimum time to wait for a probe response (in milliseconds) will be written.

## 11.2.1.12 WF\_CAGetProbeDelay Function

**File**

WFApi.h

**C**

```
void WF_CAGetProbeDelay(
    UINT16 * p_probeDelay
```

```
);
```

**Returns**

None

**Description**

The number of microseconds to delay before transmitting a probe request following the channel change event.

**Remarks**

Default is 20uS

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_probeDelay	Pointer to where probe delay is written

## 11.2.1.13 WF\_CAGetRssi Function

**File**

WFApi.h

**C**

```
void WF_CAGetRssi(
    UINT8 * p_rssi
);
```

**Returns**

None

**Description**

See WF\_CASetRssi (see page 570). Note that this function only retrieves the RSSI threshold used during the connection -- this is not the current RSSI of an existing connection. If it is desired to retrieve the current RSSI state then a scan must be performed and the scan result will contain the current RSSI state.

**Remarks**

Default is 255

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_rssi	Pointer to where RSSI value is written

## 11.2.1.14 WF\_CAGetScanCount Function

**File**

WFApi.h

**C**

```
void WF_CAGetScanCount(
    UINT8 * p_scanCount
);
```

**Returns**

None

**Description**

The number of times the Connection Manager will scan a channel while attempting to find a particular WiFi network.

**Remarks**

Default is 1

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_scanCount	Pointer to where scan count is written

## 11.2.1.15 WF\_CAGetScanType Function

**File**

WFApi.h

**C**

```
void WF_CAGetScanType(
    UINT8 * p_scanType
);
```

**Returns**

None

**Description**

Reads the current Connection Algorithm scan type.

**Remarks**

Active scanning causes the MRF24WB0M to send probe requests. Passive scanning implies the MRF24WB0M only listens for beacons. Default is WF\_ACTIVE\_SCAN.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_scanType	Pointer where Connection Algorithm scan type is written.

## 11.2.1.16 WF\_CASetBeaconTimeout Function

**File**

WFApi.h

**C**

```
void WF_CASetBeaconTimeout(
    UINT8 beaconTimeout
);
```

**Returns**

None.

**Description**

Sets the Beacon Timeout used by the Connection Algorithm.

Value	Description
0	No monitoring of the beacon timeout condition. The host will not be notified of this event.
1-255	Number of beacons missed before disconnect event occurs and beaconTimeoutAction occurs. If enabled, host will receive an event message indicating connection temporarily or permanently lost, and if retrying, a connection successful event.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
beaconTimeout	Number of beacons that can be missed before the action in beaconTimeoutAction is taken.

## 11.2.1.17 WF\_CASetBeaconTimeoutAction Function

**File**

WFApi.h

**C**

```
void WF_CASetBeaconTimeoutAction(
    UINT8 beaconTimeoutAction
);
```

**Returns**

None.

**Description**

Sets the Beacon Timeout Action used by the Connection Algorithm.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
beaconTimeoutAction	Action to take if a connection is lost due
to a beacon timeout. Choices are either	<ul style="list-style-type: none"> <li>WF_ATTEMPT_TO_RECONNECT</li> <li>WF_DO_NOT_ATTEMPT_TO_RECONNECT</li> </ul>

## 11.2.1.18 WF\_CASetChannelList Function

### File

WFApi.h

### C

```
void WF_CASetChannelList(
    UINT8 * p_channelList,
    UINT8 numChannels
);
```

### Returns

None.

### Description

Sets the Channel List used by the Connection Algorithm.

### Remarks

None.

### Preconditions

MACInit must be called first.

### Parameters

Parameters	Description
p_channelList	Pointer to channel list.
numChannels	Number of channels in p_channelList. If set to 0, the MRF24WB0M will use all valid channels for the current regional domain.

## 11.2.1.19 WF\_CASetConnectionProfileList Function

### File

WFApi.h

### C

```
void WF_CASetConnectionProfileList(
    UINT8 cpList[WF_CP_LIST_LENGTH]
);
```

### Returns

None

### Description

Not currently supported

### Remarks

Not currently supported. The list size is always WF\_CP\_LIST\_SIZE. The list should start at index 0. Unused entries in the list must be set to 0xff.

### Preconditions

MACInit must be called first.

### Parameters

Parameters	Description
cpList	Array of connection profile ID's used to create CP list

## 11.2.1.20 WF\_CASetDeauthAction Function

### File

WFApi.h

### C

```
void WF_CASetDeauthAction(
    UINT8 deauthAction
);
```

### Returns

None.

### Description

Action to take if a connection is lost due to receiving a deauthentication message from an AP.

### Remarks

None.

### Preconditions

MACInit must be called first.

### Parameters

Parameters	Description
deauthAction	Action to take in the event of a deauthentication.
Allowable values are	<ul style="list-style-type: none"> <li>WF_ATTEMPT_TO_RECONNECT</li> <li>WF_DO_NOT_ATTEMPT_TO_RECONNECT</li> </ul>

## 11.2.1.21 WF\_CASetElements Function

### File

WFApi.h

### C

```
void WF_CASetElements(
    tWFCAElements * p_elements
);
```

### Returns

None

### Description

Connection Algorithm Functions

Sends a message to the MRF24WB0M which sets all the Connection Algorithm elements.

### Remarks

None

### Preconditions

MACInit must be called first.

**Parameters**

Parameters	Description
p_elements	Pointer to the input structure (tWFCAElements) containing the connection algorithm elements.

**11.2.1.22 WF\_CASetEventNotificationAction Function****File**

WFApi.h

**C**

```
void WF_CASetEventNotificationAction(
    UINT8 eventNotificationAction
);
```

**Returns**

None.

**Description**

Sets the Event Notification Action used by the Connection Algorithm. The bit mask for the allowable entries is as follows:

Bit	Event
0	WF_NOTIFY_CONNECTION_ATTEMPT_SUCCESSFUL
1	WF_NOTIFY_CONNECTION_ATTEMPT_FAILED
2	WF_NOTIFY_CONNECTION_TEMPORARILY_LOST
3	WF_NOTIFY_CONNECTION_PERMANENTLY_LOST
4	WF_NOTIFY_CONNECTION_REESTABLISHED

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
eventNotificationAction	Bit mask indicating which events the host wants to be notified of.

**11.2.1.23 WF\_CASetListenInterval Function****File**

WFApi.h

**C**

```
void WF_CASetListenInterval(
    UINT16 listenInterval
);
```

**Returns**

None.

**Description**

Sets the listen (see page 169) interval used by the Connection Algorithm. This value is measured in 100ms intervals, the default beacon period of APs.

Value	Description
1	MRF24WB0M wakes up every 100ms to receive buffered messages.
2	MRF24WB0M wakes up every 200ms to receive buffered messages.
...	...
65535	MRF24WB0M wakes up every 65535.5 seconds (~109 minutes) to receive buffered messages.

**Remarks**

None.

**Preconditions**

MACInit must be called first. Only used when PS Poll mode is enabled.

**Parameters**

Parameters	Description
listenInterval	Number of 100ms intervals between instances when the MRF24WB0M wakes up to receive buffered messages from the network.

## 11.2.1.24 WF\_CASetListRetryCount Function

**File**

WFApi.h

**C**

```
void WF_CASetListRetryCount (
    UINT8 listRetryCount
);
```

**Returns**

None

**Description**

Number of times to cycle through Connection Profile List before giving up on the connection attempt. Since lists are not yet supported, this function actually sets the number of times the Connection Manager will try to connect (see page 165) with the current Connection Profile before giving up.

**Remarks**

None

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
listRetryCount	0 to 254 or WF_RETRY_FOREVER (255)

## 11.2.1.25 WF\_CASetMaxChannelTime Function

### File

WFApi.h

### C

```
void WF_CASetMaxChannelTime (
    UINT16 minChannelTime
);
```

### Returns

None

### Description

The maximum time (in milliseconds) the connection manager will wait for a probe response after sending a probe request. If no probe responses are received in maxChannelTime then the connection manager will go on to the next channel, if any are left to scan, or quit.

### Remarks

Default is 400ms

### Preconditions

MACInit must be called first.

### Parameters

Parameters	Description
maxChannelTime	Maximum time to wait for a probe response (in milliseconds)

## 11.2.1.26 WF\_CASetMinChannelTime Function

### File

WFApi.h

### C

```
void WF_CASetMinChannelTime (
    UINT16 minChannelTime
);
```

### Returns

None

### Description

The minimum time (in milliseconds) the connection manager will wait for a probe response after sending a probe request. If no probe responses are received in minChannelTime then the connection manager will go on to the next channel, if any are left to scan, or quit.

### Remarks

Default is 200ms

### Preconditions

MACInit must be called first.

### Parameters

Parameters	Description
minChannelTime	Minimum time to wait for a probe response (in milliseconds)

## 11.2.1.27 WF\_CASetProbeDelay Function

### File

WFApi.h

### C

```
void WF_CASetProbeDelay(  
    UINT16 probeDelay  
);
```

### Returns

None

### Description

The number of microseconds to delay before transmitting a probe request following the channel change event.

### Remarks

Default is 20uS

### Preconditions

MACInit must be called first.

### Parameters

Parameters	Description
probeDelay	Desired probe delay

## 11.2.1.28 WF\_CASetRssi Function

### File

WFApi.h

### C

```
void WF_CASetRssi(  
    UINT8 rssi  
);
```

### Returns

None

### Description

Specifies the RSSI behavior when connecting. This value is only used if 1) The current Connection Profile has not defined an SSID or BSSID 2) An SSID is defined in the current Connection Profile and multiple access points are discovered with the same SSID.

Values: 0 : Connect to the first network found 1 - 254: Only connect (see page 165) to a network if the RSSI is greater than or equal to the specified value 255: Connect to the highest RSSI found

Note that RSSI is a relative value with no units -- it is not correlated to dBm.

### Remarks

Default is 255

### Preconditions

MACInit must be called first.

**Parameters**

Parameters	Description
scanCount	Desired scan count

**11.2.1.29 WF\_CASetScanCount Function****File**

WFApi.h

**C**

```
void WF_CASetScanCount(
    UINT8 scanCount
);
```

**Returns**

None

**Description**

The number of times the Connection Manager will scan a channel while attempting to find a particular WiFi network.

**Remarks**

Default is 1

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
scanCount	Desired scan count

**11.2.1.30 WF\_CASetScanType Function****File**

WFApi.h

**C**

```
void WF_CASetScanType(
    UINT8 scanType
);
```

**Returns**

None

**Description**

Configures the Connection Algorithm for the desired scan type.

**Remarks**

Active scanning causes the MRF24WB0M to send probe requests. Passive scanning implies the MRF24WB0M only listens for beacons. Default is WF\_ACTIVE\_SCAN.

**Preconditions**

MACInit must be called first.

Parameters

Parameters	Description
scanType	Desired scan type. Either WF_ACTIVE_SCAN or WF_PASSIVE_SCAN.

### 11.2.1.31 WFCAElementsStruct Structure

File

WFApi.h

C

```

struct WFCAElementsStruct {
    UINT16 listenInterval;
    UINT8 scanType;
    UINT8 rssi;
    UINT8 connectionProfileList[WF_CP_LIST_LENGTH];
    UINT8 listRetryCount;
    UINT8 eventNotificationAction;
    UINT8 beaconTimeoutAction;
    UINT8 deauthAction;
    UINT8 channelList[WF_CHANNEL_LIST_LENGTH];
    UINT8 numChannelsInList;
    UINT8 beaconTimeout;
    UINT8 scanCount;
    UINT8 pad1;
    UINT16 minChannelTime;
    UINT16 maxChannelTime;
    UINT16 probeDelay;
};
    
```

Members

Members	Description
UINT16 listenInterval;	<p>This parameter is only used when PS Poll mode is enabled. See WF_PsPollEnable (see page 585). Number of 100ms intervals between instances when the MRF24WB0M wakes up to received buffered messages from the network. Range is from 1 (100ms) to 6553.5 sec (~109 min).</p> <p>Note that the 802.11 standard defines the listen (see page 169) interval in terms of Beacon Periods, which are typically 100ms. If the MRF24WB0M is communicating to a network with a network that has Beacon Periods that is not 100ms it will round up (or down) as needed to match the actual Beacon Period as closely as possible.</p> <p>Important Note: If the listenInterval is modified while connected to a network the MRF24WB0M will automatically reconnect to the network with the new Beacon Period value. This may cause a temporary loss of data packets.</p>
UINT8 scanType;	<p>WF_ACTIVE_SCAN (Probe Requests sent out) or WF_PASSIVE_SCAN (listen (see page 169) only)</p> <p>Default: WF_ACTIVE_SCAN</p>
UINT8 rssi;	<p>Specifies RSSI restrictions when connecting. This field is only used if:</p> <ol style="list-style-type: none"> <li>1. The Connection Profile has not defined a SSID or BSSID, or</li> <li>2. An SSID is defined in the Connection Profile and multiple AP's are discovered with the same SSID.</li> </ol>
UINT8 connectionProfileList[WF_CP_LIST_LENGTH];	<p><b>Note: Connection Profile lists are not yet supported. This array should be set to all FF's.</b></p>
UINT8 listRetryCount;	<p>This field is used to specify the number of retries for the single connection profile before taking the connection lost action.</p> <p>Range 1 to 254 or WF_RETRY_FOREVER (255)</p> <p>Default is 3</p>

UINT8 eventNotificationAction;	There are several connection-related events that can occur. The Host has the option to be notified (or not) when some of these events occur. This field controls event notification for connection-related events.
UINT8 beaconTimeoutAction;	Specifies the action the Connection Manager should take if a Connection is lost due to a Beacon Timeout. If this field is set to WF_ATTEMPT_TO_RECONNECT then the number of attempts is limited to the value in listRetryCount. Choices are: WF_ATTEMPT_TO_RECONNECT or WF_DO_NOT_ATTEMPT_TO_RECONNECT Default: WF_ATTEMPT_TO_RECONNECT
UINT8 deauthAction;	Designates what action the Connection Manager should take if it receives a Deauthentication message from the AP. If this field is set to WF_ATTEMPT_TO_RECONNECT then the number of attempts is limited to the value in listRetryCount. Choices are: WF_ATTEMPT_TO_RECONNECT or WF_DO_NOT_ATTEMPT_TO_RECONNECT Default: WF_ATTEMPT_TO_RECONNECT
UINT8 channelList[WF_CHANNEL_LIST_LENGTH];	List of one or more channels that the MRF24WB0M should utilize when connecting or scanning. If numChannelsInList is set to 0 then this parameter should be set to NULL. Default: All valid channels for the regional domain of the MRF24WB0M (set at manufacturing).
UINT8 numChannelsInList;	Number of channels in channelList. If set to 0 then the MRF24WB0M will populate the list with all valid channels for the regional domain. Default: The number of valid channels for the regional domain of the MRF24WB0M (set at manufacturing).
UINT8 beaconTimeout;	Specifies the number of beacons that can be missed before the action described in beaconTimeoutAction is taken.
UINT8 scanCount;	The number of times to scan a channel while attempting to find a particular access point. Default: 1
UINT16 minChannelTime;	The minimum time (in milliseconds) the connection manager will wait for a probe response after sending a probe request. If no probe responses are received in minChannelTime then the connection manager will go on to the next channel, if any are left to scan, or quit. Default: 200ms
UINT16 maxChannelTime;	If a probe response is received within minChannelTime then the connection manager will continue to collect any additional probe responses up to maxChannelTime before going to the next channel in the channelList. Units are in milliseconds. Default: 400ms
UINT16 probeDelay;	The number of microseconds to delay before transmitting a probe request following the channel change event. Default: 20us

**Description**

Connection Algorithm Elements

## 11.2.2 Connection Algorithm Internal Members

**Functions**

	Name	Description
	LowLevel_CAGetElement ( <a href="#">↗</a> see page 574)	Get an element of the connection algorithm on the MRF24WB0M.

	LowLevel_CASetElement (see page 574)	Set an element of the connection algorithm on the MRF24WB0M.
	SetEventNotificationMask (see page 575)	Sets the event notification mask.

**Module**

Wi-Fi Connection Algorithm (see page 554)

**Description**

The following functions and variables are designated as internal to the module.

### 11.2.2.1 LowLevel\_CAGetElement Function

**File**

WFCConnectionAlgorithm.c

**C**

```
static void LowLevel_CAGetElement(
    UINT8 elementId,
    UINT8 * p_elementData,
    UINT8 elementDataLength,
    UINT8 dataReadAction
);
```

**Returns**

None.

**Description**

Low-level function to send the appropriate management message to the MRF24WB0M to get the Connection Algorithm element.

**Remarks**

All Connection Algorithm 'Get Element' functions call this function to construct the management message. The caller must fix up any endian issues after getting the data from this function.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
elementId	Element that is being read
p_elementData	Pointer to where element data will be written
elementDataLength	Number of element data bytes that will be read
dataReadAction	If TRUE then read data per paramters and free mgmt response buffer. If FALSE then return after response received, do not read any data as the caller will do that, and don't free buffer, as caller will do that as well.

### 11.2.2.2 LowLevel\_CASetElement Function

**File**

WFCConnectionAlgorithm.c

**C**

```
static void LowLevel_CASetElement(
    UINT8 elementId,
    UINT8 * p_elementData,
```

```

    UINT8 elementDataLength
);

```

**Returns**

None.

**Description**

LOCAL FUNCTION PROTOTYPES

Low-level function to send the appropriate management message to the MRF24WB0M to set the Connection Algorithm element.

**Remarks**

All Connection Algorithm 'Set Element' functions call this function to construct the management message. The caller must fix up any endian issues prior to calling this function.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
elementId	Element that is being set
p_elementData	Pointer to element data
elementDataLength	Number of bytes pointed to by p_elementData

### 11.2.2.3 SetEventNotificationMask Function

**File**

WFConnectionAlgorithm.c

**C**

```

static void SetEventNotificationMask(
    UINT8 eventNotificationBitMask
);

```

**Returns**

None.

**Description**

Sets the event notification mask for the Connection Algorithm. Allowable values are:

Value	Event
0x01	WF_NOTIFY_CONNECTION_ATTEMPT_SUCCESSFUL
0x02	WF_NOTIFY_CONNECTION_ATTEMPT_FAILED
0x04	WF_NOTIFY_CONNECTION_TEMPORARILY_LOST
0x08	WF_NOTIFY_CONNECTION_PERMANENTLY_LOST
0x10	WF_NOTIFY_CONNECTION_REESTABLISHED
0x1f	WF_NOTIFY_ALL_EVENTS

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
eventNotificationBitMask	Bit mask defining which events the host will be notified of.

---

## 11.3 Wi-Fi Connection Manager

**Module**

Wi-Fi API ([see page 537](#))

**Description**

The connection manager uses the connection algorithm and one or more connection profiles to connect ([see page 165](#)) to a network.

---

### 11.3.1 Connection Manager Public Members

**Functions**

	Name	Description
⇒	WF_CMConnect ( <a href="#">see page 576</a> )	Commands the MRF24WB0M to start a connection.
⇒	WF_CMDisconnect ( <a href="#">see page 577</a> )	Commands the MRF24WB0M to close any open connections and/or to cease attempting to connect ( <a href="#">see page 165</a> ).
⇒	WF_CMGetConnectionState ( <a href="#">see page 577</a> )	Returns the current connection state.
⇒	WF_CMInfoGetFSMStats ( <a href="#">see page 578</a> )	CM Info Functions

**Module**

Wi-Fi Connection Manager ([see page 576](#))

**Description**

The following functions and variables are available to the stack application.

#### 11.3.1.1 WF\_CMConnect Function

**File**

WFApi.h

**C**

```
void WF_CMConnect (
    UINT8 CpId
);
```

**Returns**

None.

**Description**

Connection Manager Functions

Directs the Connection Manager to scan for and connect ([see page 165](#)) to a WiFi network. This function does not wait

until the connection attempt is successful, but returns immediately. See `WF_ProcessEvent` (see page 595) for events that can occur as a result of a connection attempt being successful or not.

Note that if the Connection Profile being used has WPA or WPA2 security enabled and is using a passphrase, the connection manager will first calculate the PSK key, and then start the connection process. The key calculation can take up to 30 seconds.

#### Remarks

None.

#### Preconditions

MACInit must be called first.

#### Parameters

Parameters	Description
CpId	If this value is equal to an existing Connection Profile's ID than only that Connection Profile will be used to attempt a connection to a WiFi network. If this value is set to <code>WF_CM_CONNECT_USING_LIST</code> then the <code>connectionProfileList</code> will be used to connect (see page 165), starting with the first Connection Profile in the list.

### 11.3.1.2 WF\_CMDisconnect Function

#### File

WFApi.h

#### C

```
void WF_CMDisconnect();
```

#### Returns

None.

#### Description

Directs the Connection Manager to close any open connection or connection attempt in progress. No further attempts to connect (see page 165) are taken until `WF_CMConnect` (see page 576)() is called. Generates the event `WF_EVENT_CONNECTION_PERMANENTLY_LOST` when the connection is successfully terminated.

#### Remarks

None.

#### Preconditions

MACInit must be called.

### 11.3.1.3 WF\_CMGetConnectionState Function

#### File

WFApi.h

#### C

```
void WF_CMGetConnectionState(
    UINT8 * p_state,
    UINT8 * p_currentCpId
);
```

#### Returns

None.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_state	Pointer to location where connection state will be written
p_currentCpld	Pointer to location of current connection profile ID that is being queried.

### 11.3.1.4 WF\_CMInfoGetFSMStats Function

**File**

WFApi.h

**C**

```
void WF_CMInfoGetFSMStats(
    tWF_CMInfoFSMStats * p_info
);
```

**Description**

CM Info Functions

## 11.4 Wi-Fi Scan

**Module**

Wi-Fi API (see page 537)

**Description**

If the application already knows the network SSID that it wants to join than it can set up a connection profile with that information and attempt to join the network. However, there are applications that first need to dynamically determine what infrastructure and/or adhoc networks are in the area, and then decide which network to join. The scan API functions are used to gather this information.

### 11.4.1 Scan Public Members

**Functions**

	Name	Description
⇒	WF_Scan (see page 579)	Commands the MRF24WB0M to start a scan operation. This will generate the WF_EVENT_SCAN_RESULTS_READY event.
⇒	WF_ScanGetResult (see page 580)	Read scan results back from MRF24WB0M.

**Module**

Wi-Fi Scan (see page 578)

**Description**

The following functions and variables are available to the stack application.

### 11.4.1.1 WF\_Scan Function

**File**

WFApi.h

**C**

```
void WF_Scan(
    UINT8 CpId
);
```

**Returns**

None.

**Description**

Scan Functions

Directs the MRF24WB0M to initiate a scan operation utilizing the input Connection Profile ID. The Host Application will be notified that the scan results are ready when it receives the WF\_EVENT\_SCAN\_RESULTS\_READY event. The eventInfo field for this event will contain the number of scan results. Once the scan results are ready they can be retrieved with WF\_ScanGetResult (see page 580).

Scan results are retained on the MRF24WB0M until:

1. Calling WF\_Scan() again (after scan results returned from previous call).
2. MRF24WB0M reset.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
CpId	<p>Connection Profile to use. If the CpId is valid then the values from that Connection Profile will be used for filtering scan results. If the CpId is set to WF_SCAN_ALL (0xFF) then a default filter will be used.</p> <p>Valid CpId</p> <ul style="list-style-type: none"> <li>• If CP has a defined SSID only scan results with that SSID are retained.</li> <li>• If CP does not have a defined SSID then all scanned SSID's will be retained</li> <li>• Only scan results from Infrastructure or AdHoc networks are retained, depending on the value of networkType in the Connection Profile</li> <li>• The channel list that is scanned will be determined from channelList in the Connection Algorithm (which must be defined before calling this function).</li> </ul> <p>CpId is equal to WF_SCAN_ALL</p> <ul style="list-style-type: none"> <li>• All scan results are retained (both Infrastructure and Ad Hoc networks).</li> <li>• All channels within the MRF24WB0M's regional domain will be scanned.</li> <li>• No Connection Profiles need to be defined before calling this function.</li> <li>• The Connection Algorithm does not need to be defined before calling this function.</li> </ul>

### 11.4.1.2 WF\_ScanGetResult Function

**File**

WFApi.h

**C**

```
void WF_ScanGetResult(
    UINT8 listIndex,
    tWFScanResult * p_scanResult
);
```

**Returns**

None.

**Description**

After a scan has completed this function is used to read one or more of the scan results from the MRF24WB0M. The scan results will be written contiguously starting at p\_scanResults (see tWFScanResult structure for format of scan result).

**Remarks**

None.

**Preconditions**

MACInit must be called first. WF\_EVENT\_SCAN\_RESULTS\_READY event must have already occurred.

**Parameters**

Parameters	Description
listIndex	Index (0-based list) of the scan entry to retrieve.
p_scanResult	Pointer to location to store the scan result structure
Retrieve RSSI	RSSI_MAX (200) , RSSI_MIN (106) p_scanResult->rssi

## 11.5 Wi-Fi Tx Power Control

**Module**

Wi-Fi API (see page 537)

**Description**

The API functions in this section are used to configure the MRF24WB0M Tx power control settings.

An application can control Tx power by modify the max Tx power via WF\_TxPowerSetMinMax (see page 581)() to something other than 10dBm.

### 11.5.1 Tx Power Control Public Members

**Functions**

	Name	Description
◆	WF_TxPowerGetMinMax (see page 581)	Gets the Tx min and max power on the MRF24WB0M.
◆	WF_TxPowerSetMinMax (see page 581)	Sets the Tx min and max power on the MRF24WB0M.

	WF_TxPowerGetFactoryMax ( <a href="#">see page 582</a> )	Retrieves the factory-set max Tx power from the MRF24WB0M.
---	---	--

**Module**

Wi-Fi Tx Power Control ([see page 580](#))

**Description**

The following functions and variables are available to the stack application.

## 11.5.1.1 WF\_TxPowerGetMinMax Function

**File**

WFApi.h

**C**

```
void WF_TxPowerGetMinMax(
    INT8 * p_minTxPower,
    INT8 * p_maxTxPower
);
```

**Returns**

None.

**Description**

After initialization the MRF24WB0M max Tx power is determined by a factory-set value. This function can set a different minimum and maximum Tx power levels. However, this function can never set a maximum Tx power greater than the factory-set value, which can be read via WF\_TxPowerGetFactoryMax ([see page 582](#))().

**Remarks**

No conversion of units needed, input to MRF24WB0M is in dB.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_minTxPower	Pointer to location to write the minTxPower
p_maxTxPower	Pointer to location to write the maxTxPower

## 11.5.1.2 WF\_TxPowerSetMinMax Function

**File**

WFApi.h

**C**

```
void WF_TxPowerSetMinMax(
    INT8 minTxPower,
    INT8 maxTxPower
);
```

**Returns**

None.

**Description**

Tx Power Control Functions

After initialization the MRF24WB0M max Tx power is determined by a factory-set value. This function can set a different

minimum and maximum Tx power levels. However, this function can never set a maximum Tx power greater than the factory-set value, which can be read via `WF_TxPowerGetFactoryMax` (see page 582).

**Remarks**

No conversion of units needed, input to MRF24WB0M is in dB.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
minTxPower	Desired minTxPower (-10 to 10dB)
maxTxPower	Desired maxTxPower (-10 to 10dB)

### 11.5.1.3 WF\_TxPowerGetFactoryMax Function

**File**

WFApi.h

**C**

```
void WF_TxPowerGetFactoryMax(
    INT8 * p_factoryMaxTxPower
);
```

**Returns**

None.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_factoryMaxTxPower	Desired maxTxPower (-10 to 10dB), in 1dB steps

## 11.6 Wi-Fi Power Save

**Module**

Wi-Fi API (see page 537)

**Description**

The MRF24WB0M supports two power-saving modes – sleep and hibernate.

Mode	Description
Sleep	This mode is used when in PS Poll mode where the MRF24WB0M wakes itself up at periodic intervals to query the network for receive messages buffered by an Access Point. See listenInterval in the tWFCAElements structure. When in sleep mode the MRF24WB0M transmitter receiver circuits are turned off along with other circuitry to minimize power consumption. Sleep mode is entered periodically as a result of the Host CPU enabling PS Poll mode.

Hibernate	<p>This mode effectively turns the MRF24WB0M off for maximum power savings. MRF24WB0M state is not retained, and when the MRF24WB0M is taken out of the Hibernate state it performs a reboot.</p> <p>Hibernate mode is controlled by toggling the XCEN33 pin on the MRF24WB0M module (high to enter hibernate, low to exit).</p> <p>This mode should be used when the application allows for the MRF24WB0M module to be off for extended periods of time.</p>
-----------	---

**Power Save Functions**

802.11 chipsets have two well known operational power modes. Active power mode is defined as the radio always on either transmitting or receiving, meaning that when it isn't transmitting then it is trying to receive. Power save mode is defined as operating with the radio turned off when there is nothing to transmit and only turning the radio receiver on when required.

The power save mode is a mode that requires interaction with an Access Point. The access point is notified via a packet from the Station that it is entering into power save mode. As a result the access point is required to buffer any packets that are destined for the Station until the Station announces that it is ready to once again receive packets. The duration that a Station is allowed to remain in this mode is limited and is typically 10 times the beacon interval of the Access point.

If the host is expecting packets from the network it should operate in Active mode. If however power saving is critical and packets are not expected then the host should consider operating in power save mode. Due to the nature of Access points not all behaving the same, there is the possibility that an Access point will invalidate a Stations connection if it has not heard from the Station over a given time period. For this reason power save mode should be used with caution.

The 802.11 name for power saving mode is PS-Poll (Power-Save Poll).

## 11.6.1 Power Save Public Members

**Functions**

	Name	Description
⇒	WF_GetPowerSaveState (see page 583)	Returns current power-save state.
⇒	WF_HibernateEnable (see page 584)	Puts the MRF24WB0M into hibernate mode.
⇒	WF_PsPollDisable (see page 585)	Disables PS-Poll mode.
⇒	WF_PsPollEnable (see page 585)	Enables PS Poll mode.

**Module**

Wi-Fi Power Save (see page 582)

**Description**

The following functions and variables are available to the stack application.

### 11.6.1.1 WF\_GetPowerSaveState Function

**File**

WFApi.h

**C**

```
void WF_GetPowerSaveState(
    UINT8 * p_powerSaveState
);
```

**Returns**

None.

**Description**

Returns the current MRF24WB0M power save state.

Value	Definition
WF_PS_HIBERNATE	MRF24WB0M in hibernate state
WF_PS_PS_POLL_DTIM_ENABLED	MRF24WB0M in PS-Poll mode with DTIM enabled
WF_PS_PS_POLL_DTIM_DISABLED	MRF24WB0M in PS-Poll mode with DTIM disabled
WF_PS_POLL_OFF	MRF24WB0M is not in any power-save state

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_powerSaveState	Pointer to where power state is written

## 11.6.1.2 WF\_HibernateEnable Function

**File**

WFApi.h

**C**

```
void WF_HibernateEnable();
```

**Returns**

None.

**Description**

Enables Hibernate mode on the MRF24WB0M, which effectively turns off the device for maximum power savings.

MRF24WB0M state is not maintained when it transitions to hibernate mode. To remove the MRF24WB0M from hibernate mode call WF\_Init().

**Remarks**

Note that because the MRF24WB0M does not save state, there will be a disconnect between the TCP/IP stack and the MRF24B0M state. If it is desired by the application to use hibernate, additional measures must be taken to save application state. Then the host should be reset. This will ensure a clean connection between MRF24WB0M and TCP/IP stack

Future versions of the stack might have the ability to save stack context as well, ensuring a clean wake up for the MRF24WB0M without needing a host reset.

**Preconditions**

MACInit must be called first.

### 11.6.1.3 WF\_PsPollDisable Function

**File**

WFApi.h

**C**

```
void WF_PsPollDisable();
```

**Returns**

None.

**Description**

Power Management Functions

Disables PS Poll mode. The MRF24WB0M will stay active and not go sleep.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

### 11.6.1.4 WF\_PsPollEnable Function

**File**

WFApi.h

**C**

```
void WF_PsPollEnable(  
    BOOL rxDtim  
);
```

**Returns**

None.

**Description**

Enables PS Poll mode. PS-Poll (Power-Save Poll) is a mode allowing for longer battery life. The MRF24WB0M coordinates with the Access Point to go to sleep and wake up at periodic intervals to check for data messages, which the Access Point will buffer. The listenInterval in the Connection Algorithm defines the sleep interval. By default, PS-Poll mode is disabled.

When PS Poll is enabled, the WF Host Driver will automatically force the MRF24WB0M to wake up each time the Host sends Tx data or a control message to the MRF24WB0M. When the Host message transaction is complete the MRF24WB0M driver will automatically re-enable PS Poll mode.

When the application is likely to experience a high volume of data traffic then PS-Poll mode should be disabled for two reasons:

1. No power savings will be realized in the presence of heavy data traffic.
2. Performance will be impacted adversely as the WiFi Host Driver continually activates and deactivates PS-Poll mode via SPI messages.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
rxDtim	TRUE if MRF24WB0M should wake up periodically and check for buffered broadcast messages, else FALSE

## 11.6.2 Power Save Internal Members

**Functions**

	Name	Description
≡	SendPowerModeMsg (see page 586)	Send power mode management message to the MRF24WB0M.
≡	SetPowerSaveState (see page 586)	Sets the desired power save state of the MRF24WB0M.

**Module**

Wi-Fi Power Save (see page 582)

**Description**

The following functions and variables are designated as internal to the module.

### 11.6.2.1 SendPowerModeMsg Function

**File**

WFPowerSave.c

**C**

```
static void SendPowerModeMsg(
    tWFPwrModeReq * p_powerMode
);
```

**Returns**

None.

**Description**

LOCAL FUNCTION PROTOTYPES

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_powerMode	Pointer to tWFPwrModeReq structure to send to MRF24WB0M.

### 11.6.2.2 SetPowerSaveState Function

**File**

WFPowerSave.c

**C**

```
static void SetPowerSaveState(
    UINT8 powerSaveState
);
```

**Returns**

None.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
powerSaveState	Value of the power save state desired.

## 11.7 Wi-Fi Miscellaneous

**Module**

Wi-Fi API (see page 537)

**Description**

### 11.7.1 Wi-Fi Miscellaneous Public Members

**Functions**

	Name	Description
◆	WF_GetDeviceInfo (see page 588)	Retrieves WF device information
◆	WF_GetMacAddress (see page 588)	Retrieves the MRF24WB0M MAC address
◆	WF_GetMacStats (see page 589)	Gets MAC statistics.
◆	WF_GetMultiCastFilter (see page 589)	Gets a multicast address filter from one of the two multicast filters.
◆	WF_GetRegionalDomain (see page 590)	Retrieves the MRF24WB0M Regional domain
◆	WF_GetRtsThreshold (see page 591)	Gets the RTS Threshold
◆	WF_SetMacAddress (see page 591)	Uses a different MAC address for the MRF24WB0M
◆	WF_SetMultiCastFilter (see page 592)	Sets a multicast address filter using one of the two multicast filters.
◆	WF_SetRegionalDomain (see page 592)	Enables or disables the MRF24WB0M Regional Domain.
◆	WF_SetRtsThreshold (see page 593)	Sets the RTS Threshold.

**Module**

Wi-Fi Miscellaneous ([see page 587](#))

**Structures**

	Name	Description
	tWFDeviceInfoStruct ( <a href="#">see page 593</a> )	used in WF_GetDeviceInfo ( <a href="#">see page 588</a> )
	WFMacStatsStruct ( <a href="#">see page 594</a> )	This is record WFMacStatsStruct.

**Description**

The following functions and variables are available to the stack application.

## 11.7.1.1 WF\_GetDeviceInfo Function

**File**

WFApi.h

**C**

```
void WF_GetDeviceInfo(
    tWFDeviceInfo * p_deviceInfo
);
```

**Returns**

None.

**Description**

Version functions

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_deviceInfo	Pointer where device info will be written

## 11.7.1.2 WF\_GetMacAddress Function

**File**

WFApi.h

**C**

```
void WF_GetMacAddress(
    UINT8 * p_mac
);
```

**Returns**

None.

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_mac	Pointer where mac will be written (must point to a 6-byte buffer)

## 11.7.1.3 WF\_GetMacStats Function

**File**

WFApi.h

**C**

```
void WF_GetMacStats(
    tWFMacStats * p_macStats
);
```

**Returns**

None.

**Description**

MAC Stats

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_macStats	Pointer to where MAC statistics are written

## 11.7.1.4 WF\_GetMultiCastFilter Function

**File**

WFApi.h

**C**

```
void WF_GetMultiCastFilter(
    UINT8 multicastFilterId,
    UINT8 multicastAddress[6]
);
```

**Returns**

None.

**Description**

Gets the current state of the specified Multicast Filter.

Normally would call SendGetParamMsg, but this GetParam returns all 6 address filters + 2 more bytes for a total of 48 bytes plus header. So, doing this msg manually to not require a large stack allocation to hold all the data.

Exact format of returned message is: [0] -- always mgmt response (2) [1] -- always WF\_GET\_PARAM\_SUBTYPE (16) [2] -- result (1 if successful) [3] -- mac state (not used) [4] -- data length (length of response data starting at index 6) [5] -- not used [6-11] -- Compare Address (see page 141) 0 address [12] -- Compare Address (see page 141) 0 group [13] -- Compare

Address (see page 141) 0 type [14-19] -- Compare Address (see page 141) 1 address [20] -- Compare Address (see page 141) 1 group [21] -- Compare Address (see page 141) 1 type [22-27] -- Compare Address (see page 141) 2 address [28] -- Compare Address (see page 141) 2 group [29] -- Compare Address (see page 141) 2 type [30-35] -- Compare Address (see page 141) 3 address [36] -- Compare Address (see page 141) 3 group [37] -- Compare Address (see page 141) 3 type [38-43] -- Compare Address (see page 141) 4 address [44] -- Compare Address (see page 141) 4 group [45] -- Compare Address (see page 141) 4 type [46-51] -- Compare Address (see page 141) 5 address [52] -- Compare Address (see page 141) 5 group [53] -- Compare Address (see page 141) 5 type

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
multicastFilterId	WF_MULTICAST_FILTER_1 or WF_MULTICAST_FILTER_2
multicastAddress	6-byte address

## 11.7.1.5 WF\_GetRegionalDomain Function

**File**

WFApi.h

**C**

```
void WF_GetRegionalDomain(
    UINT8 * p_regionalDomain
);
```

**Returns**

None.

**Description**

see tWFRegDomain enumerated types

Gets the regional domain on the MRF24WB0M. Allowable values are:

- WF\_DOMAIN\_FCC
- WF\_DOMAIN\_IC
- WF\_DOMAIN\_ETSI
- WF\_DOMAIN\_SPAIN
- WF\_DOMAIN\_FRANCE
- WF\_DOMAIN\_JAPAN\_A
- WF\_DOMAIN\_JAPAN\_B

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
p_regionalDomain	Pointer where the regional domain value will be written

## 11.7.1.6 WF\_GetRtsThreshold Function

### File

WFApi.h

### C

```
void WF_GetRtsThreshold(
    UINT16 * p_rtsThreshold
);
```

### Returns

None.

### Description

Gets the RTS/CTS packet size threshold.

### Remarks

None.

### Preconditions

MACInit must be called first.

### Parameters

Parameters	Description
p_rtsThreshold	Pointer to where RTS threshold is written

## 11.7.1.7 WF\_SetMacAddress Function

### File

WFApi.h

### C

```
void WF_SetMacAddress(
    UINT8 * p_mac
);
```

### Returns

None.

### Description

MAC Address (see page 141) Functions

Directs the MRF24WB0M to use the input MAC address instead of its factory-default MAC address. This function does not overwrite the factory default, which is in FLASH memory – it simply tells the MRF24WB0M to use a different MAC.

### Remarks

None.

### Preconditions

MACInit must be called first. Cannot be called when the MRF24WB0M is in a connected state.

### Parameters

Parameters	Description
p_mac	Pointer to 6-byte MAC that will be sent to MRF24WB0M

## 11.7.1.8 WF\_SetMultiCastFilter Function

### File

WFApi.h

### C

```
void WF_SetMultiCastFilter(
    UINT8 multicastFilterId,
    UINT8 multicastAddress[6]
);
```

### Returns

None.

### Description

Multicast Functions

This function allows the application to configure up to two Multicast Address (see page 141) Filters on the MRF24WB0M. If two active multicast filters are set up they are OR'd together – the MRF24WB0M will receive and pass to the Host CPU received packets from either multicast address. The allowable values for the multicast filter are:

- WF\_MULTICAST\_FILTER\_1
- WF\_MULTICAST\_FILTER\_2

By default, both Multicast Filters are inactive.

### Remarks

None.

### Preconditions

MACInit must be called first.

### Parameters

Parameters	Description
multicastFilterId	WF_MULTICAST_FILTER_1 or WF_MULTICAST_FILTER_2
multicastAddress	6-byte address (all 0xFF will inactivate the filter)

## 11.7.1.9 WF\_SetRegionalDomain Function

### File

WFApi.h

### C

```
void WF_SetRegionalDomain(
    UINT8 regionalDomain
);
```

### Returns

None.

### Description

see tWFRegDomain enumerated types

Sets the regional domain on the MRF24WB0M. Note that this function does not overwrite the factory-set regional domain in FLASH. By default the MRF24WB0M will use the factory-set regional domain. It is invalid to call this function while in a connected state.

Valid values for the regional domain are:

- WF\_DOMAIN\_FCC
- WF\_DOMAIN\_IC
- WF\_DOMAIN\_ETSI
- WF\_DOMAIN\_SPAIN
- WF\_DOMAIN\_FRANCE
- WF\_DOMAIN\_JAPAN\_A
- WF\_DOMAIN\_JAPAN\_B

**Remarks**

None.

**Preconditions**

MACInit must be called first. This function must not be called while in a connected state.

**Parameters**

Parameters	Description
regionalDomain	Value to set the regional domain to

## 11.7.1.10 WF\_SetRtsThreshold Function

**File**

WFApi.h

**C**

```
void WF_SetRtsThreshold(
    UINT16 rtsThreshold
);
```

**Returns**

None.

**Description**

RTS Threshold Functions

Sets the RTS/CTS packet size threshold for when RTS/CTS frame will be sent. The default is 2347 bytes – the maximum for 802.11. It is recommended that the user leave the default at 2347 until they understand the performance and power ramifications of setting it smaller. Valid values are from 0 to WF\_RTS\_THRESHOLD\_MAX (2347).

**Remarks**

None.

**Preconditions**

MACInit must be called first.

**Parameters**

Parameters	Description
rtsThreshold	Value of the packet size threshold

## 11.7.1.11 tWFDeviceInfoStruct Structure

**File**

WFApi.h

**C**

```

struct tWFDeviceInfoStruct {
    UINT8 deviceType;
    UINT8 romVersion;
    UINT8 patchVersion;
};

```

**Members**

Members	Description
UINT8 deviceType;	MRF24WB0M_DEVICE_TYPE
UINT8 romVersion;	ROM version number
UINT8 patchVersion;	Patch version number

**Description**

used in WF\_GetDeviceInfo (see page 588)

## 11.7.1.12 WFMacStatsStruct Structure

**File**

WFApi.h

**C**

```

struct WFMacStatsStruct {
    UINT32 MibWEPEXcludeCtr;
    UINT32 MibTxBytesCtr;
    UINT32 MibTxMulticastCtr;
    UINT32 MibTxFailedCtr;
    UINT32 MibTxRtryCtr;
    UINT32 MibTxMultRtryCtr;
    UINT32 MibTxSuccessCtr;
    UINT32 MibRxDupCtr;
    UINT32 MibRxCtsSuccCtr;
    UINT32 MibRxCtsFailCtr;
    UINT32 MibRxAckFailCtr;
    UINT32 MibRxBytesCtr;
    UINT32 MibRxFragCtr;
    UINT32 MibRxMultCtr;
    UINT32 MibRxFCSERRCtr;
    UINT32 MibRxWEPUndecryptCtr;
    UINT32 MibRxFragAgedCtr;
    UINT32 MibRxMICFailureCtr;
};

```

**Members**

Members	Description
UINT32 MibWEPEXcludeCtr;	Number of frames received with the Protected Frame subfield of the Frame Control field set to zero and the value of dot11ExcludeUnencrypted causes that frame to be discarded.
UINT32 MibTxBytesCtr;	Total number of Tx bytes that have been transmitted
UINT32 MibTxMulticastCtr;	Number of frames successfully transmitted that had the multicast bit set in the destination MAC address.
UINT32 MibTxFailedCtr;	Number of Tx frames that failed due to the number of transmits exceeding the retry count.
UINT32 MibTxRtryCtr;	Number of times a transmitted frame needed to be retried
UINT32 MibTxMultRtryCtr;	Number of times a frame was successfully transmitted after more than one retransmission.
UINT32 MibTxSuccessCtr;	Number of Tx frames successfully transmitted.
UINT32 MibRxDupCtr;	Number of frames received where the Sequence Control field indicates a duplicate.

UINT32 MibRxCtsSuccCtr;	Number of CTS frames received in response to an RTS frame.
UINT32 MibRxCtsFailCtr;	Number of times an RTS frame was not received in response to a CTS frame.
UINT32 MibRxAckFailCtr;	Number of times an Ack was not received in response to a Tx frame.
UINT32 MibRxBytesCtr;	Total number of Rx bytes received.
UINT32 MibRxFragCtr;	Number of successful received frames (management or data)
UINT32 MibRxMultCtr;	Number of frames received with the multicast bit set in the destination MAC address.
UINT32 MibRxFCSErrCtr;	Number of frames received with an invalid Frame Checksum (FCS).
UINT32 MibRxWEPUndecryptCtr;	Number of frames received where the Protected Frame subfield of the Frame Control Field is set to one and the WEPOn value for the key mapped to the transmitter's MAC address indicates the frame should not have been encrypted.
UINT32 MibRxFragAgedCtr;	Number of times that fragments 'aged out', or were not received in the allowable time.
UINT32 MibRxMICFailureCtr;	Number of MIC failures that have occurred.

**Description**

This is record WFMacStatsStruct.

---

# 11.8 WF\_ProcessEvent

**Module**

Wi-Fi API (see page 537)

**Description**

There are several events that can occur on the MRF24WB0M that the host CPU may want to know about. All MRF24WB0M events go through the WF\_ProcessEvent() function described in the next section.

**Event Processing**

The WF\_ProcessEvent() function is how the host application is notified of events. This function will be called by the Wi-Fi host driver when an event occurs. This function should not be called directly by the host application. This function, located in WF\_Customize.c, should be modified by the user as needed. Since this function is called from the WiFi driver there are some restrictions – namely, one cannot call any Wi-Fi driver functions when inside WFProcessEvent. It is recommended that that customer simply set a flag for a specific event and handle it in the main loop. The framework for this function is shown below.

The prototype for this function is:

```
UINT16 WF_ProcessEvent(UINT8 event, UINT16 eventInfo);
```

There are two inputs to the function:

event	The event that occurred.
eventInfo	Additional information about the event. This is not applicable to all events.

The table below shows possible values that the event and eventInfo parameters can have. Note that event notification of some events can be optionally disabled via:

1. Bit mask eventNotificationAction in the tWFCAElements structure (see Wi-Fi Connection Algorithm (see page 554)), or
2. Function WF\_CASetEventNotificationAction() (see page 567).

event	eventInfo
WF_EVENT_CONNECTION_SUCCESSFUL	The connection attempt was successful. eventInfo: <ul style="list-style-type: none"> <li>• Always WF_NO_ADDITIONAL_INFO</li> </ul> (Optional event)
WF_EVENT_CONNECTION_FAILED	The connection attempt failed eventInfo: <ul style="list-style-type: none"> <li>• WF_JOIN_FAILURE</li> <li>• WF_AUTHENTICATION_FAILURE</li> <li>• WF_ASSOCIATION_FAILURE</li> <li>• WF_WEP_HANDSHAKE_FAILURE</li> <li>• WF_PSK_CALCULATION_FAILURE</li> <li>• WF_PSK_HANDSHAKE_FAILURE</li> <li>• WF_ADHOC_JOIN_FAILURE</li> <li>• WF_SECURITY_MISMATCH_FAILURE</li> <li>• WF_NO_SUITABLE_AP_FOUND_FAILURE</li> <li>• WF_RETRY_FOREVER_NOT_SUPPORTED_FAILURE</li> </ul> (Optional event)
WF_EVENT_CONNECTION_TEMPORARILY_LOST	An established connection was temporarily lost – the connection algorithm is attempting to reconnect. The eventInfo field indicates why the connection was lost. eventInfo: <ul style="list-style-type: none"> <li>• WF_BEACON_TIMEOUT</li> <li>• WF_DEAUTH_RECEIVED</li> <li>• WF_DISASSOCIATE_RECEIVED</li> </ul> (Optional event)
WF_EVENT_CONNECTION_PERMANENTLY_LOST	An established connection was permanently lost – the connection algorithm either ran out of retries or was configured not to retry. The eventInfo field indicates why the connection was lost. eventInfo: <ul style="list-style-type: none"> <li>• WF_BEACON_TIMEOUT</li> <li>• WF_DEAUTH_RECEIVED</li> <li>• WF_DISASSOCIATE_RECEIVED</li> </ul> This event can also be generated when WF_CMDisconnect (see page 577)() is called, in which case the eventInfo field has no meaning.  (Optional event)
WF_EVENT_CONNECTION_REESTABLISHED	A connection that was temporarily lost has been reestablished Always WF_NO_ADDITIONAL_INFO (Optional event)
WF_EVENT_SCAN_RESULTS_READY	The scan request initiated by calling WF_Scan (see page 579)() has completed and results can be read from the MRF24WB0M. eventInfo: Number of scan results

## 12.2 WF\_ProcessEvent() Framework

Below is the framework for `WF_ProcessEvent()`. Each case statement should be modified as needed to handle events the application is interested in.

```
void WF_ProcessEvent(UINT8 event, UINT16 eventInfo)
{
    switch (event)
    {
        case WF_EVENT_CONNECTION_SUCCESSFUL:
            /* Application code here */
            break;

        case WF_EVENT_CONNECTION_FAILED:
            /* Application code here */
            break;

        case WF_EVENT_CONNECTION_TEMPORARILY_LOST:
            /* Application code here */
            break;

        case WF_EVENT_CONNECTION_PERMANENTLY_LOST:
            /* Application code here */
            break;

        case WF_EVENT_FLASH_UPDATE_SUCCESSFUL:
            /* Application code here */
            break;

        case WF_EVENT_FLASH_UPDATE_FAILED:
            /* Application code here */
            break;

        case WF_EVENT_KEY_CALCULATION_COMPLETE:
            /* Application code here */
            break;

        case WF_EVENT_SCAN_RESULTS_READY:
            /* Application code here */
            break;

        case WF_EVENT_IE_RESULTS_READY:
            /* Application code here */
            break;

        default:
            WF_ASSERT(FALSE);
            break;
    }
}
```

# 11.9 Access Point Compatibility

## Module

Wi-Fi API (see page 537)

## Description

### Introduction

The MRF24WB0M has passed through Wi-Fi.org certification testing. Not all routers pass through Wi-Fi.org certification, and some are pre-configured in Greenfield modes. Further, users can set configurations that severely limit performance or prevent communications. This app note is intended to provide an on-going compatibility snapshot among a few of the most popular and market leading access points as well as a larger group of worldwide units. The test results will show the usability

of the Microchip Wi-Fi modules operating with the latest release of the Microchip TCPIP stack.

### Wi-Fi Alliance Testing

To carry the Wi-Fi Alliance logo, Wi-Fi products must successfully pass numerous tests, including compatibility testing. Wi-Fi compatibility testing is performed against 4 representative access points, with a subset of tests run against each of the access points. Devices are tested against these access points for characteristics such as connectivity, security, throughput, and a breadth of other specifications. Microchip Wi-Fi modules have successfully passed the Wi-Fi Alliance testing. The report is titled WFA7150 and is available at [http://certifications.wi-fi.org/pdf\\_certificate.php?cid=WFA7150](http://certifications.wi-fi.org/pdf_certificate.php?cid=WFA7150)

### Additional Wi-Fi Compatibility Testing

Wi-Fi technology is dramatically expanding the reach and applications of the internet to embedded devices. In many cases, Wi-Fi is new to the markets and applications it is reaching. As a result, Microchip feels it is important to raise the bar on compatibility testing, and education of the developer.

Microchip has thus adopted the Wi-Fi.org test bench for more generic Access Point testing. The goal of these tests is to ensure basic connectivity in multiple non-secure and secure scenarios with a global representation of top selling access points.

### Pass Criteria

The following tests are part of the current testing suite and must pass for the Access Point to be considered compatible.

- Following in conditions of no security, WEP40 and WEP104, WPA-PSK (TKIP), WPA2-PSK (AES)
- AP association, Iperf UDP upload/download, Iperf TCP upload/download, DHCP, ICMP ping

In many cases there are other modes that can be run with the Access Points and the user must take caution that if the mode is not listed, then compatibility is not necessarily guaranteed. These modes are usually Greenfield use, modes being deprecated by Wi-Fi.org, or cases of limiting the use of the Access Point for more private networking purposes and not for true Wi-Fi compatibility.

Examples of special modes not necessarily part of the results:

- WPA-PSK(AES) security: WPA-PSK security is defined as using TKIP. This is a mixed mode. This mode works if the AP just auto-detects and does not mix.
- WPA2-PSK (TKIP) security: WPA2-PSK security is defined as using 802.11i with AES. This is a mixed mode. This mode works if the AP just auto-detects and does not mix.
- 802.11g only, 802.11n only, 802.11g/n only: these are private network modes (cutting out mandatory support for 802.11b). These modes may work if basic rates are limited to 1&2mbps per 802.11.

### List of compatible Access Points:

- 2Wire 1701HG
- 2Wire 2701HG-B
- 3COM 3CRWER100-75
- 3COM WL-524
- Actiontec GT704-WG
- AirLink AR690W
- Belkin N1
- Belkin F5D7231-4

- Buffalo WHR-G125
- Buffalo WHR-HP-G54
- Corega CG-WLAPGMN
- Corega CG-WLBARGO
- D-Link DIR-655
- D-Link WBR-1310
- Dynex DX-WGRTR
- Level1 WBR-3408
- Linksys WRT150Nv1.1
- Linksys WRT310N
- Microsoft MN-700
- Netgear WGR614v9
- Netgear WGT624v3
- Netgear WNDR3300
- Netgear WPN824v2
- Proxim AP-700
- SMC Networks SMCWBR14T-G
- TP-Link TL-WR541G
- Westell B90-327W15-06

\*Note Tests Performed:

- Basic association with the AP (no security)
- Association with WEP security
- Association with WPA/WPA2-PSK security
- Ping test validation.

---

## 11.10 WiFi Tips and Tricks

### Module

Wi-Fi API (see page 537)

### Description

#### Tips for Setting up Routers for 802.11b Use

The purpose of this section is to describe the settings for the most typical AP configurable parameters to enable compatibility with the Microchip MRF24WB0Mx devices :

1. **DHCP Settings** - For DHCP on LAN side (where AP is DHCP server), set Router to Enable DHCP server. Set Client Lease time to be longer than the typical off time of the station to ensure that the IP address provided doesn't change each time the station is powered up. If an option for Always Broadcast is present for DHCP setup (broadcasts all DHCP responses to all clients), it should be disabled.
2. **Data Rate Settings** - Ensure that service rates include 802.11b. 802.11g or 802.11n only rates (green field) should be avoided, but mixed settings are usually acceptable. If a Basic Rate setting is defined, it should be set to 1 and 2MBPS only.
3. **SSID Broadcast** - Should typically be enabled so that the AP sends beacon frames containing the SSID. If disabled, ensure that Microchip Stack is set for Active Scanning.

4. **Channel Selection** - For debug purposes, it is typical to use a fixed channel instead of Auto Channel Selection. If a fixed channel has been selected for the MRF24 Station, select the corresponding channel for the AP.
5. **Multicast Passthrough** - If using multicast features (ZeroConfig for instance) ensure that the Router is configured to enable forwarding of Multicast packets.
6. **Beacon Interval** - Set the value for the time interval between AP beacons, typical is 100msec. For lower power, this can be set to a smaller value, say 30mS, if the DTIM interval is correspondingly increased.
7. **RTS Threshold** - Set the value for the frame size above which RTS/CTS will be used, typical is 2347.
8. **Fragmentation Threshold** - Set the value for the frame size above which packets will be fragmented, typical is 2346.
9. **DTIM Interval** - Set the value for Delivery Traffic Indication Message Interval, typical is 3 if the Beacon Interval is set for 100mSec. For lower power with the MRF24WB0M, if the Beacon Interval is set to 30mS, then the DTIM should be set to 100 to allow 300mS DTIM Interval.
10. **WLAN Partition (or AP Isolation)**- Prevents AP clients from communicating to each other, typically disabled.
11. **WMM Enable** - Allows wireless multimedia traffic, disable unless necessary for other AP services.
12. **Short Guard Interval (GI)** - Lowers the guard interval between frames, disable unless necessary for other AP services.
13. **WiFi Protected Setup (WPS)** - Enables WPS device discovery, disable unless necessary for other AP services.
14. **Frame Burst** - Enables higher wireless packet throughput, disable unless necessary for other AP services. This may be called turbo, or other marketing terms.
15. **CTS Protection Mode** – Improves reliability of 802.11g traffic, disable unless necessary for other AP services.
16. **Key Entry** – Security can be entered with either a numerical key or an ASCII passphrase. Ensure you enter what the AP expects. If just starting, it is best to have another station like a laptop to validate what the AP is expecting.

---

## 11.11 Hot Topics

### Module

Wi-Fi API (see page 537)

### Description

Host Controlled Connection Manager Clarifications

The following clarifications are to be noted for use of the device with Microchip TCPIP Stack versions unless otherwise noted.

#### 1. Null String ESSID

It is possible to call `WF_CMConnect` (see page 576)(cpld) with a cpld of zero. If this happens, the connection manager can use erroneous values for the SSID, Network Mode, Security configuration, etc. which will cause the module to connect (see page 165) to a wrong AP or not connect (see page 165) at all. The only valid values that can be used for connection profile references are 1 and 2 (assuming that the `WF_CPCreate` (see page 541)(&cpld) succeeded in creating these profile references prior to the attempted connection).

#### *Work around:*

When creating a connection profile, verify that the profile number returned is always either 1 or 2. If the returned value is 0, delete the profile and recreate it. When connecting with `WF_CMConnect` (see page 576)(cpld), ensure that only a valid profile number previously returned from `WF_CPCreate` (see page 541)(&cpld) is used.

#### 2. Management scan message conflict

Management messages must always return successful or it causes an assert in the host driver. An unsuccessful

management message can occur when the connection retry is enabled (`MY_DEFAULT_LIST_RETRY_COUNT>0`) causing the device to be scanning due to a dropped connection, and then a disconnect, or connect (see page 165), or scan command is sent.

*Work around:*

If you are controlling connect (see page 165)/reconnect from the host actively, then disable all firmware retry by using “no scan retry” and “no de-authorization action”.

a. To disable Scan Retry

`WF_CASetListRetryCount (see page 568)(MY_DEFAULT_LIST_RETRY_COUNT);` should be 0

b. To disable De-authorization action

`WF_CASetDeauthAction (see page 566)(WF_DO_NOT_ATTEMPT_TO_RECONNECT);`

c. To disable De-authentication action

`WF_CASetBeaconTimeoutAction (see page 564)(WF_DO_NOT_ATTEMPT_TO_RECONNECT);`

d. Use “Connect” only on “permanent loss” or “connection failure”.

e. To do a Scan, first check the firmware state first by using `WF_CMGetConnectionState (see page 577)()`

i. If the return state is `WF_CSTATE_NOT_CONNECTED` (or `WF_CSTATE_CONNECTION_PERMANENTLY_LOST`), then this means firmware is in IDLE, so host can issue host scan safely

ii. If the return state is `WF_CSTATE_CONNECTED_INFRASTRUCTURE`, then this means firmware is in CONNECTED. In this case a scan command can be issued but a watchdog timer must be used to time for conflict. Also, ensure the management timer is set for at least 0.4seconds per channel scanned to prevent queued Tx buffer requests from timing out.

iii. If return state is `WF_CSTATE_CONNECTION_IN_PROGRESS` ( or `WF_CSTATE_RECONNECTION_IN_PROGRESS`), then this means firmware is in the middle of connection process and a scan must not be initiated.

f. If “Disconnect” function is desired, a watchdog timer needs to be used to address the case where a conflict occurs with an over the air disassociate or deauthorize.

g. For watchdog timing, advised timing is 2x the management packet timeout (that is, use 4seconds unless the management timeout has been increased).

b. If you are only using the firmware retry and not doing ANY connection management (scan, connect (see page 165), idle, etc.) then you can use `MY_DEFAULT_LIST_RETRY_COUNT>0` or `retry forever (MY_DEFAULT_LIST_RETRY_COUNT=255)`. If you lose connection, you can reconnect using the “connect (see page 165)” API. Do not use “Disconnect”.

a. If “Disconnect” function is desired, a watchdog timer needs to be used to address the case where a conflict occurs with an over the air disassociate or deauthorize.

## Index

—  
 \_checkIpSrvrResponse variable 195  
 \_LoadFATRecord function 282  
 \_MD5\_k variable 204  
 \_MD5\_r variable 204  
 \_NBNS\_HEADER structure 287  
 \_SNMPDuplexInit function 330  
 \_SNMPGet function 330  
 \_SNMPGetTxOffset macro 330  
 \_SNMPPut function 330  
 \_SNMPSetTxOffset macro 331  
 \_TFTP\_ACCESS\_ERROR enumeration 495  
 \_TFTP\_FILE\_MODE enumeration 495  
 \_TFTP\_RESULT enumeration 496  
 \_tftpError variable 505  
 \_tftpFlags variable 505  
 \_tftpRetries variable 506  
 \_TFTPSendAck function 506  
 \_TFTPSendFileName function 506  
 \_TFTPSendROMFileName function 507  
 \_tftpSocket variable 507  
 \_tftpStartTick variable 507  
 \_tftpState variable 507  
 \_updateIpSrvrResponse variable 195  
 \_Validate function 283

## A

accept function 163  
 Access Point Compatibility 597  
 Accessing the Demo Application 73  
 ACK macro 468  
 activeUDPSocket variable 532  
 Additional Features 145  
 Address 141  
 Advanced MPFS2 Settings 59  
 AF\_INET macro 164  
 AGENT\_NOTIFY\_PORT macro 331  
 Announce 149  
 Announce Stack Members 150

AnnounceIP function 150  
 APP\_CONFIG Structure 132  
 appendZeroToOID variable 331  
 ARP 151  
     Types 160  
 ARP Internal Members 157  
 ARP Public Members 151  
 ARP Stack Members 156  
 arp\_app\_callbacks structure 155  
 ARP\_IP macro 159  
 ARP\_OPERATION\_REQ macro 159  
 ARP\_OPERATION\_RESP macro 159  
 ARP\_PACKET structure 160  
 ARP\_REQ macro 155  
 ARP\_RESP macro 155  
 ARPDeRegisterCallbacks function 153  
 ARPInit function 156  
 ARPisResolved function 153  
 ARPPProcess function 156  
 ARPPut function 158  
 ARPRegisterCallbacks function 154  
 ARPResolve function 152  
 ARPSendPkt function 154  
 ASN\_INT macro 331  
 ASN\_NULL macro 332  
 ASN\_OID macro 332  
 Authentication 84  
 Available Demos 81

## B

Base64Decode function 208  
 Base64Encode function 208  
 Berkeley (BSD) Sockets 161  
 BerkeleySocketInit function 175  
 bForceUpdate variable 194  
 bind function 164  
 Bootloader Design 114  
 BSD Sockets 148  
 BSD Wrapper Internal Members 176  
 BSD Wrapper Public Members 162  
 BSD Wrapper Stack Members 175  
 BSD\_SCK\_STATE enumeration 176

- BSDSocket structure 164
  - BSDSocketArray variable 177
  - btohexa\_high function 209
  - btohexa\_low function 209
  - Building MPFS2 Images 58
- C**
- Cache variable 159
  - CalcIPBufferChecksum function 210
  - CalcIPChecksum function 210
  - CalculateFinishedHash function 392
  - Clock Frequency 136
  - closesocket function 165
  - CloseSocket function 468
  - COMMUNITY\_TYPE enumeration 317
  - Configure your WiFi Access Point 70
  - Configuring the Stack 136
  - Configuring WiFi Security 74
  - connect function 165
  - Connecting to the Network 72
  - Connection Algorithm Internal Members 573
  - Connection Algorithm Public Members 554
  - Connection Manager Public Members 576
  - Connection Profile Internal Members 552
  - Connection Profile Public Members 540
  - Cookies 86
  - Cooperative Multitasking 133
  - CRPeriod variable 305
  - curHTTP variable 234
  - curHTTPID variable 246
- D**
- DATA\_TYPE enumeration 332
  - DATA\_TYPE\_INFO structure 333
  - DATA\_TYPE\_TABLE\_SIZE macro 333
  - dataTypeTable variable 333
  - Daughter Boards 62
  - DDNS\_CHECKIP\_SERVER macro 195
  - DDNS\_DEFAULT\_PORT macro 196
  - DDNS\_POINTERS structure 188
  - DDNS\_SERVICES enumeration 189
  - DDNS\_STATUS enumeration 189
  - DDNSClient variable 190
  - DDNSData variable 90
  - DDNSForceUpdate function 190
  - DDNSGetLastIP function 191
  - DDNSGetLastStatus function 191
  - DDNSInit function 192
  - ddnsServiceHosts variable 194
  - ddnsServicePorts variable 194
  - DDNSSetService function 191
  - DDNSTask function 192
  - Demo App 81
  - Demo App MDD 129
  - Demo Compatibility Table 77
  - Demo Information 77
  - Demo Modules 82
  - Directory Structure 1
  - DiscoveryTask function 150
  - DNS Client 178
  - DNS Internal Members 182
  - DNS Public Members 178
  - DNS\_HEADER structure 186
  - DNS\_PORT macro 184
  - DNS\_TIMEOUT macro 184
  - DNS\_TYPE\_A macro 181
  - DNS\_TYPE\_MX macro 182
  - DNSBeginUsage function 179
  - DNSDiscardName function 186
  - DNSEndUsage function 179
  - DNSHostName variable 184
  - DNSHostNameROM variable 184
  - DNSIsResolved function 181
  - DNSPutROMString function 183
  - DNSPutString function 183
  - DNSResolve function 180
  - DNSResolveROM function 180
  - dwInternalTicks variable 514
  - dwLastUpdateTick variable 371
  - dwLFSRRandSeed variable 224
  - dwSNTPSeconds variable 371
  - dwUpdateAt variable 194
  - Dynamic DNS Client 187
  - Dynamic DNS Internal Members 193

Dynamic DNS Public Members 187  
 Dynamic DNS Stack Members 192  
 Dynamic Variables 83

## E

E-mail (SMTP) Demo 91  
 ENC28J60 Config 137  
 ENCX24J600 Config 138  
 Energy Monitoring 130  
 Explorer 16 and PIC32 Starter Kit 66  
 External Storage 136  
 ExtractURLFields function 211

## F

fatCache variable 283  
 fatCacheID variable 284  
 FIN macro 468  
 FindEmailAddress function 305  
 FindMatchingSocket function 469, 532  
 FindOIDsInRequest function 334  
 FindROMEmailAddress function 306  
 Flags variable 185  
 FormatNetBIOSName function 214  
 Forms using GET 84  
 Forms using POST 85

## G

gAutoPortNumber variable 177  
 GenerateHashRounds function 393  
 GenerateRandomDWORD function 214  
 GenerateSessionKeys function 393  
 Generating Server Certificates 374  
 Generic TCP Client 92  
   Variables 94  
 Generic TCP Server 95  
   Macros 96  
 GENERIC\_TRAP\_NOTIFICATION\_TYPE enumeration 316  
 GenericTCPClient function 93  
 GenericTCPServer function 96  
 GET\_BULK\_REQUEST macro 334  
 GET\_NEXT\_REQUEST macro 334  
 GET\_REQUEST macro 334

GET\_RESPONSE macro 334  
 GetDataTypeInfo function 337, 348  
 gethostname function 166  
 GetNextLeaf function 348  
 GetOIDStringByAddr function 349  
 GetOIDStringByID function 349  
 getSnmpV2GenTrapOid function 356  
 GetTickCopy function 515  
 Getting Help 1  
 Getting Started 62  
 getZeroInstance variable 365  
 gGenericTrapNotification variable 319  
 gOIDCorrespondingSnmpMibID variable 319  
 Google PowerMeter 129  
 Google PowerMeter EZConfig 130  
 gSendTrapFlag variable 318  
 gSendTrapSMstate variable 111  
 gSetTrapSendFlag variable 318  
 gSnmpNonMibReclInfo variable 365  
 gSNMPv3ScopedPduDataPos variable 365  
 gSNMPv3ScopedPduRequestBuf variable 365  
 gSNMPv3ScopedPduResponseBuf variable 366  
 gSnmpv3UserSecurityName variable 111  
 gSpecificTrapNotification variable 319

## H

HandlePossibleTCPDisconnection function 177  
 HandleTCPSEg function 469  
 Hardware Configuration 136  
 Hardware Setup 62  
 Hash Table Filter Entry Calculator 60  
 HASH\_SUM structure 200  
 HASH\_TYPE enumeration 204  
 HashAddData function 197  
 HashAddROMData function 197  
 Hashes 196  
 Hashes Internal Members 203  
 Hashes Public Members 196  
 Hashes Stack Members 201  
 hCurrentTCP variable 470  
 Helpers 206  
   Functions 222

- Variables 224
  - Helpers Public Members 207
  - hexatob function 215
  - hMPFS variable 335
  - HOST\_TO\_PING macro 98
  - Hot Topics 600
  - How the Stack Works 131
  - HSEnd function 394
  - HSGet function 394
  - HSGetArray function 395
  - HSGetWord function 395
  - HSPut function 396
  - HSPutArray function 396
  - HSPutROMArray function 397
  - HSPutWord function 397
  - HSStart function 398
  - HTTP Configuration 109
  - HTTP\_CACHE\_LEN macro 247
  - HTTP\_CONN structure 234
  - HTTP\_FILE\_TYPE enumeration 247
  - HTTP\_IO\_RESULT enumeration 235
  - HTTP\_MAX\_DATA\_LEN macro 248
  - HTTP\_MAX\_HEADER\_LEN macro 248
  - HTTP\_MIN\_CALLBACK\_FREE macro 248
  - HTTP\_PORT macro 248
  - HTTP\_READ\_STATUS enumeration 235
  - HTTP\_STATUS enumeration 248
  - HTTP\_STUB structure 249
  - HTTP\_TIMEOUT macro 250
  - HTTP2 Authentication 230
  - HTTP2 Compression 232
  - HTTP2 Cookies 232
  - HTTP2 Dynamic Variables 225
  - HTTP2 Features 225
  - HTTP2 Form Processing 227
  - HTTP2 Internal Members 245
  - HTTP2 Public Members 233
  - HTTP2 Server 224
  - HTTP2 Stack Members 244
  - HTTPCheckAuth function 235
  - httpContentTypes variable 250
  - HTTPExecuteGet function 236
  - HTTPExecutePost function 237
  - httpFileExtensions variable 250
  - HTTPGetArg function 238
  - HTTPGetROMArg function 239
  - HTTPHeaderParseAuthorization function 250
  - HTTPHeaderParseContentLength function 251
  - HTTPHeaderParseCookie function 251
  - HTTPHeaderParseLookup function 252
  - HTTPIncFile function 252
  - HTTPInit function 244
  - HTTPLoadConn function 253
  - HTTPMPFSUpload function 253
  - HTTPNeedsAuth function 239
  - HTTPPostConfig function 88
  - HTTPPostDDNSConfig function 88
  - HTTPPostEmail function 89
  - HTTPPostLCD function 89
  - HTTPPostMD5 function 90
  - HTTPPostSNMPCommunity function 87
  - HTTPPrint\_varname function 240
  - HTTPProcess function 254
  - HTTPReadPostName function 241
  - HTTPReadPostPair macro 242
  - HTTPReadPostValue function 242
  - HTTPReadTo function 254
  - HTTPRequestHeaders variable 255
  - HTTPResponseHeaders variable 255
  - HTTPS\_PORT macro 255
  - HTTPSendFile function 256
  - HTTPServer function 245
  - httpStubs variable 256
  - HTTPURLDecode function 243
  - HW\_ETHERNET macro 159
- |
- ICMP 257
  - ICMP Internal Members 262
  - ICMP Public Members 258
  - ICMP\_PACKET structure 263
  - ICMP\_TIMEOUT macro 264
  - ICMPBeginUsage function 258
  - ICMPEndUsage function 261

ICMPFlags variable 263  
 ICMPGetReply function 260  
 ICMPProcess function 262  
 ICMPSendPing function 259  
 ICMPSendPingToHost function 259  
 ICMPSendPingToHostROM function 260  
 ICMPSendPingToHostROM macro 261  
 ICMPState variable 263  
 ICMPTimer variable 264  
 ifconfig Commands 123  
 in\_addr structure 167  
 INADDR\_ANY macro 167  
 INDEX\_INFO union 335  
 Initialization 132  
 Initialization Structure 147  
 INOUT\_SNMP\_PDU enumeration 363  
 Internet Bootloader 114  
 Internet Radio 120  
 Introduction 1  
 INVALID\_SOCKET macro 437  
 INVALID\_TCP\_PORT macro 168  
 INVALID\_UDP\_PORT macro 518  
 INVALID\_UDP\_SOCKET macro 518  
 IP Address 142  
 IP\_ADDR\_ANY macro 168  
 iperf Example 125  
 IPPROTO\_IP macro 168  
 IPPROTO\_TCP macro 168  
 IPPROTO\_UDP macro 168  
 IS\_AGENT\_PDU macro 335  
 IS\_ASN\_INT macro 336  
 IS\_ASN\_NULL macro 336  
 IS\_GET\_NEXT\_REQUEST macro 336  
 IS\_GET\_REQUEST macro 336  
 IS\_GET\_RESPONSE macro 336  
 IS\_OCTET\_STRING macro 337  
 IS\_OID macro 337  
 IS\_SET\_REQUEST macro 337  
 IS\_SNMPV3\_AUTH\_STRUCTURE macro 366  
 IS\_STRUCTURE macro 338  
 IS\_TRAP macro 338  
 IsASNNull function 338

isBufferUsed variable 398  
 isHashUsed variable 399  
 isMPFSLocked variable 279  
 isStubUsed variable 399  
 IsValidCommunity function 349  
 IsValidInt function 349  
 IsValidLength function 345, 350  
 IsValidOID function 350  
 IsValidPDU function 350  
 IsValidStructure function 350  
 iwconfig Commands 122  
 iwpriv Commands 124

## L

lastBlock variable 204  
 lastFailure variable 91  
 lastKnownIP variable 194  
 LastPutSocket variable 533  
 lastRead variable 280  
 lastStatus variable 195  
 lastSuccess variable 91  
 leftRotateDWORD function 215  
 leftRotateDWORD macro 216  
 LFSRRand function 222  
 LFSRSeedRand function 223  
 listen function 169  
 LoadOffChip function 400  
 LOCAL\_PORT\_END\_NUMBER macro 470  
 LOCAL\_PORT\_START\_NUMBER macro 470  
 LOCAL\_UDP\_PORT\_END\_NUMBER macro 533  
 LOCAL\_UDP\_PORT\_START\_NUMBER macro 533  
 LowLevel\_CAGetElement function 574  
 LowLevel\_CASetElement function 574  
 LowLevel\_CPGetElement function 553  
 LowLevel\_CPSetElement function 553

## M

MAC Address 141  
 Main File 132  
 Main Loop 132  
 masks variable 399  
 MAX\_FILE\_NAME\_LEN macro 280

- MAX\_REG\_APPS macro 155
  - MAX\_TELNET\_CONNECTIONS macro 481
  - MAX\_TRY\_TO\_SEND\_TRAP macro 112
  - MD5AddData function 203
  - MD5AddROMData function 201
  - MD5Calculate function 198
  - MD5HashBlock function 206
  - MD5Initialize function 199
  - Memory Allocation 146
  - Memory Usage 54
  - MIB Browsers 100
  - MIB Files 99
  - MIB\_INFO union 338
  - Microchip TCP/IP Discoverer 60
  - MPFS\_FAT\_RECORD structure 283
  - MPFS\_HANDLE type 267
  - MPFS\_INVALID macro 267
  - MPFS\_INVALID\_FAT macro 284
  - MPFS\_INVALID\_HANDLE macro 267
  - MPFS\_PTR type 280
  - MPFS\_SEEK\_MODE enumeration 267
  - MPFS\_STUB structure 280
  - MPFS\_WRITE\_PAGE\_SIZE macro 281
  - MPFS2 265
  - MPFS2 Command Line Options 59
  - MPFS2 Internal Members 278
  - MPFS2 Public Members 266
  - MPFS2 Stack Members 278
  - MPFS2 Utility 57
  - MPFS2\_FLAG\_HASINDEX macro 281
  - MPFS2\_FLAG\_ISZIPPED macro 281
  - MPFSClose function 268
  - MPFSFormat function 268
  - MPFSGet function 269
  - MPFSGetArray function 269
  - MPFSGetBytesRem function 270
  - MPFSGetEndAddr function 270
  - MPFSGetFilename function 271
  - MPFSGetFlags function 271
  - MPFSGetID function 272
  - MPFSGetLong function 272
  - MPFSGetMicrotime function 273
  - MPFSGetPosition function 273
  - MPFSGetSize function 273
  - MPFSGetStartAddr function 274
  - MPFSGetTimestamp function 274
  - MPFSInit function 278
  - MPFSOpen function 275
  - MPFSOpenID function 275
  - MPFSOpenROM function 276
  - MPFSPutArray function 276
  - MPFSPutEnd function 277
  - MPFSSeek function 277
  - MPFSStubs variable 281
  - MPFSTell macro 282
  - msgSecrtyParamLenOffset variable 366
  - MutexVar variable 503
  - MySocket variable 306
  - MyTCB variable 470
  - MyTCBStub variable 471
- ## N
- NBNS 284
  - NBNS Stack Members 285
  - NBNS\_HEADER structure 287
  - NBNS\_PORT macro 287
  - NBNSGetName function 285
  - NBNSPutName function 286
  - NBNSTask function 286
  - Network Management (SNMP) Server 98
    - Functions 110
    - Macros 112
    - Variables 111
  - NextPort variable 480
  - NOTIFY\_COMMUNITY\_LEN macro 326
  - NTP\_EPOCH macro 372
  - NTP\_FAST\_QUERY\_INTERVAL macro 372
  - NTP\_PACKET structure 370
  - NTP\_QUERY\_INTERVAL macro 372
  - NTP\_REPLY\_TIMEOUT macro 372
  - NTP\_SERVER macro 372
  - NTP\_SERVER\_PORT macro 373
  - numFiles variable 284

**O**

OCTET\_STRING macro 339  
 OID\_INFO structure 339  
 OID\_MAX\_LEN macro 325  
 OIDLookup function 351

**P**

PDU\_INFO structure 340  
 Performance Test Internal Members 289  
 Performance Test Stack Members 287  
 Performance Tests 287  
 PERFORMANCE\_PORT macro 290  
 Peripheral Usage 54  
 PIC18 Explorer 65  
 PIC18F97J60 Config 140  
 PIC24FJ256DA210 Dev Board 69  
 PIC32MX7XX Config 140  
 PICDEM.net 2 63  
 Ping (ICMP) Demo 96  
     Macros 98  
 PingDemo function 97  
 Power Save Internal Members 586  
 Power Save Public Members 583  
 ProcessGetBulkVar function 356  
 ProcessGetNextVar function 357  
 ProcessGetSetHeader function 351  
 ProcessGetVar function 357  
 ProcessHeader function 352  
 ProcessSetVar function 352  
 ProcessSnmpv3MsgData function 357  
 ProcessVariables function 352  
 Programming and First Run 69  
 Protocol Configuration 143  
 Protocol Macros and Files 144  
 PSH macro 471  
 ptrHS variable 399  
 PutHeadersState variable 306

**R**

ReadMIBRecord function 353

ReadProgramMemory function 282  
 Reboot 311  
 Reboot Stack Members 311  
 REBOOT\_PORT macro 312  
 REBOOT\_SAME\_SUBNET\_ONLY macro 312  
 RebootTask function 311  
 RecordType variable 185  
 recv function 169  
 recvfrom function 170  
 reg\_apps variable 160  
 Release Notes 6  
 RemoteURL variable 94  
 Replace function 216  
 REPORT\_RESPONSE macro 367  
 Required Files 131  
 reqVarErrStatus structure 340  
 RESERVED\_HTTP\_MEMORY macro 257  
 RESERVED\_SSL\_MEMORY macro 399  
 ResolvedInfo variable 185  
 ResponseCode variable 307  
 ROMStringToIPAddress function 217  
 ROMStringToIPAddress macro 218  
 RST macro 471  
 RTOS 135  
 RX\_PERFORMANCE\_PORT macro 290  
 RXParserState variable 307

**S**

SaveOffChip function 400  
 Scan Public Members 578  
 send function 171  
 SendNotification function 110  
 SendPowerModeMsg function 586  
 SendTCP function 471  
 SENDTCP\_KEEP\_ALIVE macro 472  
 SENDTCP\_RESET\_TIMERS macro 472  
 sendto function 171  
 SERVER\_PORT macro 96  
 ServerName variable 94  
 ServerPort variable 94  
 SET\_REQUEST macro 341  
 SetErrorStatus function 341

- SetEventNotificationMask function 575
- SetPowerSaveState function 586
- SHA1AddData function 202
- SHA1AddROMData function 202
- SHA1Calculate function 199
- SHA1HashBlock function 205
- SHA1Initialize function 199
- Silicon Solutions 56
- sktHTTP macro 244
- SM\_HTTP2 enumeration 256
- SM\_SSL\_RX\_SERVER\_HELLO enumeration 401
- smDNS variable 185
- smHTTP macro 257
- SMTP Client 291
- SMTP Client Examples 291
- SMTP Client Internal Members 304
- SMTP Client Long Message Example 292
- SMTP Client Public Members 294
- SMTP Client Short Message Example 291
- SMTP Client Stack Members 303
- SMTP\_CONNECT\_ERROR macro 295
- SMTP\_POINTERS structure 295
- SMTP\_PORT macro 308
- SMTP\_RESOLVE\_ERROR macro 297
- SMTP\_SERVER\_REPLY\_TIMEOUT macro 308
- SMTP\_SUCCESS macro 297
- SMTPBeginUsage function 297
- SMTPClient variable 298
- SMTPDemo function 92
- SMTPEndUsage function 298
- SMTPFlags variable 308
- SMTPFlush function 298
- SMTPIsBusy function 299
- SMTPIsPutReady function 299
- SMTPPut function 300
- SMTPPutArray function 300
- SMTPPutDone function 301
- SMTPPutROMArray function 301
- SMTPPutROMString function 302
- SMTPPutString function 302
- SMTPSendMail function 303
- SMTPServer variable 308
- SMTPState variable 309
- SMTPTask function 304
- smUpload variable 507
- SNMP 312
  - Functions 355
  - Macros 366
  - Types 363
  - Variables 364
- SNMP Internal Members 327
- SNMP Operations 104
- SNMP Public Members 314
- SNMP Stack Members 354
- SNMP Traps 106
- SNMP\_ACTION enumeration 316
- SNMP\_AGENT\_PORT macro 341
- SNMP\_BIB\_FILE\_NAME macro 342
- SNMP\_COMMUNITY\_MAX\_LEN macro 325
- SNMP\_COUNTER32 macro 342
- SNMP\_END\_OF\_VAR macro 325
- SNMP\_ERR\_STATUS enumeration 342
- SNMP\_GAUGE32 macro 343
- SNMP\_ID type 324
- SNMP\_INDEX type 324
- SNMP\_INDEX\_INVALID macro 326
- SNMP\_IP\_ADDR macro 343
- SNMP\_MAX\_MSG\_SIZE macro 367
- SNMP\_MAX\_NON\_REC\_ID\_OID macro 367
- SNMP\_NMS\_PORT macro 344
- SNMP\_NOTIFY\_INFO structure 344
- SNMP\_NSAP\_ADDR macro 344
- SNMP\_OPAQUE macro 345
- SNMP\_START\_OF\_VAR macro 325
- SNMP\_STATUS union 345
- SNMP\_TIME\_TICKS macro 345
- SNMP\_V1 macro 346
- SNMP\_V2C macro 346
- SNMP\_V3 macro 367
- SNMP\_VAL union 317
- SNMPAgentSocket variable 346
- SNMPCheckIfPvtMibObjRequested function 353
- SNMPGetExactIndex function 357
- SNMPGetNextIndex function 324

---

SNMPGetTimeStamp function 111	SOCK_STREAM macro 172
SNMPGetVar function 322	sockaddr structure 172
SNMPIdRecrdValidation function 358	SOCKADDR type 173
SNMPInit function 354	sockaddr_in structure 173
SNMPIsNotifyReady function 322	SOCKADDR_IN type 173
SNMPIsValidSetLen function 359	socket function 174
SNMPNONMIBRECDINFO structure 364	SOCKET type 174
SNMPNotify function 320	Socket Types 146
SNMPNotifyInfo variable 346	SOCKET_CNXXN_IN_PROGRESS macro 174
SNMPNotifyPrepare function 323	SOCKET_DISCONNECTED macro 175
snmpReqVarErrStatus variable 347	SOCKET_ERROR macro 175
SNMPRxOffset variable 347	SOCKET_INFO structure 459
SNMPSendTrap function 319	Sockets 146
SNMPSetVar function 321	SocketWithRxData variable 533
SNMPStatus variable 347	Software 57
SNMPTask function 354	SSL 373
SNMPTxOffset variable 347	SSL Internal Members 387
Snmpv3AESDecryptRxedScopedPdu function 359	SSL Public Members 377
Snmpv3BufferPut function 359	SSL Stack Members 381
Snmpv3FormulateEngineID function 360	SSL_ALERT macro 401
Snmpv3GetAuthEngineTime function 360	SSL_ALERT_LEVEL enumeration 401
Snmpv3GetBufferData function 360	SSL_APPLICATION macro 402
Snmpv3InitializeUserDataBase function 360	SSL_BASE_BUFFER_ADDR macro 402
SNMPV3MSGDATA structure 364	SSL_BASE_HASH_ADDR macro 402
Snmpv3MsgProcessingModelProcessPDU function 361	SSL_BASE_KEYS_ADDR macro 402
Snmpv3Notify function 361	SSL_BASE_SESSION_ADDR macro 403
Snmpv3ScopedPduProcessing function 361	SSL_BASE_STUB_ADDR macro 403
Snmpv3TrapScopedpdu function 361	SSL_BUFFER union 403
Snmpv3UserSecurityModelProcessPDU function 362	SSL_BUFFER_SIZE macro 403
Snmpv3UsmAesEncryptDecryptInitVector function 362	SSL_BUFFER_SPACE macro 404
Snmpv3UsmOutMsgAuthenticationParam function 362	SSL_CERT variable 404
Snmpv3ValidateEngineId function 362	SSL_CERT_LEN variable 404
Snmpv3ValidateSecNameAndSecLvl function 363	SSL_CHANGE_CIPHER_SPEC macro 404
Snmpv3ValidateSecurityName function 363	SSL_HANDSHAKE macro 405
SNMPValidateCommunity function 320	SSL_HASH_SIZE macro 405
SNTP Client 368	SSL_HASH_SPACE macro 405
SNTP Client Internal Members 369	SSL_INVALID_ID macro 377
SNTP Client Public Members 368	SSL_KEYS structure 405
SNTP Client Stack Members 369	SSL_KEYS_SIZE macro 406
SNTPClient function 369	SSL_KEYS_SPACE macro 406
SNTPGetUTCSeconds function 368	SSL_MESSAGES enumeration 406
SOCK_DGRAM macro 172	SSL_MIN_SESSION_LIFETIME macro 386

---

---

SSL_PKEY_INFO structure 380	SSLRxClientKeyExchange function 421
SSL_RSA_EXPORT_WITH_ARCFOUR_40_MD5 macro 407	SSLRxFinished function 421
SSL_RSA_LIFETIME_EXTENSION macro 387	SSLRxHandshake function 422
SSL_RSA_WITH_ARCFOUR_128_MD5 macro 407	SSLRxRecord function 422
SSL_SESSION structure 408	SSLRxServerCertificate function 423
SSL_SESSION_SIZE macro 408	SSLRxServerHello function 423
SSL_SESSION_SPACE macro 408	sslSession variable 424
SSL_SESSION_STUB structure 409	sslSessionID variable 424
SSL_SESSION_TYPE enumeration 409	SSLSessionMatchID function 424
SSL_STATE enumeration 382	SSLSessionMatchIP function 425
SSL_STUB structure 409	SSLSessionNew function 425
SSL_STUB_SIZE macro 411	sslSessionStubs variable 426
SSL_STUB_SPACE macro 411	SSLSessionSync function 426
SSL_SUPPLEMENTARY_DATA_TYPES enumeration 380	SSLSessionUpdated macro 427
SSL_VERSION macro 411	sslSessionUpdated variable 427
SSL_VERSION_HI macro 411	SSLStartPartialRecord function 427
SSL_VERSION_LO macro 411	SSLStartSession function 380
SSLBufferAlloc function 412	sslStub variable 428
SSLBufferFree function 412	SSLStubAlloc function 428
sslBufferID variable 413	SSLStubFree function 428
SSLBufferSync function 413	sslStubID variable 429
SSLFinishPartialRecord macro 413	SSLStubSync function 429
SSLFlushPartialRecord macro 414	SSLTerminate function 430
sslHash variable 414	SSLTxCCSFin function 430
SSLHashAlloc function 414	SSLTxClientHello function 431
SSLHashFree function 415	SSLTxClientKeyExchange function 431
sslHashID variable 415	SSLTxMessage function 432
SSLHashSync function 415	SSLTxRecord function 432
SSLInit function 382	SSLTxServerCertificate function 433
sslKeys variable 416	SSLTxServerHello function 433
sslKeysID variable 416	SSLTxServerHelloDone function 434
SSLKeysSync function 416	Stack API 149
SSLMACAdd function 417	Stack Architecture 131
SSLMACBegin function 417	Stack Performance 54
SSLMACCalc function 417	Standalone Commands 121
SSLPeriodic function 382	StaticVars variable 264
SSLRSAOperation function 418	strAuthenticated variable 484
sslRSAStubID variable 418	strDisplay variable 484
SSLRxAlert function 418	strGoodBye variable 484
SSLRxAntiqueClientHello function 419	stricmppgm2ram function 218
SSLRxCCS function 419	StringToIPAddress function 218
SSLRxClientHello function 420	strnchr function 219

---

- strPassword variable 485
  - strSpaces variable 484
  - strTitle variable 485
  - STRUCTURE macro 347
  - strupr function 219
  - SW License Agreement 3
  - SwapARPPacket function 158
  - swapl function 220
  - swaps function 220
  - SwapTCPHeader function 472
  - SYN macro 473
  - SyncTCB function 473
  - SyncTCBStub macro 473
  - SYNQueue variable 473
- T**
- TCB structure 460
  - TCB\_STUB structure 461
  - TCBStubs variable 473
  - TCP 434
    - Variables 480
  - TCP Internal Members 466
  - TCP Public Members 436
  - TCP Stack Members 458
  - TCP/IP Configuration Wizard 57
  - TCP\_ADJUST\_GIVE\_REST\_TO\_RX macro 438
  - TCP\_ADJUST\_GIVE\_REST\_TO\_TX macro 438
  - TCP\_ADJUST\_PRESERVE\_RX macro 438
  - TCP\_ADJUST\_PRESERVE\_TX macro 439
  - TCP\_AUTO\_TRANSMIT\_TIMEOUT\_VAL macro 474
  - TCP\_CLOSE\_WAIT\_TIMEOUT macro 474
  - TCP\_DELAYED\_ACK\_TIMEOUT macro 474
  - TCP\_FIN\_WAIT\_2\_TIMEOUT macro 475
  - TCP\_HEADER structure 475
  - TCP\_KEEP\_ALIVE\_TIMEOUT macro 476
  - TCP\_MAX\_RETRIES macro 476
  - TCP\_MAX\_SEG\_SIZE\_RX macro 476
  - TCP\_MAX\_SEG\_SIZE\_TX macro 477
  - TCP\_MAX\_SYN\_RETRIES macro 477
  - TCP\_MAX\_UNACKED\_KEEP\_ALIVES macro 477
  - TCP\_OPEN\_IP\_ADDRESS macro 439
  - TCP\_OPEN\_NODE\_INFO macro 439
  - TCP\_OPEN\_RAM\_HOST macro 439
  - TCP\_OPEN\_ROM\_HOST macro 440
  - TCP\_OPEN\_SERVER macro 440
  - TCP\_OPTIMIZE\_FOR\_SIZE macro 477
  - TCP\_OPTIONS structure 478
  - TCP\_OPTIONS\_END\_OF\_LIST macro 478
  - TCP\_OPTIONS\_MAX\_SEG\_SIZE macro 478
  - TCP\_OPTIONS\_NO\_OP macro 478
  - TCP\_SOCKET type 462
  - TCP\_SOCKET\_COUNT macro 479
  - TCP\_START\_TIMEOUT\_VAL macro 479
  - TCP\_STATE enumeration 462
  - TCP\_SYN\_QUEUE structure 479
  - TCP\_SYN\_QUEUE\_MAX\_ENTRIES macro 480
  - TCP\_SYN\_QUEUE\_TIMEOUT macro 480
  - TCP\_WINDOW\_UPDATE\_TIMEOUT\_VAL macro 474
  - TCPAddSSLListener function 378
  - TCPAdjustFIFOSize function 440
  - TCPClose function 441
  - TCPConnect macro 441
  - TCPCDiscard function 442
  - TCPCDisconnect function 442
  - TCPCFind macro 443
  - TCPCFindArray macro 443
  - TCPCFindArrayEx function 443
  - TCPCFindEx function 444
  - TCPCFindROMArray macro 445
  - TCPCFindROMArrayEx function 445
  - TCPCFlush function 446
  - TCPCGet function 446
  - TCPCGetArray function 447
  - TCPCGetRemoteInfo function 447
  - TCPCGetRxFIFOFree function 448
  - TCPCGetRxFIFOFull macro 448
  - TCPCGetTxFIFOFree macro 449
  - TCPCGetTxFIFOFull function 449
  - TCPCInit function 463
  - TCPIP Demo App Features by Hardware Platform 81
  - TCPCIsConnected function 449
  - TCPCIsGetReady function 450
  - TCPCIsPutReady function 450
  - TCPCIsSSL function 379

---

TCPListen macro 451	TFTP_ARP_TIMEOUT_VAL macro 501
TCPOpen function 451	TFTP_BLOCK_SIZE macro 504
TCPPeek function 453	TFTP_BLOCK_SIZE_MSB macro 504
TCPPeekArray function 453	TFTP_CHUNK_DESCRIPTOR structure 498
TCPPerformanceTask function 288	TFTP_CLIENT_PORT macro 504
TCPProcess function 464	TFTP_END_OF_FILE enumeration member 496
TCPPut function 454	TFTP_ERROR enumeration member 496
TCPPutArray function 454	TFTP_ERROR_ACCESS_VIOLATION enumeration member 495
TCPPutROMArray function 455	TFTP_ERROR_DISK_FULL enumeration member 495
TCPPutROMString function 455	TFTP_ERROR_FILE_EXISTS enumeration member 495
TCPPutString function 456	TFTP_ERROR_FILE_NOT_FOUND enumeration member 495
TCPRAMCopy function 456	TFTP_ERROR_INVALID_OPERATION enumeration member 495
TCPRAMCopyROM function 457	TFTP_ERROR_NO_SUCH_USE enumeration member 495
TCPRequestSSLMessage function 383	TFTP_ERROR_NOT_DEFINED enumeration member 495
TCPRXPerformanceTask function 289	TFTP_ERROR_UNKNOWN_TID enumeration member 495
TCPSSLDecryptMAC function 465	TFTP_FILE_MODE enumeration 495
TCPSSLGetPendingTxSize function 383	TFTP_FILE_MODE_READ enumeration member 495
TCPSSLHandleIncoming function 384	TFTP_FILE_MODE_WRITE enumeration member 495
TCPSSLHandshakeComplete function 384	TFTP_GET_TIMEOUT_VAL macro 502
TCPSSLInPlaceMACEncrypt function 385	TFTP_MAX_RETRIES macro 502
TCPSSLIsHandshaking function 378	TFTP_NOT_READY enumeration member 496
TCPSSLPutRecordHeader function 385	TFTP_OK enumeration member 496
TCPStartSSLClient function 379	TFTP_OPCODE enumeration 504
TCPStartSSLClientEx function 465	TFTP_RESULT enumeration 496
TCPStartSSLServer function 386	TFTP_RETRY enumeration member 496
TCPTick function 464	TFTP_SERVER_PORT macro 505
TCPTXPerformanceTask function 290	TFTP_STATE enumeration 505
TCPWasReset function 458	TFTP_TIMEOUT enumeration member 496
Telnet 481	TFTP_UPLOAD_COMPLETE macro 499
Telnet Internal Members 483	TFTP_UPLOAD_CONNECT macro 499
Telnet Public Members 481	TFTP_UPLOAD_CONNECT_TIMEOUT macro 499
Telnet Stack Members 483	TFTP_UPLOAD_GET_DNS macro 499
TELNET_PASSWORD macro 482	TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT macro 500
TELNET_PORT macro 482	TFTP_UPLOAD_RESOLVE_HOST macro 500
TELNET_USERNAME macro 482	TFTP_UPLOAD_SEND_DATA macro 500
TELNETS_PORT macro 482	TFTP_UPLOAD_SEND_FILENAME macro 500
TelnetTask function 483	TFTP_UPLOAD_SERVER_ERROR macro 500
TFTP 485	TFTP_UPLOAD_WAIT_FOR_CLOSURE macro 501
TFTP Internal Members 502	TFTPClose macro 488
TFTP Public Members 486	TFTPCloseFile function 488
TFTP Stack Members 501	
TFTP_ACCESS_ERROR enumeration 495	

---

---

TFTPGet function 489  
 TFTPGetError macro 489  
 TFTPGetUploadStatus function 496  
 TFTPIsFileClosed function 490  
 TFTPIsFileOpened function 490  
 TFTPIsFileOpenReady macro 491  
 TFTPIsGetReady function 491  
 TFTPIsOpened function 492  
 TFTPIsPutReady function 492  
 TFTPOpen function 493  
 TFTPOpenFile function 494  
 TFTPOpenROMFile function 494  
 TFTPPut function 495  
 TFTPUploadFragmentedRAMFileToHost function 497  
 TFTPUploadRAMFileToHost function 498  
 Tick Internal Members 514  
 Tick Module 509  
 Tick Public Members 510  
 Tick Stack Functions 513  
 TICK type 510  
 TICK\_HOUR macro 511  
 TICK\_MINUTE macro 511  
 TICK\_SECOND macro 511  
 TickConvertToMilliseconds function 511  
 TickGet function 512  
 TickGetDiv256 function 512  
 TickGetDiv64K function 513  
 TickInit function 513  
 TICKS\_PER\_SECOND macro 515  
 TickUpdate function 514  
 TransportState variable 310  
 TRAP macro 348  
 TRAP\_COMMUNITY\_MAX\_LEN macro 326  
 TRAP\_INFO structure 318  
 TRAP\_TABLE\_SIZE macro 326  
 trapInfo variable 348  
 tWFDeviceInfoStruct structure 593  
 Tx Power Control Public Members 580  
 TX\_PERFORMANCE\_PORT macro 291

**U**  
 UART-to-TCP Bridge 112

UDP 516  
     Types 536  
 UDP Internal Members 531  
 UDP Public Members 517  
 UDP Sockets 148  
 UDP Stack Members 529  
 UDP\_HEADER structure 534  
 UDP\_OPEN\_IP\_ADDRESS macro 528  
 UDP\_OPEN\_NODE\_INFO macro 528  
 UDP\_OPEN\_RAM\_HOST macro 529  
 UDP\_OPEN\_ROM\_HOST macro 529  
 UDP\_OPEN\_SERVER macro 529  
 UDP\_PORT type 534  
 UDP\_SOCKET type 518  
 UDP\_SOCKET\_INFO structure 534  
 UDP\_STATE enumeration 536  
 UDPClose function 521  
 UDPDiscard function 521  
 UDPFlush function 522  
 UDPGet function 522  
 UDPGetArray function 523  
 UDPInit function 530  
 UDPIsGetReady function 523  
 UDPIsOpened function 528  
 UDPIsPutReady function 524  
 UDPOpen macro 520  
 UDPOpenEx function 519  
 UDPPerformanceTask function 288  
 UDPProcess function 530  
 UDPPut function 524  
 UDPPutArray function 525  
 UDPPutROMArray function 525  
 UDPPutROMString function 526  
 UDPPutString function 526  
 UDPRxCount variable 535  
 UDPSetRxBuffer function 527  
 UDPSetTxBuffer function 527  
 UDPSocketInfo variable 535  
 UDPTask function 531  
 UDPTxCount variable 535  
 uitoa function 221  
 ultoa function 221

---

UnencodeURL function 222  
 UNKNOWN\_SOCKET macro 438  
 uploadChunkDescriptor variable 508  
 uploadChunkDescriptorForRetransmit variable 508  
 Uploading Pre-built MPFS2 Images 58  
 Uploading Web Pages 72  
 URG macro 480  
 Using the Bootloader 117  
 Using the Stack 131

## V

VENDOR\_SPECIFIC\_TRAP\_NOTIFICATION\_TYPE enumeration 316  
 vTickReading variable 515  
 vUploadFilename variable 508  
 vUploadRemoteHost variable 508

## W

Web Page Demos 82  
     Functions 87  
     Variables 90  
 WebVend 120  
 WF\_CAGetBeaconTimeout function 555  
 WF\_CAGetBeaconTimeoutAction function 556  
 WF\_CAGetChannelList function 557  
 WF\_CAGetConnectionProfileList function 557  
 WF\_CAGetDeauthAction function 558  
 WF\_CAGetElements function 558  
 WF\_CAGetEventNotificationAction function 559  
 WF\_CAGetListenInterval function 559  
 WF\_CAGetListRetryCount function 560  
 WF\_CAGetMaxChannelTime function 560  
 WF\_CAGetMinChannelTime function 561  
 WF\_CAGetProbeDelay function 561  
 WF\_CAGetRssi function 562  
 WF\_CAGetScanCount function 562  
 WF\_CAGetScanType function 563  
 WF\_CASetBeaconTimeout function 563  
 WF\_CASetBeaconTimeoutAction function 564  
 WF\_CASetChannelList function 565  
 WF\_CASetConnectionProfileList function 565  
 WF\_CASetDeauthAction function 566  
 WF\_CASetElements function 566  
 WF\_CASetEventNotificationAction function 567  
 WF\_CASetListenInterval function 567  
 WF\_CASetListRetryCount function 568  
 WF\_CASetMaxChannelTime function 569  
 WF\_CASetMinChannelTime function 569  
 WF\_CASetProbeDelay function 570  
 WF\_CASetRssi function 570  
 WF\_CASetScanCount function 571  
 WF\_CASetScanType function 571  
 WF\_CMConnect function 576  
 WF\_CMDisconnect function 577  
 WF\_CMGetConnectionState function 577  
 WF\_CMInfoGetFSMStats function 578  
 WF\_CPCreate function 541  
 WF\_CPDelete function 542  
 WF\_CPGetAdHocBehavior function 542  
 WF\_CPGetBssid function 543  
 WF\_CPGetDefaultWepKeyIndex function 543  
 WF\_CPGetElements function 544  
 WF\_CPGetIds function 544  
 WF\_CPGetNetworkType function 545  
 WF\_CPGetSecurity function 546  
 WF\_CPGetSsid function 547  
 WF\_CPSetAdHocBehavior function 547  
 WF\_CPSetBssid function 548  
 WF\_CPSetDefaultWepKeyIndex function 548  
 WF\_CPSetElements function 549  
 WF\_CPSetNetworkType function 549  
 WF\_CPSetSecurity function 550  
 WF\_CPSetSsid function 551  
 WF\_GetDeviceInfo function 588  
 WF\_GetMacAddress function 588  
 WF\_GetMacStats function 589  
 WF\_GetMultiCastFilter function 589  
 WF\_GetPowerSaveState function 583  
 WF\_GetRegionalDomain function 590  
 WF\_GetRtsThreshold function 591  
 WF\_HibernateEnable function 584  
 WF\_ProcessEvent 595  
 WF\_PsPollDisable function 585  
 WF\_PsPollEnable function 585

WF\_Scan function 579  
WF\_ScanGetResult function 580  
WF\_SetMacAddress function 591  
WF\_SetMultiCastFilter function 592  
WF\_SetRegionalDomain function 592  
WF\_SetRtsThreshold function 593  
WF\_TxPowerGetFactoryMax function 582  
WF\_TxPowerGetMinMax function 581  
WF\_TxPowerSetMinMax function 581  
WFCAElementsStruct structure 572  
WFCPElementsStruct structure 551  
WFMacStatsStruct structure 594  
wGetOffset variable 535  
wICMPSequenceNumber variable 265  
Wi-Fi API 537  
Wi-Fi Connection Algorithm 554  
Wi-Fi Connection Manager 576  
Wi-Fi Connection Profile 540  
WiFi Console 121  
WiFi EZConfig 127  
Wi-Fi Miscellaneous 587  
Wi-Fi Miscellaneous Public Members 587  
Wi-Fi Power Save 582  
Wi-Fi Scan 578  
WiFi Tips and Tricks 599  
Wi-Fi Tx Power Control 580  
wPutOffset variable 536  
wUploadChunkOffset variable 509  
wUploadChunkOffsetForRetransmit variable 509

## Z

Zero Configuration (ZeroConf) 113