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® is a registered trademark of the Wi-Fi Alliance.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Help</td>
<td>Where to get help for the TCP/IP Stack</td>
</tr>
<tr>
<td>Directory Structure</td>
<td>Describes where to find files in the TCP/IP Stack</td>
</tr>
</tbody>
</table>
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](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](http://forum.microchip.com).

Microchip also offers embedded network classes through Regional Training Centers. For more information, visit [http://www.microchip.com/rtc](http://www.microchip.com/rtc).
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 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 that can run the TCP/IP stack.
  - The **Utilities** subdirectory contains PC-based utilities that can help when using the stack. See the **Software** section for more information. The source code for the **Microchip TCPIP Discoverer**, the **MPFS2** 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.
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.

NON-EXCLUSIVE SOFTWARE LICENSE AGREEMENT

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, Security Package 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").

1. Definitions. As used in this Agreement, the following capitalized terms will have the meanings defined below:
   a. "Microchip Products" means Microchip microcontrollers and Microchip digital signal controllers.
   b. "Licensee Products" means Licensee products that use or incorporate Microchip Products.
   c. "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.
   d. "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.
   e. "Third Party" means Licensee’s agents, representatives, consultants, clients, customers, or contract manufacturers.
   f. "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:
   a. use the Software in connection with Licensee Products and/or Third Party Products;
   b. if Source Code is provided, modify the Software; provided that Licensee clearly notifies Third Parties regarding the source of such modifications;
   c. 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") and this Agreement accompanies such distribution;

d. 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");

e. 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;

f. 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. This includes, by way of example but not as a limitation, any standards setting organizations requirements and, particularly with respect to the Security Package Software, local encryption laws and requirements. 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. 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.** Licensee will not: (a) engage in unauthorized use, modification, disclosure or distribution of Software or Documentation, or its derivatives; (b) use all or any portion of the Software, Documentation, or its derivatives except in conjunction with Microchip Products, Licensee Products or Third Party Products; or (c) reverse engineer (by disassembly, decompilation or otherwise) Software or any portion thereof. Licensee may not remove or alter any Microchip copyright or other proprietary rights notice posted in any portion of the Software or Documentation. 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: (x) any claims directly or indirectly arising from or related to the use, modification, disclosure or distribution of the Software, Documentation, or any intellectual property rights related thereto; (y) the use, sale and distribution of Licensee Products or Third Party Products; and (z) breach of this Agreement.

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 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 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). Licensee and Third Party use of such modifications and derivatives is limited to the license rights described in 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.

11. **Limited Liability.** IN NO EVENT WILL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER ANY 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.

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 exclusively in either the U.S. District Court for the District of Arizona, Phoenix Division, or the Superior Court of Arizona located in Maricopa County, Arizona. 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 Microchip. 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 import and 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. 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
Release Notes

******

Microchip TCP/IP Stack Version Log:

******

Remarks


******

v5.42.08 June 2013

******

Changes:

1. Clarify item#4 in v5.42.04 Oct 2012 release. Change is applicable to MRF24WB/G only. (MRF24WB/G) SSL client with RSA 2048bits is enabled by default. Applies only for PIC24/32. Not supported for PIC18.

2. (WiFi G Demo Board) Disable option WF_EASY_CONFIG_DEMO in WF_Config.h to resolve compilation error with IperfAppInit().

Fixes:

1. (MRF24WB) Resolve firmware 0x120C host scan bug : Host scan asserts when it is conducted after WPA-PSK fails due to key mismatch. Refer to WFScan.c and WFMgmtMsg.c.

2. (MRF24WB) Resolve reset problem bug. Modified WF_SetCE_N
and WF_SetRST_N to first set the level, than configure pin as output. Refer to WFDriverPrv.h. This change is already available in WFDriverPrv_24G.h for MRF24WG in MLA v5.42.02 Aug 2012 release.

3. (MRF24W) Resolve WiFi Console demo IPERF issue bug fix: Enlarge generic tcp rx/tx buffer size. Refer to TCPIP MRF24W.h.

4. (MRF24W) Resolve EasyConfig bug fix for retry count when DISABLE_MODULE_FW_CONNECT_MANAGER_IN_INFRASTRU is enabled. Refer to function WFEasyConfigProcess() in WFEasyconfig.c

5. (MRF24W) Resolve handling of max length SSID of AP. Refer to WFEasyConfig.c and CustomHTTPApp.c

******

v5.42.06 Feb 2013

******

Changes:

1. (MRF24WG) Due to memory constraints, all future RF module FW releases will follow this roll-out order (Odd number eg 0x3109, 0x310b) Differences: Supports Wi-Fi Direct (P2P). But no EAP and no multi-client support. (Even number eg 0x3108, 0x310a) Differences: Supports EAP (with MCHP approval) and multi-clients. But no Wi-Fi Direct support.

2. (MRF24WG) Added new project Wifi G demo. Wifi G demo currently only support softAP function,

3. (MRF24WG) In Wifi Console, for XC32-EX116-EAP_MRF24WG, supports security mode WF_SECURITY_WPA2_ENTERPRISE (EAP-PEAP/MSCHAPv2 and EAP-TTLS/MSCHAPv2) . Requires RF module FW version 0x3108 and even number releases. This requires special approval submitted to marketing.

4. (MRF24WG) In EZConfig, added mobile application demo to Explorer16+PIC32. Refer to MobileTCPServer.c Supported devices
Current functions supported are (i) Temperature reading (ii) Potentiometer reading and (iii) LEDs ON/OFF toggling App name is "OpenWifiFog"

5. (MRF24WG) In WiFi Console, added cloud demo to Explorer16+PIC32. Refer to CloudTCPClient.c Current functions supported are (i) Temperature reading (ii) Potentiometer reading and (iii) LEDs ON/OFF toggling Reference web server is http://www.openwificloud.com

6. (MRF24WG) Supports multi-client DHCP server (max 4 clients). SoftAP supports up to max 4 client Requires RF module FW version 0x3108 and even number releases.

7. (MRF24WG) SoftAP supports active scan. Requires RF module FW version 0x3108 and future releases.

8. (MRF24WG) RF module FW version 0x3107 and future releases will no longer support programming of regional domain via function WF_SetRegionalDomain() due to changes in FCC requirements, which does not allow programming of the regional domain.

9. (MRF24WB/G) Only WEP key index 0 is valid for security mode WEP.

10. (MRF24WG) Created new event
WF_EVENT_SOFT_AP_EVENT_SUBTYPE / WF_EVENT_SOFT_AP_EVENT, for softAP mode, to indicate client's status (connected or disconnected to softAP). Requires RF module FW version 0x3108 and later releases. If prior MLA releases are used with this new RF module FW version release, you need to port over this event handling in WFEVentHandler.c and WF_ProcessEvent() (WiFi EZConfig).

11. Added full versions of RSA.c, ARCFOUR.c, and AES_PIC32MX.a to the TCP/IP Stack Distribution to support SSL and SNMPv3. Previously these files were distributed in a separate cryptographic code distribution. These files are subject to the U.S. Export Administration Regulations and other U.S. law, and may not be exported or re-exported to certain countries or to persons or entities prohibited from receiving U.S. exports (including Denied
Parties, entities on the Bureau of Export Administration Entity List, and Specially Designated Nationals).

Fixes:

1. (MRF24WG) In EZConfig, SoftAP now supports Zeroconf & mDNS. Resolves wrong port issue. Search for keyword SOFTAP ZEROCONF SUPPORT.

******

v5.42.04 Oct 2012

******

Changes:

1. (MRF24WB/G) Patch update features are added into MRF24WB0MA/B and MRF24WG0MA/MB. MRF24WB0MA/B (no RF module FW version update is necessary) Serial Port (Xmodem) MRF24WG0MA/B (Requires RF module FW version 3107 or later.) Serial Port (Xmodem), Web Client, Web Server

2. (MRF24WB/G) FTP client is added into WiFi console demo app.

3. (MRF24WB/G) Flexible scratch memory is used to allocate TCB.

4. SSL client with RSA 2048bits is enabled by default. Applies only for PIC24/32. Not supported for PIC18.

5. 16-bit mode feature is available on PIC32.

6. (MRF24WG) Enhance SoftAP with EZConfig features with pre-scan and redirection features. Change SoftAP address from 192.168.1.1 to 192.168.1.3 Added WEP security, which requires RF module FW version 3107 or later. Refer to WF_Config.h for more information.

7. Added support for PIC32MX6XX external PHY’s: RTL 8201FL. Only RMII configuration is supported with RTLPHY8201FL PHY.
This PHY driver works in Default Ethernet IO mode with PIC32MX675F512H. If Alternate Ethernet IO mode is used for other PIC devices, then configuration bit need to be changed w.r.t Alternate Ethernet IO. To get the status of MAC link with this PHY, PHY Link Status need to be read twice. EthPhyGetLinkStatus(int refresh) API is used to read the PHY link status and for this PHY to read link status twice, the parameter refresh should be 1.

Fixes:

1. (MRF24WG) SoftAP under EZConfig has pre-scan and redirection features. Able to be redirected to infrastructure mode AP.
2. If SNMP_TRAP_DISABLED macro is enabled from TCPIP XXX.h file, there will be no TRAP table information for SNMP manager. That is SNMP agent won’t send information related to trap. By default SNMP_TRAP_DISABLED macro is disabled.
3. Fix typo error in v5.42.02 (MRF24WG) SoftAP default address is changed from 169.254.1.1 to 192.168.1.1
4. Fixed SNMP tabular issue if SNMP instances starts other than 0.
5. Fixed ARP initialization bug that resulted in TCP packets sent to ETH MAC broadcast address.

******

v5.42.02 Aug 2012

******

Fixes:

1. (MRF24WB/G) TCPIPConfig.exe is modified to support changes in macro definitions (CFG_WF_INFRASTRUCTURE, CFG_WF_ADHOC) used in WF_Config.h.
2. (MRF24WB/G) Ad-hoc mode was left out in the TCPIP-Demo app. This mode is reinstated back into TCPIP-Demo app. 2 files are changed. .MicrochipIncludeTCPIP StackWFDebugStrings.h
3. (MRF24WB/G) SoftAP default address is changed from 192.168.1.1 to 169.254.1.1

4. (MRF24WG) Apply SSL fix. Root cause is traced to be due to scratch pad memory to memory copy functions in WFMac_24G.c

5. (MRF24WG) Reset fix. a. In file WFDriverCom_24G.c, MRF24WG reset sequence is modified.
   - WFHardwareInit() does hard reset (putting I/O lines in correct state)
   - ChipReset() does soft reset

   b. In file WFDriverPrv_24G.h, macros WF_SetCE_N() and WF_SetRST_N() modified to first set the output level, then set the I/O direction

6. (MRF24WG) Applies to using PICDEMNet2 board and PIC18. Related to SPI interface settings. a. In file WF_Spi.c, ConfigureSpiMRF24W() modified, for PIC18 only, to change the SPI clock idle state to low.

7. (MRF24WB/G) When host attempts to read scan results while the module FW is in reconnecting state, the module FW returns WF_ERROR_NO_STORED_BSS_DESCRIPTOR error. Reading scan results is only allowed in A) connected state; B) idle state. Fix is in WaitForMgmtResponseAndReadData().

```c
void WaitForMgmtResponseAndReadData(UINT8 expectedSubtype, UINT8 numDataBytes, UINT8 startIndex, UINT8 *p_data) {
    ........................................
    /* check header result and subtype fields */
    WF_ASSERT(hdr.result == WF_SUCCESS || hdr.result == WF_ERROR_NO_STORED_BSS_DESCRIPTOR);
    WF_ASSERT(hdr.subtype == expectedSubtype);
    ........................................
}
```

*****

v5.42 Jul 2012
Changes:

1. All MPLAB 8 and MPLAB X projects have been modified to use the XC16 and XC32 compilers.
2. The colon character "::" has been added as a valid terminator for an IP address in the StringTo IPAddress function.
3. (MRF24WB/G) PIC18 is only supported with "TCPIP Demo App". MRF24WB0MA/MB works on PICDEM PIC18 Explorer. MRF24WG0MA/MB works on PICDemNet2 PIC 18 but has issues with PICDEM PIC18 Explorer.
4. (MRF24WB/G) Enhanced debug messages providing more details such as authentication or association failues, etc.
5. (MRF24WB/G) Added option to disable module FW connection manager by adding "#define DISABLE_MODULE_FW_CONNECT_MANAGER_IN_INFRASTRUCTURE".
6. (MRF24WB/G) Added option to derive real key from pass-phrase in host side by adding "#define DERIVE_KEY_FROM_PASSPHRASE_IN_HOST".
7. (MRF24WB/G) Enabled WEP security for EasyConfig Ad-hoc connection
8. (MRF24WB/G) Added support for MRF24WG0MA/MB, a superset of MRF24WB0MA/MB (MRF24WB). Required to add to project definition files (#define MRF24WG). Ranges are different from MRF24WB0MA/MB. i. Valid RSSI Range : 43 ~ 128 (max) (WF_ScanGetResult()) ii. Max transmit power range : 0 ~ 18 dbm (WF_TxPowerSetMax())

MRF24WG0MA/MB enhancements are above and beyond on those features listed above. a. Added WPS(1.0) security method (#define WF_SECURITY_WPS_PUSH_BUTTON & WF_SECURITY_WPS_PIN ) (TCPIP demo + console demo). b. Added Wi-Fi Direct (P2P) function in the GC (group client) role (#define MY_DEFAULT_NETWORK_TYPE CFG_WF_P2P) (TCPIP demo + console demo) c. Added simplified
basic SoftAP functionality (WF_SOFT_AP). Current features are 1 client STA, open security and no routing. (Easy config demo) d. Selection of support for 16 (max) software multicast filters (ENABLE_SOFTWARE_MULTICAST_FILTER ) or 2 hardware multicast filters. RTS or ES release requires software patch.

Fixes:

1. The BigInt_helper.S file will now correctly include the processor include files for the PIC24E and dsPIC33E architectures.
2. Fixed bug in TCP state machine that corrupted the sequence number when client socket is closed.
3. Fixed bug in BerkeleiAPI.c::sendto() that checked for the wrong UDP socket to be opened.
4. (All parts including RTS or ES release) MRF24WG0MA/MB requires PLL work-around initialization code. Please refer to July/2012 release.
5. (All parts including RTS or ES release) MRF24WG0MA/MB needs to update stack from July/2012 release to cater for SSL.
6. (MRF24WB/G) Fixed stack issues encountered in Stack version v2012-04-03 and older. Code fix is needed in WF_CPSetElements() API. Modification is as below.

```c
void WF_CPSetElements(UINT8 Cpld, tWFCPElements *p_elements) {
    UINT8 elements[sizeof(tWFCPElements) + 2];
    WF_ASSERT(p_elements->ssidLength <= WF_MAX_SSID_LENGTH);
    memset(elements, 0, sizeof(elements));
    memcpy(elements, p_elements, sizeof(*p_elements));
    LowLevel_CPSetElement(Cpld, /* CP ID */ WF_CP_ELEMENT_ALL, /* Element ID */ (UINT8 *)elements, /* pointer to element data */ sizeof(elements)); /* number of element data bytes */
}
```

Code fix is also needed in WaitForMgmtResponse() API (WFMgmtMsg.c). Refer to description of this fix is in the Supplementary TCPIIP Help folder.
Known Problems:

1. **Telnet** server module does not implement a lot of **Telnet** functions. As a result, it will likely not display correctly or work at all with some **Telnet** 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** 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.

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.

7. (MRF24WG0MA/MB) RTS or ES release requires software patch for multicast filters.

8. (MRF24WG0MA/MB) Under softap option with easy config demo, unable to **connect** to infrastructure mode AP.

9. Certain APs have power save issue with reconnect. Please refer to
AP compatibility list.

******
v5.41.02 Apr 2012
******

Fixes:

1. Added the Bigint helper libraries for C30 into the stack source folder to prevent linker errors in 16-bit projects that include those libraries.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet 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 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.

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.

******

v5.41 Feb 2012

******

Changes:

1. Added socket number validation to TCP functions.
2. Removed the WiFi projects from the Energy Monitoring and WebVend demos.
3. Added the "-mperipheral-libs" linker option to C32 projects.
4. Moved AES library and header files used by SNMPv3 from the "Crypto" source and include folders to the to the "TCPIP Stack" source and include folders.
5. Changed SNMP MIB trap number from 7 to 0.
6. Changed SNMP MIB SYOBJECTID to 43.6.1.4.1.17095.1 (added .1).
7. Updated MPFS2 Utility to make path settings persistent.
8. Changed default channel list from \{1,6,11\} to \{1,2,3,4,5,6,7,8,9,10,11\} for FCC domain
9. Added support for WEP with Shared Key
10. Added new command "iwconfig scan" & "iwconfig scanresults"
11. Added gRFModuleVer1209orLater" flag to identify version 1209 and later specific API's.
12. Added the following 1209 and later specific API: a)
**WF_CPSetWepKeyType**()  b) **WF_CMGetConnectContext**()  c) **WFEnableBroadcastProbeResponse**()  d) **WFEnableAggressivePowerSave**()  e) **WF_CPSetssidType**()  f) **WFEnableDeferredPowerSave**()  g) **WF_FixTxRateWithMaxPower**()

**Fixes:**

1. Updated MPFS2 Utility path strings for Mac/Linux compatibility.
2. Iperf - Resolved application crash due to network disconnection.
3. Host Scan - More robust and prevent system hang up. #define **WF_HOST_SCAN**.
4. Power Save - Improvements with Aggressive PS mode, and better handling of PS in host code.
5. Host **connect** & disconnect - Prevent system hang.
6. DHCP Refresh - Better handling of DHCP session in PS mode, and issue DHCP renewal any STA reconnect with AP.

**Known Problems:**

1. **Telnet** server module does not implement a lot of **Telnet** functions. As a result, it will likely not display correctly or work at all with some **Telnet** 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** 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 its 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.

******

v5.36.4 Oct 2011

******

Changes:

1. **SNMPNotify()** is updated to support ASCII string varible type for both TRAPv1 and TRAPv2. ASCII string address pointer is assigned to argument val(**SNMP_VAL**) of **SNMPNotify()**.

2. The SSL module has been updated to support 1024-bit RSA key lengths for server and client on all architectures. PIC32 microcontrollers now support client/server RSA key lengths of 2048 bits. NOTE: To support these changes, you must manually modify your copy of RSA.c. A description of the required changes ("Required SSL Changes.pdf") can be found in your Microchip Applications Libraries installation directory in the "MicrochipHelpSupplementary TCPIP Help" subdirectory.

Fixes:
1. SNMP local variable community length increased with plus one. SNMP warnings has been removed for the compiler version C32 2.01 for zero optimisation.

2. Updated MPFS2.jar and mib2bib.jar to support Java version 1.7.

3. Fixed MPFS2.jar offset issue for fileRcrd.bin and dynRcrd.bin file and it was due to the file which has zero dynamic variable. Fixed Crimson editor problem with webPage2 folder where user couldn't save files using Crimson Editor if the WebPages2 folder that contained those files was selected in the MPFS2 utility.

4. MPFS2.jar file was getting hanged for the zero file size access. Now Zero file size also is the part of the respective generated files.

Known Problems:

1. **Telnet** server module does not implement a lot of **Telnet** functions. As a result, it will likely not display correctly or work at all with some **Telnet** 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** 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.

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.

******

v5.36.2 July 2011

******

Changes:


2. Modified the Energy Monitoring demo to remove Google PowerMeter functionality. The demo will still display measured power data on its internal web page.

3. Updated the TCP/IP Stack Performance table to use the testing methodology from previous releases. More information is available in the TCP/IP Stack Help file.

4. `gSnmpNonMibRecInfo[]` has been moved from snmp.c file to CustomSNMPApp.c file and `SNMP_MAX_NON_REC_ID_OID` macro has been moved from snmp.h file to CustomSNMPApp.c file. `gSnmpNonMibRecInfo[]` is used to list the static variables parent OID strings which are not part of mib.h file. This structure is used to restrict the access to the SNMPv3 objects from SNMPv2c and SNMPv1 version requests. Macro `STACK_USE_SMIV2` is used to support `gSnmpNonMibRecInfo[]` with MODULE-IDENTITY
number. For V5.31 STACK_USE_SMIV2 need to commented.

5. Removed the SPI2CON register freeze-on-halt bit macro from the SPIFlash, RAM, and EEPROM driver files to provide compatibility with C32 v2.00.

Fixes:

1. Removed the MPFSImg2 files from the MPLAB X C18/C30 projects so that the projects will compile. Disabled MPFSImg2.c for PIC32 Explorer 16 projects.
2. Added a heap and minimum stack size for the PIC32 Ethernet Starter Kit MPLAB X project.
3. The TCP/IP Stack Help File's performance table has been updated using the same test procedure used in previous releases.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet 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 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.
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.

******

v5.36 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 function provides this additional functionality. Please consult the TCP/IP Stack Help File's "Stack API > UDP" topic for more information.

3. The UDPOpen macro has been added to conform to the legacy UDPOpen interface.

4. The Announce, BerkeleyAPI, DHCP client/server, DNS client/server, NBNS, Reboot, 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. Functionality 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 SMIv2 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 SMIv2 standard. This mib includes MODULE-IDENTITY which will provide MICROCHIP and MIB information. snmp.mib also includes MODULE-IDENTITY(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 IPPROTO_IP 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."


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 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 module to ensure the MAC layer is linked before attempting to transmit an Announce 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 server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet 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 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.

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
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() and LFSRSeedRand() 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() function returns a 16-bit integer, regardless of which PIC you are using.

6. Added support for various SST/Microchip brand SST25xxxxx SPI Flash chips to SPIFlash.c driver. Previously only devices supporting the Auto Address 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 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 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_ETHETERNET.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 | Data Contents
---------|--------------
The additional checkums 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:

1. 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 \texttt{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 \texttt{UDPSocketInfo[SocketHandle]} global structure before calling \texttt{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 \texttt{UDPOpen()} API, not the literal string "SocketHandle".

2. Fixed ADC state save/restore bug in \texttt{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.

3. 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).

4. Fixed an ENCX24J600.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.

5. 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()` 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** (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 \( y = x^e \mod n \) 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** server module does not implement a lot of **Telnet** functions. As a result, it will likely not display correctly or work at all with some **Telnet** 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** 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(), TFTPUploadFragmentedRAMFileToHost() and TFTPGetUploadStatus() 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 (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 (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" is used to indicate the start and end of SNMPv2 trap varbinds. If gSetTrapSendFlag is FALSE, then very next variable varbind for the SNMPv2 TRAP, is the last or only one variable varbind. If gSetTrapSendFlag 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 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, HTTPS_PORT, HTTP_MAX_DATA_LEN, and HTTP_MIN_CALLBACK_FREE 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() 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() 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 TCPIP 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(), TickGetDiv256() and TickGetDiv64K() 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 server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet 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** 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 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 disable. The exact IP address chosen will be pseudo-random, but as required by the protocol, it will perform gratuatous 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 option to the TCP.c file (default 200ms). This timeout controls the time after calling TCPGet() or TCPGetArray() before the stack will transmit a RX window update to the remote node. Historically, the TCP_AUTO_TRANSMIT_TIMEOUT_VAL 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 equal to or shorter than TCP_AUTO_TRANSMIT_TIMEOUT_VAL 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 and TCP_MAX_SEG_SIZE_RX 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 RSASSetData() and RSASSetE() 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.

5. Fixed `TickGet()`, `TickGetDiv256()` and `TickGetDiv64K()` 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()` on a `SOCK_STREAM` 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()` 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()` 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()` 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()` in CustomHTTPApp.c of the TCPIP WiFi Demo App and TCPIP Iperf 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 to. The define `USE_GRATUITOUS_ARP` (in TCPIPConfig.h) turns this feature on or off.

Known Problems:

1. **Telnet** server module does not implement a lot of **Telnet** functions. As a result, it will likely not display correctly or work at all with some **Telnet** 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** 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 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 server. If STACK_USE_SSL_SERVER is defined, the Telnet server will now listen on port 992 for SSL secured connections. If you do not have a telnets 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() 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).

- `strnchr()` in Helpers.c. Finds the first occurrence of a character within a string but limited to a maximum length before giving up.
- `TCPPeek()` and `TCPPeekArray()` in TCP.c. Reads from a TCP socket's RX FIFO buffer without removing the data from the stream.
- `TCPClose()` 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()` API except for the handling of server sockets. `TCPDisconnect()` returns server sockets to the listning state and maintains the socket handle. `TCPClose()` 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 discover 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 fourth 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(), DHCPIsEnabled(), DHCPStateChanged(), DHCPIsBound(), 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 "TCP/IP 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 TCP/IP 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 TCP/IP WiFi Demo App projects have been optimized for better performance.

Fixes:

1. Fixed a TCPDisconnect() 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() 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 "TCP/IP Demo App-C32 EXPLORER_16 32MX360F512L ENC624J600 PSP 9.hex" precompiled application, among others, if you connected to the TCPIPPerformanceTest.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() 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() 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 server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet 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 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 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 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 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()` and `ICMPSendPingToHostROM()` 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 pinging 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 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 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.


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.


Fixes:

1. Fixed a denial of service vulnerability in the NBNSGetName() 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() 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() 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() 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() 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() 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() 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 server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet 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 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 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` API was called twice with different socket handles without an intermediate call to any other TCP API that takes a `TCP_SOCKET` input. Thanks go to Bob Topper for identifying this problem and suggesting a solution.

3. Fixed the `UDPIsGetReady()` 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 `UDPIsGetReady()` 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()` and `UDPGetArray()` 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** server module does not implement a lot of **Telnet** functions. As a result, it will likely not display correctly or work at all with some **Telnet** 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** 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.

******

v4.51 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()** 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 to a remote server before the Ethernet cable was attached. Once the Ethernet cable was attached, the socket would attempt to resolve and connect 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 client-mode TCP sockets to certain rare servers/services. Thanks go to Jean LE TUTOUR for finding one of these problem servers.

3. **MPFSOpen()** and **MPFSOpenROM()** 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 formus 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()** 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 prinz3nroll3 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** server module does not implement a lot of **Telnet** functions. As a result, it will likely not display correctly or work at all with some **Telnet** 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 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.
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 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. The UDP client demo is similarly identical to the SNTP.c

*******

v4.50 02 June 2008

*******
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 TCPIIP.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 (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). Server oriented sockets will return to the listening state. Client oriented sockets will close, but the TCP_SOCKET handle will continue to remain valid until the application calls TCPDisconnect(). Applications can check if the socket became disconnected and reset by calling TCPWasReset() or TCPIsConnected(). 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 (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 to the next client. This improves connect 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 (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 IndexOutOfBoundsException 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 SolutionsMicrochipTCPIP StackUtilitiesSourceMPFS21 folder. This project is designed to compile with Microsoft Visual C# 2008 Express
Edition. 14. Added support for SST25VFxxxB 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, HTTPReadPostValue, and HTTPReadPostPair functions to facilitate easier processing of data arriving via POST. 16. Split HTTPAuthenticate API into separate functions: HTTPNeedsAuth and HTTPCheckAuth. 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(val) 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 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() 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(*) 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 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** 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()** function from not working if it was called before having called **UDPGet()** or **UDPGetArray()** at least once. 19.Fixed an offset error of +2 milliseconds being returned from **TickConvertToMilliseconds()**. 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** server module does not implement a lot of **Telnet** functions. As a result, it will likely not display correctly or work at all with some **Telnet** 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** 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 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.

******

v4.18 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 (defined at the top of TCP.c) elapses or immediately when the application calls the TCPDisconnect() 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` seconds. `TCP_CLOSE_WAIT_TIMEOUT` defaults to 200ms in this stack version.

7. Pushed the SNTP requery on failure timeout up some. It was ~14 seconds and is now ~20 seconds.

8. Added `TFTPOpenROMFile()` API to complement `TFTPOpenFile()` 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` 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` value defined at the top of SMTP.c. `SMTP_PORT` 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()` 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()` 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 0xFFFFF000 if you provided NULL for the destination address pointer.
2. Fixed an MPFS2.c MPFSGet() 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 option was being sent, which is illegal. These changes may fix compatibility problems with certain DHCP servers.

6. Fixed TFTP Client's TFTPCloseFile() function from sending data using a wrong UDP socket if StackTsk() was called after TFTPIsFileOpened() 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 server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet 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 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.

*******

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` 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() API function. This replaces TCPListen() and TCPConnect(). TCPOpen() 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() and TCPConnect() APIs. TCPOpen() 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 \texttt{TCP\_KEEP\_ALIVE\_TIMEOUT} 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 \texttt{SNTPGetUTCSeconds()} 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. \texttt{Telnet} server module does not implement a lot of \texttt{Telnet} functions. As a result, it will likely not display correctly or work at all with some \texttt{Telnet} 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 \texttt{connect} 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.

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_HTTPgetConnections.

*****
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() 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(), 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
\texttt{TCPIsConnected()} API.

4. Added a counter to the UDPPerformanceTest module and made it supress 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 \texttt{MPFSGetFilename()} 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 \texttt{SNMP_COMMUNITY_MAX_LEN}) 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
incorrect web page when being bombarded with new connection requests.

9. Fixed a `TickGet+'() 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, reconnects 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 if the local PIC called `TCPDisconnect()' and the remote node didn't send us a correct FIN response. `Sockets' could previously get lost in the TCP_FIN_WAIT_2 state and wouldn't recover unless the application called `TCPDisconnect()' a second time with the same socket handle.

Known Problems:

1. `Telnet' server module does not implement a lot of `Telnet' functions. As a result, it will likely not display correctly or work at all with some `Telnet' 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' 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.

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 example of downloading a web page from www.google.com is extremely slow. The default TCP socket has too little RX space to accept 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() API.

8. Added support functions Base64Encode() and Base64Decode() in Helpers.c. Base 64 is required for the new HTTP2 module, but of general use to many applications.

9. Added SMTP Authentication 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(). 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() 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() 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(), TickGetDiv64K(), and TickConvertToMilliseconds(). 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**. 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** 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 address pointer was set.

8. Fixed the DHCP Server module from incorrectly parsing received packets which had a DHCP_PARAM_REQUEST_IPADDRESS 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() was called and a previous application did not call UDPPDiscard() on an RX packet. Now, StackTsk() calls UDPPDiscard() 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() 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() 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
**GenericTCP**Server 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()** tracking problem that would result in extra bytes being appended to the end of a packet if the **UDPS**et**Tx**Buffer() 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** server module does not implement a lot of **Telnet** functions. As a result, it will likely not display correctly or work at all with some **Telnet** 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** 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.

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.
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 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() 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() behavior. Now, if space in the Ethernet TX buffer runs out, the packet will not automatically be transmitted. You must call UDPFlush() to cause the packet to be transmitted.

6. Added UDPGetArray(), UDPPutArray(), UDPPutROMArray(), UDPPutString() and UDPPutROMString() user API functions. These functions perform substantially better than calling UDPPut()
successively and allow greater application programming flexibility.

7. Changed `TCPPutString()` and `TCPPutROMString()` 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()` function. Applications which use successive calls to `UDPput()` 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 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()` function. Applications which use successive calls to `TCPPut()` 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()` 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() 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 a NULL pointer. If NULL, the retrieved data will simply be discarded. This also changes the behavior of UDPGetArray() and TCPGetArray() 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 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() 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 (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() 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(), TCPFindArray(), and TCPFindROMArray() 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 rn). It supports case insensitive text searching or binary searching, as well as an offset to start searching at.
23. Added **TCPGetRxFIFOFree()** 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()`, `ICMPSendPing()`, `ICMPGetReply()`, and `ICMPEndUsage()`. The functions should be called in this order. See the `PingDemo()` 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()` 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 readded to the TCPIP Demo App-C18 project using relative paths instead of absolute paths.

3. **UDPOpen()** 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()** function in Helpers.c is now tested and working.

6. Fixed a TCP Window Update problem when **TCPGetArray()** 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 server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet 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 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.

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 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()` API.
5. Added `TCPPutROMArray()` API.
6. Added `TCPPutString()` API.
7. Added `TCPPutROMString()` API.
8. Added `TCPGetArray()` API.
9. Changed `TCPIsPutReady()` API. Instead of returning a BOOL, it now returns a WORD. The WORD is a count of the number of bytes that `TCPPut()`, `TCPPutArray()`, etc. can immediately place in the output buffer. MAKE SURE THAT YOUR CODE DOES NOT COMPARE THE RETURN RESULT OF `TCPIsPutReady()` DIRECTLY TO TRUE. For example, "if(TCPIsPutReady(MySocket) == TRUE){...}" must be converted over to: "if(TCPIsPutReady(MySocket)){...}"

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

11. Changed `TCPDiscard()` 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()` and `DNSEndUsage()`. 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()** 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(). Rewrote CalcIPBufferChecksum() 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 **ARPProcess()** 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()** would return TRUE
even if no data was left in the received packet. Previously you had to call **TCPGet()** one last time and have it fail before **TCP_IsGetReady()** 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. DeclaredTickCount in Tick.c/Tick.h as volatile and implemented an interrupt safe reading procedure in **TickGet()**. 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 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).
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()` with the server name, ex: `DNSResolve("www.microchip.com")`, and then periodically call `DNSIsResolved()` until it returns TRUE, ex: `DNSIsResolved(&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[]` should be downloaded from `ServerName[]`
and written to the UART. For the default values of ServerName[] and RemoteURL[], 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 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() or UDPClose() 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 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 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 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() 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.
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 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 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 "Compatable" 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
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 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 ERXRDPT 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 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 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()
   - 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** 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 `UDPProcess()` in UDP.c file.
5. Renamed CalcTCPChecksum() in tcp.c file to CalcIPBufferChecksum().
6. Moved CalcIPBufferChecksum() to helpers.c to reuse it for UDP checksum calculation.
7. Modified `UDPProcess()` in UDP.c and `TCPProcess()` 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, HandleTCPSeg() would acknowledge, but throw valid received TCP packets away, resulting in loss of application data. An invalid comparison in HandleTCPSeg() 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` 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 first DHCP offer instead of second response.

2. Corrected DHCP logic to check for chaddr in DHCP offer and accept 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 response from one server instead of verifying who was intended node.

3. Fixed UDPClose() in UDP.c to use INVALID_UDP_PORT instead of INVALID_UDP_SOCKET 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() was modified to overcome compiler rule where only 8-bit value was used to evaluate non-zero condition.
2. ARPResolve() in ARPTsk was fixed to clear Cache.IPAddr value.

******

v2.20.01:

******

Bug fix:

1. Fixed SMSC91C111.c where MACInit() would hand if ethernet link is not detected.
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() where it would return FALSE on last good byte. Fixed UDPProcess() 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 begining 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()**, **MPFSTell()**.
7. MPFSImg.* - Rebuilt to reflect v2.20, footprint changes etc.
9. UDP.h - Added UDPSetTx and UDPSetRx macros. Moved **UDP_SOCKET_INFO** 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 websrvr.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.

   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 dependent 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() to support IP Gleaning
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", "xeeprom.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", "xeeprom.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 remotely programmed.

4. Modified SendFile() to make use of sequential EEPROM read.
None

8. ip.c 1. Added one more parameter 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 xeeprom.h.
None

2. Added MPFSSFormat(), 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 for MPFSPut operations.
   Apps using different EEPROM page size needs to be modified.

11. mpnicee.pjt 1. Removed "ee256.c"
   None
   2. Added "xcld.c", "xeeprom.c" files
   None

12. stacktsk.c 1. Replaced ee256.h with xeeprom.h
    None
    2. Added IP Gleaning and DHCP support.
    None

13. stacktsk.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
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.

None
2. Fixed TCPIsConnected()
None
3. Fixed TCPDisconnect()
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
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() on Timer0 interrupt
N/A

3. Added code to save/restore board configuration
N/A

N/A

5. Added call to FTP modules
If needed, add this.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack Performance</td>
<td>Latest stack performance measurements using TCPPerformanceTest.c and UDPPerformanceTest.c test modules.</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>Details stack program and data memory requirements.</td>
</tr>
</tbody>
</table>
Peripheral Usage

Describes the peripherals required by the TCP/IP stack.
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.

Stack and Compiler Versions

TCP/IP Stack v5.36
MPLAB C Compiler for PIC18 MCUs (C18) v3.38
MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs (C30) v3.25
MPLAB C Compiler for PIC32 MCUs (C32) v1.12

Wired test setup: Connect the board under test directly to a computer using a crossover cable.
Wireless test setup: Connect the PC to a router with a Category 5 cable. Connect

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MIPS</th>
<th>Network Controller</th>
<th>Interface</th>
<th>LAN Transmit Throughput (Kbytes/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCP with 200 byte TX FIFO</td>
</tr>
<tr>
<td>PIC18F97J60</td>
<td>10.4</td>
<td>Internal 10BaseT</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>PIC18F8722</td>
<td>10</td>
<td>ENC28J60</td>
<td>SPI, 10 MHz</td>
<td>21</td>
</tr>
<tr>
<td>PIC24FJ128GA010</td>
<td>16</td>
<td>ENC28J60</td>
<td>SPI, 8 MHz</td>
<td>56</td>
</tr>
<tr>
<td>dsPIC33FJ256GP71</td>
<td>10(2)</td>
<td>ENC28J60</td>
<td>SPI, 8 MHz</td>
<td>75</td>
</tr>
<tr>
<td>PIC32MX360F512L</td>
<td>80</td>
<td>ENC28J60</td>
<td>SPI, 20 MHz</td>
<td>110</td>
</tr>
<tr>
<td>PIC32MX795F512L</td>
<td>80</td>
<td>ENC28J60</td>
<td>SPI, 20 MHz</td>
<td>115</td>
</tr>
<tr>
<td>PIC18F8722</td>
<td>10</td>
<td>ENC624J600(3)</td>
<td>SPI, 10 MHz</td>
<td>28</td>
</tr>
<tr>
<td>PIC24FJ128GA010</td>
<td>16</td>
<td>ENC624J600(3)</td>
<td>SPI, 8 MHz</td>
<td>57</td>
</tr>
<tr>
<td>dsPIC33FJ256GP71</td>
<td>10(2)</td>
<td>ENC624J600(3)</td>
<td>SPI, 8 MHz</td>
<td>94</td>
</tr>
<tr>
<td>PIC32MX360F512L</td>
<td>80</td>
<td>ENC624J600(3)</td>
<td>SPI, 13.33 MHz</td>
<td>142</td>
</tr>
<tr>
<td>PIC32MX795F512L</td>
<td>80</td>
<td>ENC624J600(3)</td>
<td>SPI, 13.33 MHz</td>
<td>154</td>
</tr>
<tr>
<td>PIC24FJ128GA010</td>
<td>16</td>
<td>ENC624J600(3)</td>
<td>PSP Mode 5, PMP</td>
<td>101</td>
</tr>
<tr>
<td>dsPIC33FJ256GP71</td>
<td>10(2)</td>
<td>ENC624J600(3)</td>
<td>PSP Mode 5, Bitbang</td>
<td>102</td>
</tr>
<tr>
<td>Device</td>
<td>SPI Speed</td>
<td>Throughput (KB/s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC32MX360F512L</td>
<td>80</td>
<td>EN624J600 (3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC32MX795F512L</td>
<td>80</td>
<td>Internal 100BaseTX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC18F97J60</td>
<td>10.4</td>
<td>MRF24WB0M SPI, 10.4 MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC24FJ256GB110</td>
<td>16</td>
<td>MRF24WB0M SPI, 8 MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dsPIC33FJ256GP710</td>
<td>10(2)</td>
<td>MRF24WB0M SPI, 8 MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC32MX360F512L</td>
<td>80</td>
<td>MRF24WB0M SPI, 20 MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC32MX795F512L</td>
<td>80</td>
<td>MRF24WB0M SPI, 20 MHz</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Internet throughput with acknowledged protocols, such as TCP, are capped by the formula (Buffer Size)/(Ping Time). Assuming round trip acknowledgement latency cannot be controlled, bigger buffers must be used to increase throughput. Internet performance is derived from LAN throughput and is not measured.

2. Stack performance with a dsPIC33F is identical to a PIC24H device at the same processor clock. No DSP features are used by the TCP/IP stack.

3. ENC624J600 and PIC32MX7XX/6XX Internal 100BaseTX Ethernet measurements are done in 100Mbps, full duplex mode.
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, 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.

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.

<table>
<thead>
<tr>
<th>C18</th>
<th>Program Memory (bytes)</th>
<th>Global Data (byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>No Optimizations</td>
<td>All Optimizations</td>
</tr>
<tr>
<td>Required Stack Code - PIC18F97J60 family controller</td>
<td>9118</td>
<td>7568</td>
</tr>
<tr>
<td>Required Stack Code - MRF24WB0M Wi-Fi controller</td>
<td>25482</td>
<td>19498</td>
</tr>
<tr>
<td>Announce(1)</td>
<td>+3395</td>
<td>+2775</td>
</tr>
<tr>
<td>AutoIP</td>
<td>+2860</td>
<td>+2090</td>
</tr>
<tr>
<td>DHCP Client(1)</td>
<td>+6244</td>
<td>+4684</td>
</tr>
<tr>
<td>DHCP Server(1)</td>
<td>+5012</td>
<td>+3490</td>
</tr>
<tr>
<td>DNS Client(1)</td>
<td>+5955</td>
<td>+4333</td>
</tr>
<tr>
<td>Dynamic DNS Client(1,2,3)</td>
<td>+31519</td>
<td>+22039</td>
</tr>
<tr>
<td>Module</td>
<td>Program Memory (bytes)</td>
<td>Program Memory Speed Optimization</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td></td>
<td>Size Optimization (-0s)</td>
<td>-03</td>
</tr>
<tr>
<td>Required Stack Code - ENC28J60 network controller</td>
<td>8244</td>
<td>23451</td>
</tr>
<tr>
<td>Required Stack Code - MRF24WB0M Wi-Fi controller</td>
<td>8244</td>
<td>25356</td>
</tr>
<tr>
<td>Announce(1)</td>
<td>+1809</td>
<td>+2061</td>
</tr>
<tr>
<td>AutoIP</td>
<td>+2220</td>
<td>+2307</td>
</tr>
<tr>
<td>DHCP Client(1)</td>
<td>+4395</td>
<td>+4965</td>
</tr>
<tr>
<td>DHCP Server(1)</td>
<td>+3090</td>
<td>+3873</td>
</tr>
<tr>
<td>DNS Client(1)</td>
<td>+3840</td>
<td>+4725</td>
</tr>
<tr>
<td>Dynamic DNS Client(1,2,3)</td>
<td>+15675</td>
<td>+23982</td>
</tr>
<tr>
<td>HTTP2 Server/MPFS2(2)</td>
<td>+18780</td>
<td>+30114</td>
</tr>
<tr>
<td>ICMP Client</td>
<td>+1356</td>
<td>+1449</td>
</tr>
<tr>
<td>ICMP Server</td>
<td>+348</td>
<td>+375</td>
</tr>
<tr>
<td>NBNS(1)</td>
<td>+2844</td>
<td>+3165</td>
</tr>
<tr>
<td>SMTP Client(1,2,3)</td>
<td>+16323</td>
<td>+26709</td>
</tr>
<tr>
<td>SNMP Agent(1,4)</td>
<td>+14280</td>
<td>+25041</td>
</tr>
<tr>
<td>SNTP Client(1,3)</td>
<td>+4644</td>
<td>+5715</td>
</tr>
<tr>
<td>SSL Client</td>
<td>+24144</td>
<td>+44760</td>
</tr>
<tr>
<td>SSL Server</td>
<td>+24297</td>
<td>+45867</td>
</tr>
<tr>
<td>SSL Server/Client</td>
<td>+27888</td>
<td>+53724</td>
</tr>
<tr>
<td>TCP</td>
<td>+7989</td>
<td>+14550</td>
</tr>
<tr>
<td>UDP</td>
<td>+1311</td>
<td>+1503</td>
</tr>
<tr>
<td>Total(5)</td>
<td>57960</td>
<td>97617</td>
</tr>
</tbody>
</table>
Notes:
(1): Code size includes the UDP module (required for operation).
(2): Code size includes the TCP module (required for operation).
(3): Code size includes the DNS module (required for operation).
(4): Code size includes the MPFS2 module (required for operation).
(5): Total includes the required files, and the ARP, AutoIP, DHCP Client and Server, DNS, Dynamic DNS, HTTP2, ICMP, MPFS2, NBNS, SMTP, SNMP, SNTP, TCP, and UDP modules.
(6): Compiled code size exceeds the capacity of the target processor.

<table>
<thead>
<tr>
<th>C32</th>
<th>Program Memory (bytes)</th>
<th>Program Memory Speed Optimization (-0s)</th>
<th>Global Data (byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required Stack Code - ENC28J60 network controller</td>
<td>+14192</td>
<td>32636</td>
<td>108</td>
</tr>
<tr>
<td>Required Stack Code - MRF24WB0M Wi-Fi controller</td>
<td>+24288</td>
<td>34632</td>
<td>195</td>
</tr>
<tr>
<td>Announce(1)</td>
<td>+3018</td>
<td>+3214</td>
<td>+28</td>
</tr>
<tr>
<td>AutoIP</td>
<td>+2644</td>
<td>+2764</td>
<td>+64</td>
</tr>
<tr>
<td>DHCP Client(1)</td>
<td>+6164</td>
<td>+6384</td>
<td>+80</td>
</tr>
<tr>
<td>DHCP Server(1)</td>
<td>+4192</td>
<td>+4376</td>
<td>+36</td>
</tr>
<tr>
<td>DNS Client(1)</td>
<td>+4988</td>
<td>+5656</td>
<td>+76</td>
</tr>
<tr>
<td>Dynamic DNS Client(1,2,3)</td>
<td>+23658</td>
<td>+35402</td>
<td>+397</td>
</tr>
<tr>
<td>HTTP2 Server/MPFS2(2)</td>
<td>+29362</td>
<td>+43236</td>
<td>+351</td>
</tr>
<tr>
<td>ICMP Client</td>
<td>+1360</td>
<td>+1336</td>
<td>+36</td>
</tr>
<tr>
<td>ICMP Server</td>
<td>+360</td>
<td>+372</td>
<td>+0</td>
</tr>
<tr>
<td>NBNS(1)</td>
<td>+3924</td>
<td>+4152</td>
<td>+40</td>
</tr>
<tr>
<td>SMTP Client(1,2,3)</td>
<td>+26002</td>
<td>+40638</td>
<td>+353</td>
</tr>
<tr>
<td>SNMP Agent(1,4)</td>
<td>+20979</td>
<td>+32247</td>
<td>+350</td>
</tr>
<tr>
<td>SNTP Client(1,3)</td>
<td>+5886</td>
<td>+6674</td>
<td>+108</td>
</tr>
<tr>
<td>SSL Client</td>
<td>+37622</td>
<td>+58650</td>
<td>+581</td>
</tr>
<tr>
<td>SSL Server</td>
<td>+37726</td>
<td>+60946</td>
<td>+595</td>
</tr>
<tr>
<td>SSL Server/Client</td>
<td>+42690</td>
<td>+67890</td>
<td>+613</td>
</tr>
<tr>
<td>TCP</td>
<td>+13924</td>
<td>+24656</td>
<td>+214</td>
</tr>
<tr>
<td>UDP</td>
<td>+2356</td>
<td>+2476</td>
<td>+20</td>
</tr>
<tr>
<td>Total(5)</td>
<td>85143</td>
<td>131412</td>
<td>1181</td>
</tr>
<tr>
<td>Total + SSL(5)</td>
<td>112948</td>
<td>173585</td>
<td>1566</td>
</tr>
</tbody>
</table>

Notes:
(1): Code size includes the UDP module (required for operation).
(2): Code size includes the TCP module (required for operation).
(3): Code size includes the DNS module (required for operation).
(4): Code size includes the MPFS2 module (required for operation).
(5): Total includes the required files, and the ARP, AutoIP, DHCP Client and Server, DNS, Dynamic DNS, HTTP2, ICMP, MPFS2, NBNS, SMTP, SNMP, SNTP, TCP, and UDP modules.
(6): Compiled code size exceeds the capacity of the target processor.
Peripheral Usage

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Specific/Configurable</th>
<th>Polled/Interrupt</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>Timer 0 for PIC18, Timer 1 otherwise</td>
<td>Interrupt.</td>
<td>Used to implement a tick timer</td>
</tr>
<tr>
<td>SPI or PMP</td>
<td>Select via #define in HardwareProfile.h. See Hardware Configuration</td>
<td>Polled.</td>
<td>The SPI module is used to drive the ENC28J60 or MRF24WB0M / MRF24WG0M. An ENCX24J600 can be driven by the SPI module or a PMP module.</td>
</tr>
<tr>
<td>SPI</td>
<td>Select via #define in HardwareProfile.h. See External Storage</td>
<td>Polled.</td>
<td>Used to interface to an EEPROM or Serial Flash chip, as an option to store web pages for MPFS/MPFS2 or the AppConfig structure.</td>
</tr>
<tr>
<td>SPI</td>
<td>Select via #define in HardwareProfile.h. See External Storage</td>
<td>Polled.</td>
<td>Used to interface to a serial RAM as an optional socket allocation method.</td>
</tr>
</tbody>
</table>

Release Notes > Peripheral Usage
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 ENC24J600 are stand-alone Ethernet controller chips, developed by Microchip Technology. The MRF24WB0M / MRF24WG0M is a stand-alone 802.11b/g 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 section.

<table>
<thead>
<tr>
<th>Feature</th>
<th>ENC28J60</th>
<th>ENC24J600</th>
<th>PIC18F97J60</th>
<th>MRF24WB0M / MRF24WG0M</th>
<th>PIC32MX7XX/6XX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>Wired Ethernet</td>
<td>Wired Ethernet</td>
<td>Wired Ethernet</td>
<td>802.11b/g Wireless</td>
<td>Wired Ethernet</td>
</tr>
<tr>
<td>MAC</td>
<td>Internal</td>
<td>Internal</td>
<td>Internal</td>
<td>Internal</td>
<td>Internal</td>
</tr>
<tr>
<td>PHY</td>
<td>Internal (10-Base-T)</td>
<td>Internal (10/100-Base-T)</td>
<td>Internal (10-Base-T)</td>
<td>Internal</td>
<td>External PHY (MII/RMII Interface)</td>
</tr>
<tr>
<td>RAM Buffer (bytes)</td>
<td>8,192</td>
<td>24,576</td>
<td>3,808</td>
<td>14,170</td>
<td>Configurable descriptors in Internal RAM</td>
</tr>
<tr>
<td>Interface</td>
<td>SPI</td>
<td>SPI, 8 or 16 bit multiplexed or demultiplexed parallel interface</td>
<td>None (built-in Ethernet MAC/PHY)</td>
<td>SPI</td>
<td>None (built-in Ethernet MAC)</td>
</tr>
<tr>
<td>Pins</td>
<td>28</td>
<td>44, 64</td>
<td>64/80/100</td>
<td>36</td>
<td>64/100/121</td>
</tr>
<tr>
<td>Package</td>
<td>SOIC, SPDIP, SSOP,</td>
<td>TQFP, QFN</td>
<td>TQFP</td>
<td>Surface Mount WiFi I/O</td>
<td>TQFP, QFN</td>
</tr>
<tr>
<td></td>
<td>QFN (6x6 mm)</td>
<td>module</td>
<td>mm), BG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------</td>
<td>--------</td>
<td>---------</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>Cryptographic Engines</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Pre-programmed MAC address</td>
<td>No(1)</td>
<td>Yes</td>
<td>No(1)</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

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.
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.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP Configuration Wizard</td>
<td>Describes the TCP/IP Configuration Wizard.</td>
</tr>
<tr>
<td>MPFS2 Utility</td>
<td>Documentation for the MPFS2 utility</td>
</tr>
<tr>
<td>Hash Table Filter Entry Calculator</td>
<td>Describes the Ethernet Hash Table Calculator.</td>
</tr>
<tr>
<td>Microchip TCP/IP Discoverer</td>
<td>Describes the Microchip TCP/IP Discoverer project.</td>
</tr>
</tbody>
</table>

Software

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
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.
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 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 or FTP server.

The source code for this application is included in the Microchip Applications Libraries installer.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building MPFS2 Images</td>
<td>Describes how to build an MPFS Image</td>
</tr>
<tr>
<td>Uploading Pre-built MPFS2 Images</td>
<td>Procedure to upload images to external storage</td>
</tr>
<tr>
<td>Advanced MPFS2 Settings</td>
<td>Working with advanced configuration options</td>
</tr>
<tr>
<td>MPFS2 Command Line Options</td>
<td>How to use the MPFS2 Utility from the command line</td>
</tr>
</tbody>
</table>

Software > MPFS2 Utility
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.

![Source Settings]

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).

![Processing Options]

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.

![Output Files]

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 target host name (or IP address), upload protocol, and upload path may need to be changed to the one chosen when the board was first configured. You may also need to modify the user name and password used to access the secured functionality in your application, like web page upload. Use the **Settings** button to edit these values.

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.
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 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 target host name (or IP address), upload protocol, and upload path may need to be changed to the one chosen when the board was first configured. You may also need to modify the user name and password used to access the secured functionality in your application, like web page upload. 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.
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.
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:

<table>
<thead>
<tr>
<th>Switch</th>
<th>Short</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/BIN</td>
<td>/b</td>
<td>Output a BIN image (Default)</td>
</tr>
<tr>
<td>/C18_C32</td>
<td>/c</td>
<td>Output a C18 or XC32 image</td>
</tr>
<tr>
<td>/ASM16</td>
<td>/s</td>
<td>Output an ASM16 image</td>
</tr>
<tr>
<td>/mpfs2</td>
<td>/2</td>
<td>Use the MPFS2 format (Default)</td>
</tr>
<tr>
<td>/html &quot;...&quot;</td>
<td>/h &quot;...&quot;</td>
<td>File types to be parsed for dynamic variables (Default: &quot;*.htm, *.html, *.cgi, *.xml&quot;)</td>
</tr>
<tr>
<td>/xgzip &quot;...&quot;</td>
<td>/z &quot;...&quot;</td>
<td>File types to be excluded from GZIP compression (Default: &quot;*.bib, *.inc&quot;)</td>
</tr>
</tbody>
</table>

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.
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.
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 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 protocol enabled are connected to the network, they will respond with a UDP packet containing their host name (NBNS) 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.
### 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 **TCP/IP Demo App**. Most of this setup information can be applied to get started with other demo applications as well.

#### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware Setup</strong></td>
<td>Walks through hardware configuration for supported development platforms</td>
</tr>
<tr>
<td><strong>Programming and First Run</strong></td>
<td>Programming the device and clearing EEPROM for first run</td>
</tr>
<tr>
<td><strong>Configure your WiFi Access Point</strong></td>
<td>Configure the access point to work with the development kit. For information on configuring the MRF24WB0M / MRF24WG0M for your AP refer to “Configuring WiFi Security”</td>
</tr>
<tr>
<td><strong>Connecting to the Network</strong></td>
<td>Connecting your development board to the network</td>
</tr>
<tr>
<td><strong>Uploading Web Pages</strong></td>
<td>Uploading web pages to the device</td>
</tr>
<tr>
<td><strong>Accessing the Demo Application</strong></td>
<td>Accessing the demo application</td>
</tr>
<tr>
<td><strong>Configuring WiFi Security</strong></td>
<td>Configures the wireless access point and demo code for wireless security</td>
</tr>
</tbody>
</table>

---

### Getting Started

Microchip TCP/IP Stack 5.42.08 - June 15, 2013  
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] | [Index] | [Home]
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.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daughter Boards</td>
<td>Describes the daughter boards used to provide TCP/IP functionality to Microchip Demo boards.</td>
</tr>
<tr>
<td>PICDEM.net 2</td>
<td>Development platform for PIC18F97J60 series 8-bit MCU with integrated Ethernet MAC+PHY</td>
</tr>
<tr>
<td>PIC18 Explorer</td>
<td>Development platform for 8-bit parts using external ENC28J60, ENC424J600, ENC624J600, or MRF24WB0MA / MRF24WG0MA MAC+PHY.</td>
</tr>
<tr>
<td>Explorer 16 and PIC32 Starter Kit</td>
<td>Development platforms for 16- and 32-bit parts using external ENC28J60, ENC424J600, ENC624J600, or MRF24WB0M / MRF24WG0M MAC+PHY.</td>
</tr>
<tr>
<td>PIC24FJ256DA210 Dev Board</td>
<td>Development platform for the PIC24FJ256DA210 using external ENC28J60, ENC424J600, ENC624J600, or MRF24WB0M / MRF24WG0M MAC+PHY.</td>
</tr>
<tr>
<td>Wi-Fi G Demo Board</td>
<td>Describes the wireless 802.11b/g demo board based on Microchip 802.11b/g MRF24WG0MA RF transceiver and PIC32MX695F512H processor to provide Wi-Fi connectivity.</td>
</tr>
</tbody>
</table>
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 or the PIC18 Explorer board (populated with a PIC18 processor)

and/or

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

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 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/g WiFi PICtail Plus Daughter Board
The Microchip 802.11b/g 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 (802.11b: 1 to 2Mbps) or MRF24WG0MA (802.11b/g) module, which includes a Wi-Fi transceiver and associated circuit elements.

MRF24WB0MA supports both infrastructure and adhoc network types. MRF24WG0MA supports more extensive features covering infrastructure, adhoc, Wi-Fi Direct (peer-to-peer) and softAP network types. In addition, MRF24WG0MA supports Wi-Fi Protected Setup (WPS).

Visit the Microchip Web Site to view more information on Wireless Solutions and the 802.11b/g WiFi PICtail Product Page.
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 5.42.08.
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 ENCX24J600 configuration page.

Using the Microchip MRF24WB0MA / MRF24WG0MA 802.11b/g 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 in the Getting Started section of this help file provide a tutorial for setting up the generic TCPIP demo application.
PIC18 Explorer

Visit the Microchip web site to view the PIC18 Explorer a 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 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 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 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 ENCX24J600 configuration page.

**Using the Microchip MRF24WB0MA / MRF24WG0MA 802.11b/g 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 in the Getting Started section of this help file provide a tutorial for setting up the generic TCPIP demo application.
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 the same PICtail Plus board that connect to the Explorer 16.

Using the Ethernet PICtail Plus

To enable testing and debugging of the ENC28J60 on these boards, you must connect 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 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 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** 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 PSPCFG3, PSPCFG2, PSPCFG1&4, PMA to AD, and PMA to A 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 ENCX24J600 configuration page.

**Using the Microchip MRF24WB0MA / MRF24WG0MA 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 in the Getting Started section of this help file provide a tutorial for setting up the generic TCP/IP demo application.
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 MRF24WB0MA / MRF24WG0MA 802.11b/g 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.
Wi-Fi G Demo Board

Wi-Fi G Demo Board provides a low-cost and portable development system for Microchip MRF24WG0MA 802.11b/g RF Transceiver. The Wi-Fi G Demo Board is preloaded with the demo software for the user to explore the features of the MRF24WG0MA RF Transceiver. It is also expandable through a 8-pin expansion port interface, which allows the user to extend its functionality by adding various sensor expansion circuit designs.

Wi-Fi® G Demo Board
(Part # DV102412)

Wi-Fi G Demo Board source code is featured in MLA v5.42.06 v2013-
02-15 and future releases.

Visit the Microchip Web Site to view more information on Wireless Solutions and the Wi-Fi G Demo Board Product Page.
Programming and First Run

Once the hardware is configured, 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 / MRF24WG0M WiFi PICtail, you’ll need to configure your wireless access point first. For all Ethernet devices, Connect your Development Board to your network.
Configure your WiFi Access Point

To run the Wi-Fi demos with the MRF24WB0M / MRF24WG0M 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". 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.

![Image of Basic Wireless Settings](image)

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

<table>
<thead>
<tr>
<th>SSID</th>
<th>MicrochipDemoAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>None</td>
</tr>
<tr>
<td>Channel</td>
<td>Either 1, 6, or 11</td>
</tr>
</tbody>
</table>

You should have settings similar to the following:
Once the network is setup, you can connect your device to the network.
Connecting to the Network

All devices on a TCP/IP network must be assigned an IP address. Whereas the MAC address 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 topic).

The demo application comes with both a DHCP server and DHCP client configured. This allows the board to connect 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 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.

Getting Started > Connecting to the Network
Uploading Web Pages

Web pages are stored as an **MPFS2** image. This image can be placed in either **external non-volatile storage** (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 your board can be accessed at http://mchpboard. This host name uses the **NetBIOS Name Service**. 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**.

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](http://mchpboard/mpfsupload).

---

*Microchip TCP/IP Stack 5.42.08 - June 15, 2013*

Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] | [Index] | [Home]
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.

![Demo Application Screenshot](image)

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 web page, or in the HTTP2 server
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 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 topic.

Once you have finished exploring the demo application, you can proceed to the Stack API 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.
Configuring WiFi Security

The MRF24WB0M / MRF24WG0M can be configured to connect to wireless networks with encryption enabled. Both MRF24WB0M / MRF24WG0M supports WEP (40-bit and 104-bit), as well as WPA (TKIP) and WPA2 (TKIP/AES). In addition, MRF24WG0M supports WiFi Protected Setup (WPS) both Push Button Configuration (WPS-PBC) and Personal Information Number (WPS-PIN). MRF24WG0M also supports WPA2 Enterprise (EAP-PEAP/ EAP-TTLS), requiring special approval from Microchip.

Device Security Modes

Security settings for the MRF24WB0M / MRF24WG0M are located in the file \WF_Config.h\h. To enable security features the \#define preprocessor definition for \WF_SECURITY_Wifi_SECURITY_MODE\ must be defined as one of the following options:

<table>
<thead>
<tr>
<th>Security Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_SECURITY_WEP_40</td>
<td>40-bit WEP security. This equates to 5 ASCII characters or 10 hex digits.</td>
</tr>
<tr>
<td>MY_DEFAULT_WEP_KEYS_40</td>
<td>Contains up to four keys that can be programmed (default is key 0).</td>
</tr>
<tr>
<td>WF_SECURITY_WEP_104</td>
<td>104-bit WEP security. This equates to 13 ASCII characters or 26 hex digits.</td>
</tr>
<tr>
<td>MY_DEFAULT_WEP_KEYS_104</td>
<td>Contains up to four keys that can be programmed (default is key 0).</td>
</tr>
<tr>
<td>WF_SECURITY_WPA_WITH_KEY</td>
<td>Uses the 32 bytes in \WF_SECURITY_WPA\h as the key to join the network. The values are generated from a hash of the SSID name and WPA passphrase. For the purpose of the demo, the 32 bytes in \WF_SECURITY_WPA\h correspond to an SSID of “MicrochipDemoAP” and...</td>
</tr>
</tbody>
</table>
passphrase "Microchip 802.11 Secret PSK Password".

Instructs the MRF24WB0M / MRF24WG0M 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" that it takes approximately 30 seconds for the MRF24WB0M / MRF24WG0M to calculate this value[1].

Supported by MRF24WG0M only.
For WPS-PBC, define the MY_DEFAULT_SSID_NAME as "".
For WPS-PIN, define the MY_DEFAULT_WPS_PIN to be the same as the AP/router PIN, for example, 12390212 and define the MY_DEFAULT_SSID_NAME to be the same as the AP/router's SSID.

Supported by MRF24WG0M only. Requires MLA v5.42.06 March 2013 release and future releases. EAP-PEAP/MSCHAPv2 and EAP-TTLS/MSCHAPv2. Special approval needs to be submitted to marketing.

**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 / MRF24WG0M 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 / MRF24WG0M 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 / MRF24WG0M. The host can then save this key to external non-volatile memory. On future connection attempts, the host can program the MRF24WB0M / MRF24WG0M with the WF_SECURITY_WPA/WPA2/WPA_AUTO_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](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.

**Wi-Fi Protected Setup (WPS)**

WiFi Protected Setup (WPS) allows users to set up and expand the WiFi networks with security enabled, even if they are not familiar with the underlying technologies or processes involved. For example, users no longer have to know that SSID refers to the network name or WPA2 refers to the security mechanism. WPS will configure the network name SSID and security key for the AP and WPS client devices on a network. It supports the WEP / WPA / WPA2 security methods.

AP/routers from 2007 onwards will have this WPS feature.
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.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo Compatibility Table</td>
<td>Describes the hardware platforms that are compatible with each Demo.</td>
</tr>
<tr>
<td>Available Demos</td>
<td>The TCP/IP Stack comes with several example applications. These applications are described in the following sections.</td>
</tr>
</tbody>
</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\TCP\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

<table>
<thead>
<tr>
<th>Demo Board</th>
<th>Processor</th>
<th>MAC/PHY Layer</th>
<th>Comm. Bus</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC18 Explorer</td>
<td>18F87J11</td>
<td>ENC28J60</td>
<td>SPI</td>
<td>Requires silicon revision A4 or later.</td>
</tr>
<tr>
<td>PIC18 Explorer</td>
<td>18F87J11</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC18 Explorer</td>
<td>18F87J11</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC18 Explorer</td>
<td>18F87J50</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC18 Explorer</td>
<td>18F87J50</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC18 Explorer</td>
<td>18F87J50</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC18 Explorer</td>
<td>18F8722</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC18 Explorer</td>
<td>18F8722</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC18 Explorer</td>
<td>18F8722</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PICDEM.net 2</td>
<td>18F97J60</td>
<td>ETH97J60</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PICDEM.net 2</td>
<td>18F97J60</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PICDEM.net 2</td>
<td>18F97J60</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PICDEM.net 2</td>
<td>18F97J60</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ128GA010</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ128GA010</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ128GA010</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ128GA010</td>
<td>ENCX24J600</td>
<td>PSP 5 Indirect</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GA110</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GA110</td>
<td>ENC24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GA110</td>
<td>ENCX24J600</td>
<td>PSP 5 Indirect</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GA110</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GA110</td>
<td>ENCX24J600</td>
<td>PSP 5 Indirect</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GA110</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GB110</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GB110</td>
<td>ENC24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GB110</td>
<td>ENCX24J600</td>
<td>PSP 5 Indirect</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GB110</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GB210</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GB210</td>
<td>ENC24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GB210</td>
<td>ENCX24J600</td>
<td>PSP 5 Indirect Bitbang</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ256GB210</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>33FJ256GP710</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>33FJ256GP710</td>
<td>ENC24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>33FJ256GP710</td>
<td>ENCX24J600</td>
<td>PSP 5 Indirect</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>33FJ256GP710</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX360F512L</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>33EP512MU810</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>33EP512MU810</td>
<td>ENC28J60</td>
<td>SPI2</td>
<td></td>
</tr>
<tr>
<td>Device</td>
<td>Part Number</td>
<td>Microcontroller</td>
<td>Ethernet Controller</td>
<td>Interface</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------</td>
<td>----------------</td>
<td>---------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24EP512GU810</td>
<td>ENC28J60</td>
<td>SPI2</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX360F512L</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX360F512L</td>
<td>ENCX24J600</td>
<td>PSP 5 Indirect</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX360F512L</td>
<td>ENCX24J600</td>
<td>PSP 9</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX360F512L</td>
<td>MRF24WB0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX460F512L</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX460F512L</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX460F512L</td>
<td>MRF24WB0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX795F512L</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX795F512L</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX795F512L</td>
<td>MRF24WB0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC24FJ256DA210 Development Board</td>
<td>24FJ256DA210</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC24FJ256DA210 Development Board</td>
<td>24FJ256DA210</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC24FJ256DA210 Development Board</td>
<td>24FJ256DA210</td>
<td>ENCX24J600</td>
<td>PSP 5 Indirect Bitbang</td>
<td></td>
</tr>
<tr>
<td>PIC24FJ256DA210 Development Board</td>
<td>24FJ256DA210</td>
<td>MRF24WB0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC32 General Purpose Starter Kit (DM320001)</td>
<td>32MX360F512L</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC32 General Purpose Starter Kit (DM320001)</td>
<td>32MX360F512L</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC32 General Purpose Starter Kit (DM320001)</td>
<td>32MX360F512L</td>
<td>ENCX24J600</td>
<td>PSP 5 Indirect</td>
<td></td>
</tr>
<tr>
<td>PIC32 General Purpose Starter Kit (DM320001)</td>
<td>32MX360F512L</td>
<td>MRF24WB0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC32 USB Starter Kit (DM320003_2)</td>
<td>32MX795F512L</td>
<td>ENC28J60</td>
<td>SPI2</td>
<td></td>
</tr>
<tr>
<td>PIC32 USB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## TCPIP WebVend

<table>
<thead>
<tr>
<th>Demo Board</th>
<th>Processor</th>
<th>MAC/PHY Layer</th>
<th>Comm. Bus</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PICDEM.net 2</td>
<td>18F97J60</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PICDEM.net 2</td>
<td>18F97J60</td>
<td>ETH97J60</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ128GA010</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ128GA010</td>
<td>ENC24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ128GA010</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Demo Board</td>
<td>Processor</td>
<td>MAC/PHY Layer</td>
<td>Comm. Bus</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>Explorer 16</td>
<td>33FJ256GP710</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>33FJ256GP710</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>33FJ256GP710</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX360F512L</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX360F512L</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX360F512L</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX460F512L</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX460F512L</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX460F512L</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX795F512L</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX795F512L</td>
<td>ENCX24J600</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX795F512L</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC24FJ256DA210</td>
<td>PIC24FJ256DA210</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
</tbody>
</table>

**TCPIP WiFi EasyConfig Demo App**

**TCPIP WiFi Console Demo App**
<table>
<thead>
<tr>
<th>Demo Board</th>
<th>Processor</th>
<th>MAC/PHY Layer</th>
<th>Comm. Bus</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PICDEM.net 2</td>
<td>18F97J60</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>24FJ128GA010</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>33FJ256GP710</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX360F512L</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX460F512L</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>Explorer 16</td>
<td>32MX795F512L</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>PIC24FJ256DA210 Development Board</td>
<td>PIC24FJ256DA210</td>
<td>MRF24WB0M MRF24WG0M</td>
<td>SPI</td>
<td></td>
</tr>
</tbody>
</table>

**TCPIP Internet Radio App**

<table>
<thead>
<tr>
<th>Demo Board</th>
<th>Processor</th>
<th>MAC/PHY Layer</th>
<th>Comm. Bus</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Radio Board</td>
<td>18F67J60</td>
<td>ENC28J60</td>
<td>SPI</td>
<td></td>
</tr>
</tbody>
</table>

**TCPIP Internet Bootloader**

<table>
<thead>
<tr>
<th>Demo Board</th>
<th>Processor</th>
<th>MAC/PHY Layer</th>
<th>Comm. Bus</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>18F66J60</td>
<td>ETH97J60</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>18F66J60</td>
<td>ETH97J60</td>
<td>-</td>
<td>Extended Instruction Mode</td>
</tr>
<tr>
<td>N/A</td>
<td>18F66J65</td>
<td>ETH97J60</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>18F66J65</td>
<td>ETH97J60</td>
<td>-</td>
<td>Extended Instruction Mode</td>
</tr>
<tr>
<td>N/A</td>
<td>18F67J60</td>
<td>ETH97J60</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>18F67J60</td>
<td>ETH97J60</td>
<td>-</td>
<td>Extended Instruction Mode</td>
</tr>
<tr>
<td>N/A</td>
<td>18F86J60</td>
<td>ETH97J60</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>18F86J60</td>
<td>ETH97J60</td>
<td>-</td>
<td>Extended Instruction Mode</td>
</tr>
<tr>
<td>N/A</td>
<td>18F86J65</td>
<td>ETH97J60</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
### TCPIP MDD Demo App

<table>
<thead>
<tr>
<th>Demo Board</th>
<th>Processor</th>
<th>MAC/PHY Layer</th>
<th>Comm. Bus</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explorer 16</td>
<td>PIC24FJ256GB110</td>
<td>ENC28J60</td>
<td>SPI</td>
<td>Uses USB Thumb Drive as a storage medium for web pages.</td>
</tr>
<tr>
<td>Explorer 16</td>
<td>PIC24FJ128GA010</td>
<td>ENC28J60</td>
<td>SPI</td>
<td>Uses SD Card as a storage medium for web pages.</td>
</tr>
</tbody>
</table>

### 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.

[Demo Information > Demo Compatibility Table](#)

---

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc.  All rights reserved.

[Contents] [Index] [Home]
Available Demos

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

### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo App</td>
<td>Main demo application for the TCP/IP Stack.</td>
</tr>
<tr>
<td>Internet Bootloader</td>
<td>Implements a TFTP bootloader for the PIC18F97J60 family of parts.</td>
</tr>
<tr>
<td>WebVend</td>
<td>Sample Web-Enabled Vending Machine application used by the TCP/IP Webinar series.</td>
</tr>
<tr>
<td>Internet Radio</td>
<td>Implements a stand-alone Internet radio receiver.</td>
</tr>
<tr>
<td>WiFi Console</td>
<td>Implements a wireless console demo application that accepts client commands.</td>
</tr>
<tr>
<td>WiFi EZConfig</td>
<td>Implements a wireless application to configure an embedded device for a network in which the device has no natural human interface mechanism (i.e. keyboard and display). This demo also demonstrates dynamic scanning of networks and getting back calculated PSK keys from the MRF24W device.</td>
</tr>
<tr>
<td>Demo App MDD</td>
<td>Variation of TCPIP Demo App that uses an SD card or USB Thumb Drive to store web pages.</td>
</tr>
<tr>
<td>Google PowerMeter</td>
<td>Implements a power monitoring application that uploads data to Google PowerMeter.</td>
</tr>
<tr>
<td>Energy Monitoring</td>
<td>Implements a power monitoring application that uploads data to Google PowerMeter using the PIC18F87J72 Energy Monitoring PiCtail Plus Daughter Board.</td>
</tr>
<tr>
<td>WiFi G Demo</td>
<td>Implements a wireless 802.11b/g low-cost and portable application to demonstrate Wi-Fi capabilities using MRF24WG0MA and PIC32MX695F512H processor.</td>
</tr>
</tbody>
</table>
The **TCP/IP 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](#). 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:

### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP Demo App Features by Hardware Platform</td>
<td>Describes which features of the TCP/IP Demo Application are available for each hardware configuration.</td>
</tr>
<tr>
<td>Demo Modules</td>
<td>Describes the custom modules used in this demo.</td>
</tr>
</tbody>
</table>

[Demo Information](#) > [Available Demos](#) > [Demo App](#)
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.

### Platforms

<table>
<thead>
<tr>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>PICDEM.net 2 - ENC28J60</td>
</tr>
<tr>
<td>PICDEM.net 2 - ETH97J60</td>
</tr>
<tr>
<td>PICDEM.net 2 - ENC624J600 (SPI)</td>
</tr>
<tr>
<td>PICDEM.net 2 - MRF24JWB0M</td>
</tr>
<tr>
<td>PIC18 Explorer - ENC28J60</td>
</tr>
<tr>
<td>PIC18 Explorer - ENC624J500 (SPI)</td>
</tr>
<tr>
<td>PIC18 Explorer - MRF24JWB0M</td>
</tr>
<tr>
<td>Explorer 16 - ENC28J60</td>
</tr>
<tr>
<td>Explorer 16 - ENC624J600 (SPI)</td>
</tr>
<tr>
<td>Explorer 16 - ENC624J600 (PSP 5 - Indirect)</td>
</tr>
<tr>
<td>Explorer 16 - ENC624J600 (PSP 9)</td>
</tr>
<tr>
<td>Explorer 16 - MRF24JWB0M</td>
</tr>
<tr>
<td>PIC24FJ256DA210 Dev Board - ENC28J60</td>
</tr>
<tr>
<td>PIC24FJ256DA210 Dev Board - ENC624J600 (SPI)</td>
</tr>
<tr>
<td>PIC24FJ256DA210 Dev Board - ENC624J600 (PSP 5 - Indirect)</td>
</tr>
<tr>
<td>PIC24FJ256DA210 Dev Board - MRF24JWB0M</td>
</tr>
<tr>
<td>PIC32 GP/USB Starter Kits - ENC28J60</td>
</tr>
<tr>
<td>PIC32 GP/USB Starter Kits - ENC624J600 (SPI)</td>
</tr>
<tr>
<td>PIC32 GP/USB Starter Kits - ENC624J600 (PSP 5 - Indirect)</td>
</tr>
<tr>
<td>PIC32 GP/USB Starter Kits - ENC624J600 (PSP 9)</td>
</tr>
<tr>
<td>PIC32 GP/USB Starter Kits - MRF24JWB0M</td>
</tr>
<tr>
<td>PIC32 Ethernet Starter Kit - ETH795</td>
</tr>
</tbody>
</table>

### Features

- MPFS EEPROM 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, 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 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.
Demo Modules

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

For Microchip 802.11b/g WiFi PICtail MRF24W, the following network types are supported.

- CFG_WF_INFRASTRUCTURE
- CFG_WF_ADHOC
- CFG_WF_P2P (supported by MRF24WG0M only)

For Microchip 802.11b/g WiFi PICtail MRF24W, the new security modes are supported.

- WF_SECURITY_WPS_PUSH_BUTTON (supported by MRF24WG0M only)
- WF_SECURITY_WPS_PIN (supported by MRF24WG0M only)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Page Demos</td>
<td>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.</td>
</tr>
<tr>
<td>E-mail (SMTP) Demo</td>
<td>Demonstrates how to use an e-mail client to send messages when events occur. This is a standalone demo; for the web page &quot;Send Email&quot; demo, see the Forms using POST topic.</td>
</tr>
<tr>
<td>Generic TCP Client</td>
<td>Demonstrates how to build a TCP Client application through an HTTP client example.</td>
</tr>
<tr>
<td>Generic TCP Server</td>
<td>Demonstrates how to build a TCP server application</td>
</tr>
<tr>
<td>Ping (ICMP) Demo</td>
<td>Demonstrates how to build a Ping client.</td>
</tr>
<tr>
<td>SNMP Server (Agent)</td>
<td>Describes the Simple Network Management Protocol Demo.</td>
</tr>
</tbody>
</table>
### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART-to-TCP Bridge</td>
<td>Describes how to use the UART-to-TCP Bridge demo.</td>
</tr>
<tr>
<td>Zero Configuration (ZeroConf)</td>
<td>Describes usage of the Zeroconf modules</td>
</tr>
</tbody>
</table>
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, processing web forms (using HTTP GET and POST), and providing status information through the output of dynamic variables.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Variables</td>
<td>Dynamic variables allow the PIC to output custom data when serving a web page.</td>
</tr>
<tr>
<td>Authentication</td>
<td>Authentication requires a correct username/password prompt before displaying select web pages.</td>
</tr>
<tr>
<td>Forms using GET</td>
<td>Web forms are easier to process when using the GET method, but the amount of data is limited.</td>
</tr>
<tr>
<td>Forms using POST</td>
<td>Forms using POST are more difficult to process, but large amounts of data can be submitted.</td>
</tr>
<tr>
<td>Cookies</td>
<td>Cookies allow storage of session information in a user's browser.</td>
</tr>
<tr>
<td>Functions</td>
<td>The following table lists functions in this documentation.</td>
</tr>
<tr>
<td>Variables</td>
<td>The following table lists variables in this documentation.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPPostImage</td>
<td>This is function HTTPPostImage.</td>
</tr>
<tr>
<td>HTTPPostConfig</td>
<td>Processes the configuration form on config/index.htm</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>HTTPPostSNMPCommunity</td>
<td>This is function HTTPPostSNMPCommunity.</td>
</tr>
<tr>
<td>HTTPPostDDNSConfig</td>
<td>Parsing and collecting http data received from http form.</td>
</tr>
<tr>
<td>HTTPPostEmail</td>
<td>Processes the e-mail form on email/index.htm</td>
</tr>
<tr>
<td>HTTPPostLCD</td>
<td>Processes the LCD form on forms.htm</td>
</tr>
<tr>
<td>HTTPPostMD5</td>
<td>Processes the file upload form on upload.htm</td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag_ImageUpdate_running</td>
<td>This is variable Flag_ImageUpdate_running.</td>
</tr>
<tr>
<td>ImageUpdate_Addr</td>
<td>Processes the file upload form on upload_image.htm</td>
</tr>
<tr>
<td>DDNSData</td>
<td>RAM allocated for DDNS parameters</td>
</tr>
<tr>
<td>ImageUpdate_Checksum</td>
<td>This is variable ImageUpdate_Checksum.</td>
</tr>
<tr>
<td>ImageUpdate_Size</td>
<td>This is variable ImageUpdate_Size.</td>
</tr>
<tr>
<td>lastFailure</td>
<td>Stick status message variable. See lastSuccess for details.</td>
</tr>
<tr>
<td>lastSuccess</td>
<td>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.</td>
</tr>
</tbody>
</table>
Dynamic Variables

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.

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 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](#) 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.

```html
<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.
Authentication

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.

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.

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` 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.

Module

Web Page Demos

Demo Information > Available Demos > Demo App > Demo Modules > Web Page Demos > Authentication
Forms using GET

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.

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 topic for more information.

3. Navigate to the Form Processing page using the navigation panel on the left of the page.

4. 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.

```
4: Off  3: Off  2: Off  1: Off

Save
```

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 first.

1. Start by opening forms.htm in your "TCPIP Demo App\WebPages2" folder.
2. Locate the GET method implementation the 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.

3. Create a new `select` input for LED5.

4. Open `CustomHTTPApp.c` in the TCP/IP Demo App MPLAB project.

5. Verify that the `HTTPPrint_ledSelected` dynamic variable callback function has been implemented for LED5.

6. Find the `HTTPExecuteGet` function. Locate the section of code the processes GET arguments for the `forms.htm` file.

7. Add implementation to search for the "led5" argument string in the GET data buffer and then set LED5_IO based on the associated value.

Module

**Web Page Demos**
Forms using POST

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.

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 Form Processing page using the navigation panel on the left of the page.
3. 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.

4. Navigate to the File Uploads page using the navigation panel on the left of the page.
5. Browse 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.

6. 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 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 topic for more information.

![SMTP Server Configuration Example]

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

Module

Web Page Demos

Demo Information > Available Demos > Demo App > Demo Modules > Web Page Demos > Forms using POST

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
Cookies

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.

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.

<table>
<thead>
<tr>
<th>First Name:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Favorite:</td>
<td>PIC18</td>
</tr>
<tr>
<td></td>
<td>Set Cookies</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>not set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Favorite:</td>
<td>not implemented</td>
</tr>
</tbody>
</table>

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, GET, and cookie 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 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 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_cookiname for an example.

```c
void HTTPPrint_cookiname(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.
# Functions

## Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPPostImage</td>
<td>This is function HTTPPostImage.</td>
</tr>
<tr>
<td>HTTPPostConfig</td>
<td>Processes the configuration form on config/index.htm</td>
</tr>
<tr>
<td>HTTPPostSNMPCommunity</td>
<td>This is function HTTPPostSNMPCommunity.</td>
</tr>
<tr>
<td>HTTPPostDDNSConfig</td>
<td>Parsing and collecting http data received from http form.</td>
</tr>
<tr>
<td>HTTPPostEmail</td>
<td>Processes the e-mail form on email/index.htm</td>
</tr>
<tr>
<td>HTTPPostLCD</td>
<td>Processes the LCD form on forms.htm</td>
</tr>
<tr>
<td>HTTPPostMD5</td>
<td>Processes the file upload form on upload.htm</td>
</tr>
</tbody>
</table>

## Module

**Web Page Demos**

[Demo Information] > [Available Demos] > [Demo App] > [Demo Modules] > [Web Page Demos] > [Functions]
HTTPPostImage Function

```c
static HTTP_IO_RESULT HTTPPostImage();
```

Description

This is function HTTPPostImage.
HTTPPostConfig Function

```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**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_IO_DONE</td>
<td>all parameters have been processed</td>
</tr>
<tr>
<td>HTTP_IO_NEED_DATA</td>
<td>data needed by this function has not yet arrived</td>
</tr>
</tbody>
</table>
HTTPPostSNMPCommunity Function

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

Description

This is function HTTPPostSNMPCommunity.
HTTPPostDDNSConfig Function

```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) 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` is loaded.

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_IO_DONE</td>
<td>Finished with procedure</td>
</tr>
<tr>
<td>HTTP_IO_NEED_DATA</td>
<td>More data needed to continue, call again later</td>
</tr>
<tr>
<td>HTTP_IO_WAITING</td>
<td>Waiting for asynchronous process to complete, call again later</td>
</tr>
</tbody>
</table>

Demo Information > Available Demos > Demo App > Demo Modules > Web Page Demos > Functions > HTTPPostDDNSConfig Function

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
HTTPPostEmail Function

```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 obseleted, 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

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_IO_DONE</td>
<td>the message has been sent</td>
</tr>
<tr>
<td>HTTP_IO_WAITING</td>
<td>the function is waiting for the SMTP process to complete</td>
</tr>
<tr>
<td>HTTP_IO_NEED_DATA</td>
<td>data needed by this function has not yet arrived</td>
</tr>
</tbody>
</table>
HTTPPostEmail Function
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

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_IO_DONE</td>
<td>the parameter has been found and saved</td>
</tr>
<tr>
<td>HTTP_IO_WAITING</td>
<td>the function is pausing to continue later</td>
</tr>
<tr>
<td>HTTP_IO_NEED_DATA</td>
<td>data needed by this function has not yet arrived</td>
</tr>
</tbody>
</table>

Microchip TCP/IP Stack Help  Contents | Index | Home  Previous | Up | Next

HTTPPostLCD Function

static HTTP_IO_RESULT HTTPPostLCD();
HTTPPostMD5 Function

**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 CRLF CRLF, 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 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**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_IO_DONE</td>
<td>all parameters have been processed</td>
</tr>
<tr>
<td>HTTP_IO_WAITING</td>
<td>the function is pausing to continue later</td>
</tr>
<tr>
<td>HTTP_IO_NEED_DATA</td>
<td>data needed by this function has not yet arrived</td>
</tr>
</tbody>
</table>
## Variables

### Module

**Web Page Demos**

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag_ImageUpdate_running</td>
<td>This is variable Flag_ImageUpdate_running.</td>
</tr>
<tr>
<td>ImageUpdate.Addr</td>
<td>Processes the file upload form on upload(Image.htm)</td>
</tr>
<tr>
<td>DDNSData</td>
<td>RAM allocated for DDNS parameters</td>
</tr>
<tr>
<td>ImageUpdate.Checksum</td>
<td>This is variable ImageUpdate_Checksum.</td>
</tr>
<tr>
<td>ImageUpdate.Size</td>
<td>This is variable ImageUpdate_Size.</td>
</tr>
<tr>
<td>lastFailure</td>
<td>Stick status message variable. See lastSuccess for details.</td>
</tr>
<tr>
<td>lastSuccess</td>
<td>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.</td>
</tr>
</tbody>
</table>
Flag_ImageUpdate_running Variable

C

```c
UINT8 Flag_ImageUpdate_running = 0;
```

Description

This is variable Flag_ImageUpdate_running.
ImageUpdate.Addr Variable

C

UINT32 ImageUpdate.Addr = 0;

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 CRLFCRRLF, the file data begins, and we read it 16 bytes at a time and add that to the update. 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 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 ~uploadImage~ callback reads this data later to send back to the client.

Preconditions

None

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_IO_DONE</td>
<td>all parameters have been processed</td>
</tr>
<tr>
<td>HTTP_IO_WAITING</td>
<td>the function is pausing to continue later</td>
</tr>
<tr>
<td>HTTP_IO_NEED_DATA</td>
<td>data needed by this function has not yet arrived</td>
</tr>
</tbody>
</table>
DDNSData Variable

C

BYTE DDNSData[100];

Description

RAM allocated for DDNS parameters
ImageUpdate_Checksum Variable

C

UINT32 ImageUpdate_Checksum = 0;

Description

This is variable ImageUpdate_Checksum.

Demo Information > Available Demos > Demo App > Demo Modules > Web Page Demos > Variables > ImageUpdate_Checksum Variable
ImageUpdate_Size Variable

```c
UINT32 ImageUpdate_Size = 0;
```

Description

This is variable ImageUpdate_Size.
lastFailure Variable

```c
BOOL lastFailure = FALSE;
```

Description

Stick status message variable. See `lastSuccess` for details.
lastSuccess Variable

```c
BOOL lastSuccess = FALSE;
```

**Description**

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.
E-mail (SMTP) Demo

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. 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() function.
   1. Replace the initializer of the RAMStringTo[] array with the target email address.
   2. Replace 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 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.

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

The short-message SMTPDemo 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 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) 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.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMTPDemo</td>
<td>Demonstrates use of the e-mail (SMTP) client.</td>
</tr>
</tbody>
</table>

Demo Information > Available Demos > Demo App > Demo Modules > E-mail (SMTP) Demo

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
SMTPDemo Function

```c
void SMTPDemo();
```

**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` in CustomHTTPApp.c.

**Preconditions**

The SMTP client is initialized.

**Returns**

None

**Module**

E-mail (SMTP) Demo
Generic TCP Client

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 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. This socket will use a TCP_PURPOSE_GENERIC_TCP_CLIENT socket type from the TCP socket structure 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 125 bytes of data and then use an HTTP GET 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.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenericTCPClient</td>
<td>Implements a simple HTTP client (over TCP).</td>
</tr>
</tbody>
</table>

### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>The following table lists variables in this documentation.</td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoteURL</td>
<td>Defines the URL to be requested by this HTTP client</td>
</tr>
<tr>
<td>ServerName</td>
<td>Defines the server to be accessed for this application</td>
</tr>
<tr>
<td>ServerPort</td>
<td>Note that if HTTPS is used, the ServerName and URL must change to an SSL enabled server.</td>
</tr>
</tbody>
</table>

**Demo Information** 
- [Available Demos](#) 
- [Demo App](#) 
- [Demo Modules](#) 
- [Generic TCP Client](#)
GenericTCPClient Function

```c
void GenericTCPClient();
```

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.

Returns

None

Module

Generic TCP Client
## Variables

### Module

**Generic TCP Client**

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoteURL</td>
<td>Defines the URL to be requested by this HTTP client</td>
</tr>
<tr>
<td>ServerName</td>
<td>Defines the server to be accessed for this application</td>
</tr>
<tr>
<td>ServerPort</td>
<td>Note that if HTTPS is used, the ServerName and URL must change to an SSL enabled server.</td>
</tr>
</tbody>
</table>

---

**Demo Information** > **Available Demos** > **Demo App** > **Demo Modules** > **Generic TCP Client** > **Variables**

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
RemoteURL Variable

\begin{verbatim}
C
ROM BYTE RemoteURL[] = "/search?as_q=Microchip&as_sitesearch=microchip
\end{verbatim}

Description

Defines the URL to be requested by this HTTP client
ServerName Variable

| C |
| BYTE ServerName[] = "www.google.com";

Description

Defines the server to be accessed for this application
ServerPort Variable

C

WORD ServerPort = HTTP_PORT;

Description

Note that if HTTPS is used, the ServerName and URL must change to an SSL enabled server.
Generic TCP Server

Overview

The Generic TCP Server example demonstrates how to build a TCP server application. Once you connect 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 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** demo implements a task function with 3 states. In the first state, SM_HOME, the task will attempt to open a TCP server socket. This socket will use a **TCP_PURPOSE_GENERIC_TCP_SERVER socket type** from the TCP socket structure that was initialized in your configuration files. It will also listen on TCP port 9760 (defined by the macro **SERVER_PORT**).

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 for new connections.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GenericTCPServer</code></td>
<td>Implements a simple ToUpper TCP Server.</td>
</tr>
</tbody>
</table>

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macros</td>
<td>The following table lists macros in this documentation.</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>~--</td>
<td>SERVER_PORT</td>
</tr>
</tbody>
</table>

Demo Information > Available Demos > Demo App > Demo Modules > Generic TCP Server

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
GenericTCPServer Function

```c
void GenericTCPServer();
```

**Description**

This function implements a simple TCP server. The function is invoked periodically by the stack to listen 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.

**Returns**

None

**Module**

Generic TCP Server

---

Demo Information > Available Demos > Demo App > Demo Modules > Generic TCP Server > GenericTCPServer Function

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
## Macros

### Name | Description
---|---
SERVER PORT | Defines which port the server will listen on

## Module

### Generic TCP Server

[Demo Information > Available Demos > Demo App > Demo Modules > Generic TCP Server > Macros](#)
SERVER_PORT Macro

```
#define SERVER_PORT 9760
```

**Description**

Defines which port the server will listen on.
Ping (ICMP) Demo

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 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 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 callback function and take action depending on the return value:

<table>
<thead>
<tr>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>Remain in this state and keep waiting for a response.</td>
</tr>
<tr>
<td>-1</td>
<td>Write a message to the LCD indicating that the ping timed out. Change state to SM_HOME.</td>
</tr>
</tbody>
</table>
-3 Write a message to the LCD indicating that the DNS module couldn't resolve the target address. Change state to SM_HOME.

Other Convert the response time to a text string and print it to the LCD. Change state to SM_HOME.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PingDemo</td>
<td>Demonstrates use of the ICMP (Ping) client.</td>
</tr>
</tbody>
</table>

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macros</td>
<td>The following table lists macros in this documentation.</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST_TO_PING</td>
<td>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.</td>
</tr>
</tbody>
</table>

Demo Information > Available Demos > Demo App > Demo Modules > Ping (ICMP) Demo
PingDemo Function

C

```c
void PingDemo();
```

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.

Returns

None

Module

Ping (ICMP) Demo

Demo Information > Available Demos > Demo App > Demo Modules > Ping (ICMP) Demo > PingDemo Function
Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST_TO_PING</td>
<td>Address that ICMP client will ping. If the DNS client module</td>
</tr>
<tr>
<td></td>
<td>is not available in the stack, then this hostname is ignored</td>
</tr>
<tr>
<td></td>
<td>and the local gateway IP address will be pinged instead.</td>
</tr>
</tbody>
</table>

Module

Ping (ICMP) Demo

Demo Information > Available Demos > Demo App > Demo Modules > Ping (ICMP) Demo > Macros
HOST_TO_PING Macro

```c
#define HOST_TO_PING "ww1.microchip.com"  // Address that ICMP client will ping.
```

### Description

**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.
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 `#define STACK_USE_SNMP_SERVER`. The SNMPv3 server could be selectively enabled with independent macro `#define STACK_USE_SNMPV3_SERVER`. As SNMPv3 stacks are required to support SNMPv1 and SNMPv2c, enabling the SNMPv3 Server will automatically enable SNMPv1 and SNMPv2c servers in the Microchip TCPIP Stack SNMP agent. These macros are defined in the `TCPIP (MACPHY).h` file located at `<Installation Path>:\Microchip Solutions MAIN\TCPIP\Demo App\Configs\`.

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, and 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.

Note that even though SNMPv3 also requires SNMPv1 and SNMPv2c, a layer in the SNMP stack will prevent access to the variables that should be secured by SNMPv3. SNMP variables are structures in a tree in the Management Information Base (MIB). Access to parts of this tree are determined by version. For example, SYSTEM-type variables can be accessed regardless of SNMP version, SNMPv2c requests can access part of the tree, and authenticated SNMPv3 requests can access the complete tree.

**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.
## Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIB Files</td>
<td>Describes Management Information Base (MIB) files.</td>
</tr>
<tr>
<td>MIB Browsers</td>
<td>Describes SNMP MIB browsers.</td>
</tr>
<tr>
<td>SNMP Operations</td>
<td>Describes the SNMP operations supported in this demo.</td>
</tr>
<tr>
<td>SNMP Traps</td>
<td>Describes SNMP traps.</td>
</tr>
<tr>
<td>HTTP Configuration</td>
<td>Describes how to configure SNMP community names via HTTP.</td>
</tr>
</tbody>
</table>

## Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendNotification</td>
<td>Prepare, validate remote node which will receive trap and send trap pdu.</td>
</tr>
<tr>
<td>SNMPGetTimeStamp</td>
<td>Obtains the current Tick value for the SNMP time stamp.</td>
</tr>
</tbody>
</table>

## Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_TRY_TO_SEND_TRAP</td>
<td>Update the Non record id OID value which is part of CustomSnmpDemo.c file</td>
</tr>
<tr>
<td>SNMP_MAX_NON_REC_ID_OID</td>
<td>Default STACK_USE_SMIV2 is enabled . For Stack V5.31, STACK_USE_SMIV2 should be disabled.</td>
</tr>
<tr>
<td>STACK_USE_SMIV2</td>
<td>Default STACK_USE_SMIV2 is enabled . For Stack V5.31, STACK_USE_SMIV2 should be disabled.</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gSendTrapSMstate</td>
<td>This is variable gSendTrapSMstate.</td>
</tr>
<tr>
<td>gSnmpNonMibRecInfo</td>
<td>OLD snmp.mib file with SMIv1 standard</td>
</tr>
<tr>
<td>gSnmpv3UserSecurityName</td>
<td>This is variable gSnmpv3UserSecurityName.</td>
</tr>
<tr>
<td>gtrapSMStateUpdate</td>
<td>This is variable gtrapSMStateUpdate.</td>
</tr>
</tbody>
</table>
MIB Files

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.

Module

SNMP Server (Agent)
MIB Browsers

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 mchp.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 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 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.

<table>
<thead>
<tr>
<th></th>
<th>USER 1</th>
<th>USER 2</th>
<th>USER 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>USM User</td>
<td>microchip</td>
<td>SnmpAdmin</td>
<td>root</td>
</tr>
<tr>
<td>Security Level</td>
<td>auth, priv</td>
<td>auth, no priv</td>
<td>no auth, no priv</td>
</tr>
<tr>
<td>Auth Algorithm</td>
<td>MD5</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>Auth password</td>
<td>auth12345</td>
<td>ChandlerUS</td>
<td></td>
</tr>
<tr>
<td>Privacy Algorithm</td>
<td>AES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Privacy password</td>
<td>priv12345</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. 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.

6. 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 USER3 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.

7. Configure the IP address of the SNMP agent to the "Address" field.
7. 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.

8. Select the SNMP Get operation from the operations tab.

9. 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.

10. 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.

11. 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."

Module

SNMP Server (Agent)
SNMP Operations

Get

1. Select the "Advanced" tab and configure the SNMP version to '1' and the Read community to "public".
2. Select "Get" from the operations menu.
3. 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

1. 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.
2. 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 verison to V3, configre the credentails, again try a Get_Next operation for the sae 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.

![Image of SNMP interface](image)

**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 `ledB5` 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.
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.

Several preprocessor macro in the `TCPIPConfig.h` variant header file can be used to enable or disable traps in the agent. Commenting and un-commenting these macros in the file will have different effects. The `SNMP_TRAP_DISABLED` macro will disable traps entirely if it is not commented:

```c
#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 received and process SNMP V2 trap PDUs. To configure the trap format, comment or uncomment the following macro in the `TCPIPConfig.h` header file:

```c
#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.

To enable traps in SNMPv3, the `#define SNMP_V1_V2_TRAP_WITH_SNMPv3` macro must be uncommented.

The following table illustrates how to enable/disable traps for different versions of SNMP:

<table>
<thead>
<tr>
<th></th>
<th>SNMPv1</th>
<th>SNMPv2c</th>
<th>SNMPv3</th>
</tr>
</thead>
</table>

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 of 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 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 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 Failure trap.

![Trap Receiver](image)

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

SNMP Server (Agent)

<table>
<thead>
<tr>
<th>Description</th>
<th>Source</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.41.10.78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 minutes 15 seconds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.1.3.6.1.4.1.17095</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variable Bindings:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1.3.6.1.2.1.1.3.0</td>
<td><code>TimeTicks</code> 2 minutes 15 seconds</td>
</tr>
<tr>
<td>snmpTrapOID</td>
<td><code>OID</code> .1.3.6.1.4.1.17095</td>
</tr>
<tr>
<td>.1.3.6.1.4.1.17095.3.4.0</td>
<td><code>Integer</code> 820</td>
</tr>
<tr>
<td>.1.3.6.1.4.1.17095.3.3.0</td>
<td><code>Integer</code></td>
</tr>
<tr>
<td>.1.3.6.1.4.1.17095.3.1.0</td>
<td><code>Integer</code></td>
</tr>
</tbody>
</table>

Microchip TCP/IP Stack 5.42.08 · June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
HTTP Configuration

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.

Module

SNMP Server (Agent)
Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SendNotification</code></td>
<td>Prepare, validate remote node which will receive trap and send trap pdu.</td>
</tr>
<tr>
<td><code>SNMPGetTimeStamp</code></td>
<td>Obtains the current Tick value for the SNMP time stamp.</td>
</tr>
</tbody>
</table>

Module

**SNMP Server (Agent)**

Demo Information  >  Available Demos  >  Demo App  >  Demo Modules  >  SNMP Server (Agent)  >  Functions
SendNotification Function

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.

Preconditions

SNMPTrapDemo() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>receiverIndex</td>
<td>The index to array where remote ip address is stored.</td>
</tr>
<tr>
<td>var</td>
<td>SNMP var ID that is to be used in notification</td>
</tr>
<tr>
<td>val</td>
<td>Value of var. Only value of BYTE, WORD or DWORD can be sent.</td>
</tr>
<tr>
<td>targetIndex</td>
<td>snmpv3 target index</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If notification send is successful.</td>
</tr>
</tbody>
</table>
Remarks

None.
SNMPGetTimeTimeStamp Function

C

```c
static DWORD SNMPGetTimeTimeStamp();
```

Description

This function retrieves the absolute time measurements for SNMP time stamp. Use `TickGet` and `TickGetDiv64K` to collect all 48 bits of the internal Tick Timer.

Preconditions

None

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeStamp</td>
<td>DWORD timeval</td>
</tr>
</tbody>
</table>

Remarks

None.
### SNMP Server (Agent)

#### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gSendTrapSMstate</td>
<td>This is variable gSendTrapSMstate.</td>
</tr>
<tr>
<td>gSnmpNonMibRecInfo</td>
<td>OLD snmp.mib file with SMIv1 standard</td>
</tr>
<tr>
<td>gSnmpv3UserSecurityName</td>
<td>This is variable gSnmpv3UserSecurityName.</td>
</tr>
<tr>
<td>gtrapSMStateUpdate</td>
<td>This is variable gtrapSMStateUpdate.</td>
</tr>
</tbody>
</table>

[Demo Information] > [Available Demos] > [Demo App] > [Demo Modules] > [SNMP Server (Agent)] > [Variables]

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
gSendTrapSMstate Variable

C

UINT8 gSendTrapSMstate = 0;

Description

This is variable gSendTrapSMstate.
gSnmpNonMibRecInfo Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
</table>
| SNMPNONMIBRECDINFO gSnmpNonMibRecInfo[SNMP_MAX_NON_REC_ID_OID] = {{4

Description

OLD snmp.mib file with SMIv1 standard

Demo Information > Available Demos > Demo App > Demo Modules > SNMP Server (Agent) > Variables > gSnmpNonMibRecInfo Variable
gSnmpv3UserSecurityName Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE gSnmpv3UserSecurityName[SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE];</td>
</tr>
</tbody>
</table>

Description

This is variable gSnmpv3UserSecurityName.
gtrapSMStateUpdate Variable

C

BOOL gtrapSMStateUpdate = FALSE;

Description

This is variable gtrapSMStateUpdate.

Demo Information > Available Demos > Demo App > Demo Modules > SNMP Server (Agent) > Variables > gtrapSMStateUpdate Variable
### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_TRY_TO_SEND_TRAP</td>
<td>Update the Non record id OID value which is part of CustomSnmpDemo.c file</td>
</tr>
<tr>
<td>SNMP_MAX_NON_REC_ID_OID</td>
<td>Default STACK_USE_SMIV2 is enabled. For Stack V5.31, STACK_USE_SMIV2 should be disabled.</td>
</tr>
<tr>
<td>STACK_USE_SMIV2</td>
<td></td>
</tr>
</tbody>
</table>

### Module

**SNMP Server (Agent)**

[Demo Information] > [Available Demos] > [Demo App] > [Demo Modules] > [SNMP Server (Agent)] > [Macros]
MAX_TRY_TO_SEND_TRAP Macro

C

#define MAX_TRY_TO_SEND_TRAP (10)
SNMP_MAX_NON_REC_ID_OID Macro

C

#define SNMP_MAX_NON_REC_ID_OID 3

Description

Update the Non record id OID value which is part of CustomSnmpDemo.c file
STACK_USE_SMIV2 Macro

C

#define STACK_USE_SMIV2

Description

Default STACK_USE_SMIV2 is enabled. For Stack V5.31, STACK_USE_SMIV2 should be disabled.
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 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.
4. Open a command/DOS prompt. Type "telnet ip_address 9761" where ip_address is the IP address that you got from step 4.
6. As you type characters in the command prompt, they will be transmitted over the Telnet 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 session.

New IP Address: 169.254.1.1

New IP Address: 192.168.0.131

This string was typed in the telnet session.

This one was typed in the terminal.
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 / MRF24WG0M 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 accesses 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 directly to it. This further eliminates the need to know whether something exists on a network (and what it's 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 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.
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/](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

### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bootloader Design</td>
<td>Describes the design and theory of operation for the</td>
</tr>
<tr>
<td>Using the Bootloader</td>
<td>Describes two methods for using the bootloader.</td>
</tr>
</tbody>
</table>

**Demo Information > Available Demos > Internet Bootloader**

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] | [Index] | [Home]
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 \texttt{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 \texttt{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 \texttt{GOTO} $000004h$ instruction to ensure normal application operation if the first instruction was not a \texttt{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 \texttt{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 \texttt{RCON<NOT_POR>} bit and then execute a \texttt{RESET} instruction. When the bootloader returns control to the main application, the \texttt{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 \texttt{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 module which can perform the above procedure upon detection of a TFTP packet. When the Reboot 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® 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® microcontroller Code Protect feature turned on.
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**: 192.168.97.60
- **MAC Address**: 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** 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.

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.

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 were 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 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.

   ```shell
tftp mchpboard put "C:\Microchip Solutions\TCP/IP Demo App\TCP/IP Demo App-C18.hex"
   
   If the bootload 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 bootload 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.
WebVend

The TCP/IP 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)
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 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 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.
This is the configuration page for the streaming MP3 Internet Radio board that you have. It was designed to interface to Shoutcast servers, but several different HTTP based MP3 servers may also be used.

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.

**Genre:** World
**Station:** #MUSIK.MAIN - WWW.RAUTEMUSIK.FM - 24HR 40 POP HITS 80S 90S DANCE HOUSE ROCK RB AND MORE!
**URL:** http://scfre-dtc-aa07.stream.aol.com:80/stream/1051
**Playing:** Track Update on www.RauteMusik.FM

**Configurations:**
**TCP/IP Stack Version:** v5.00
**Build Date:** Apr 23 2009 23:48:04
**MAC Address:** 00:04:A3:12:34:56

---

**Demo Information** > **Available Demos** > **Internet Radio**

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] | [Index] | [Home]
WiFi Console

The TCPIP WiFi Console Demo App (previously the TCPIP WiFi Iperf Demo App) is a command line interface (CLI) to the MRF24WB0M / MRF24WG0M. 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.

The following network types are supported

- CFG_WF_INFRASTRUCTURE (as a client in infrastructure network)
- CFG_WF_ADHOC
- CFG_WF_P2P (Wi-Fi Direct) (as a group client in Wi-Fi Direct network)

CFG_WF_P2P is only available for MRF24WG0MA/B.

New security modes are supported.

- WF_SECURITY_WPS_PUSH_BUTTON (supported by MRF24WG0M only)
- WF_SECURITY_WPS_PIN (supported by MRF24WG0M only)

Wireless configurations are set up in WF_Config.h

New demo feature: Cloud Networking Application

From MLA v5.42.06 March 2013 and future releases, a new demo is added to showcase cloud networking. The relevant source file is
CloudTCPClient.c. Reference web server is http://www.openwificloud.com

This cloud demo supports the functions

- Temperature reading
- Potentiometer reading
- LEDs ON/OFF toggling

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standalone Commands</strong></td>
<td>Standalone CLI commands for talking with the MRF24WB0M / MRF24WG0M.</td>
</tr>
<tr>
<td><strong>iwconfig Commands</strong></td>
<td>Commands for controlling the wireless network interface.</td>
</tr>
<tr>
<td><strong>ifconfig Commands</strong></td>
<td>Commands for controlling the network interface.</td>
</tr>
<tr>
<td><strong>iwpriv Commands</strong></td>
<td>Commands for controlling the wireless encryption settings.</td>
</tr>
<tr>
<td><strong>iperf Example</strong></td>
<td>An example of using iperf to measure network bandwidth and performance.</td>
</tr>
</tbody>
</table>

Demo Information > Available Demos > WiFi Console
# Standalone Commands

These command line interface (CLI) commands are not related to the wireless or networking interface directly.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>help</td>
<td>Lists all the available CLI commands for the MRF24WB0M / MRF24WG0M.</td>
</tr>
<tr>
<td>getwfver</td>
<td>Lists the MRF24W firmware version and host driver version numbers.</td>
</tr>
<tr>
<td>reset</td>
<td>Issues a host reset.</td>
</tr>
<tr>
<td>cls</td>
<td>Resets the prompt.</td>
</tr>
<tr>
<td>iperf</td>
<td>Initiates an iperf session. See the section on iperf for more information.</td>
</tr>
<tr>
<td>kill iperf</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>ping</td>
<td>Initiates a ping session. This will verify IP-level connectivity to another TCP/IP terminal by sending Internet Control Message Protocol (ICMP) Echo Request messages. The receipt of corresponding Echo Reply messages are displayed, along with round-trip times.</td>
</tr>
<tr>
<td>killping</td>
<td>Stops a running ping session.</td>
</tr>
<tr>
<td>wpscred</td>
<td>Display WPS credentials.</td>
</tr>
</tbody>
</table>

```
> ifconfig
  IP addr: 192.168.1.146
  MAC addr: 00:1E:C0:00:00:01
  Netmask: 255.255.255.0
  Gateway: 192.168.1.1
  DHCP: Started

> ping 192.168.1.1
  Reply From 192.168.1.1: time=5ms
  Reply From 192.168.1.1: time=10ms
  Reply From 192.168.1.1: time=8ms
  Reply From 192.168.1.1: time=10ms
  Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss)
    Approximate round trip times in milli-seconds:
      Minimum = 5ms, Maximum = 10ms, Average = 8ms

> wpscred
  SSID: A0_MicrochipDemoAP'
  Net Key:
    6D 69 63 72 6F 63 68 69 70 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    Auth Type: 8192
    Fnc Type: 2048
    Net ID: 1
    Key ID: 1
    BSSID: C0 C1 C0 27 5A A0
```
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> ]
[ scan ]
[ hibernate ]
[ wakeup ]
```

Note: iwconfig with no options will display wireless status.

<table>
<thead>
<tr>
<th>ssid</th>
<th>1-32 ASCII characters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Currently doesn't accept spaces in the SSID name.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mode</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>Forces the MRF24WB0M / MRF24WG0M to disconnect from any currently connected network (adhoc or infrastructure).</td>
</tr>
<tr>
<td>managed</td>
<td>The MRF24WB0M / MRF24WG0M will connect to the SSID in infrastructure mode. Note that all the network parameters must be correct before this command is called.</td>
</tr>
<tr>
<td>adhoc</td>
<td>The MRF24WB0M / MRF24WG0M will connect to the SSID in adhoc mode. Note that all the network parameters must be correct before this command is called.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>channel</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>channel list</td>
<td>A comma separated list of all the channels to scan.</td>
</tr>
</tbody>
</table>

Sets the MRF24WB0M / MRF24WG0M to scan all channels in the
<table>
<thead>
<tr>
<th>all</th>
<th>given regulatory domain.</th>
</tr>
</thead>
<tbody>
<tr>
<td>power</td>
<td>Enables all power saving features (PS_POLL) of the MRF24WB0M / MRF24WG0M. MRF24WB0M / MRF24WG0M will wake up to check for all types of traffic (unicast, multicast, and broadcast).</td>
</tr>
<tr>
<td>disable</td>
<td>Disables any power savings features. The MRF24WB0M / MRF24WG0M will always be in an active power state.</td>
</tr>
<tr>
<td>unicast</td>
<td>The MRF24WB0M / MRF24WG0M will be in it's deepest sleep state, only waking up at periodic intervals to check for unicast data. The MRF24WB0M / MRF24WG0M will not wake up on the DTIM period for broadcast or multicast traffic.</td>
</tr>
<tr>
<td>domain</td>
<td></td>
</tr>
<tr>
<td>fcc</td>
<td>United States channels 1-11.</td>
</tr>
<tr>
<td>ic</td>
<td>Canada channels 1-11. Applicable for MRF24WB0M only.</td>
</tr>
<tr>
<td>etsi</td>
<td>European channels 1-13.</td>
</tr>
<tr>
<td>spain</td>
<td>Spanish channels 10-11. Applicable for MRF24WB0M only.</td>
</tr>
<tr>
<td>france</td>
<td>French channels 10-13. Applicable for MRF24WB0M only.</td>
</tr>
<tr>
<td>japan</td>
<td>Japanese channel 1-14. Applicable for MRF24WG0M only.</td>
</tr>
<tr>
<td>japana</td>
<td>Japanese channel 14. Applicable for MRF24WB0M only.</td>
</tr>
<tr>
<td>japanb</td>
<td>Japanese channels 1-11. Applicable for MRF24WB0M only.</td>
</tr>
<tr>
<td>rts</td>
<td>Set the requested number of bytes to send. Default max is 2347.</td>
</tr>
<tr>
<td>scan</td>
<td>Instructs the MRF24WB0M / MRF24WG0M to perform an active site scan. Scan results will be displayed to the output terminal.</td>
</tr>
<tr>
<td>hibernate</td>
<td>Turns off LDO of the MRF24W module, which is equivalent to removing power to the MRF24WB0M / MRF24WG0M. Has the same effect of</td>
</tr>
</tbody>
</table>
resetting MRF24WB0M / MRF24WG0M. MRF24W state is not maintained when transitioning to hibernate mode.

<table>
<thead>
<tr>
<th>wakeup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restores power to the MRF24WB0M / MRF24WG0M and reconnects.</td>
</tr>
</tbody>
</table>

Note: scan is only supported by the WiFi EZConfig demo.
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 (ifconfig auto-dhcp drop) 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 / MRF24WG0M is in idle mode. Doing so at other times can have unexpected results.

| netmask     | Use the specified IP address for the netmask. The netmask value is specified in dot-decimal notation.

| gateway     | 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. |
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:

```
[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.

<table>
<thead>
<tr>
<th>enc</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>The MRF24WB0M / MRF24WG0M will not use any encryption to connect to the specified network.</td>
</tr>
<tr>
<td>wep</td>
<td>The MRF24WB0M / MRF24WG0M will use either WEP-40 (short) or WEP-104 (long) encryption to connect to the specified network.</td>
</tr>
<tr>
<td>wpa-psk</td>
<td>The MRF24WB0M / MRF24WG0M will use the specified 32-byte PSK to connect to the WPA/WPA2 network.</td>
</tr>
<tr>
<td>wpa-phrase</td>
<td>The MRF24WB0M / MRF24WG0M will take the given 1-32 ASCII character passphrare, 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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Instructs the MRF24WB0M / MRF24WG0M 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.</td>
</tr>
<tr>
<td>[2]</td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td>If value is specified, this will instruct the MRF24WB0M / MRF24WG0M 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</td>
</tr>
<tr>
<td>text</td>
<td>value</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>Value characters of 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.</td>
<td></td>
</tr>
<tr>
<td>PSK value</td>
<td>32-byte hex value for the PSK. This value can be calculated from the following website hosted on the Wireshark website.</td>
</tr>
<tr>
<td>Phrase value</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iperf [-s</td>
<td>c &lt;IP addr&gt;] [-u] [-i &lt;sec&gt;] [-b &lt;bandwidth&gt;] [-t &lt;time&gt;]</td>
</tr>
<tr>
<td>-s</td>
<td>Runs the iperf client. The IP address is the IP address of the server.</td>
</tr>
<tr>
<td>-c &lt;IP addr&gt;</td>
<td>Server side only. Sends UDP datagrams.</td>
</tr>
<tr>
<td>-u</td>
<td>Specified the time interval, in seconds, that the display will be updated.</td>
</tr>
<tr>
<td>-i &lt;sec&gt;</td>
<td>Specifies the amount of data to try and send. This option is only valid with UDP datagrams.</td>
</tr>
<tr>
<td>-b &lt;bandwidth&gt;</td>
<td></td>
</tr>
</tbody>
</table>
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 / MRF24WG0M receiver performance. If you start the iperf server on a PC, then you will be measuring MRF24WB0M / MRF24WG0M transmit performance. Below are two images that show receiver and transmitter performance, respectively.
Demo Information > Available Demos > WiFi Console > iperf Example

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
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 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 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 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 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.

The following network types are supported

- CFG_WF_ADHOC
- CFG_WF_SOFT_AP

Wireless configurations are set up in WF_Config.h

**Adhoc Networks**

Upon starting the demo, the network will either connect to another adhoc network, or will start it's 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 / MRF24WG0M 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.
SoftAP Networks

This is only available for MRF24WG0MA/B. Upon starting the demo, the MRF24WG0MA/B will start up a network as a software-enabled access point (AP), acting as the centralized coordinator for the network. Devices can **connect** to the MRF24WG0MA/B softAP. Depending on the RF module firmware version, either 1 or 4 clients can be connected to the softAP. Routing is not supported. The demo can start a softAP network with no or WEP security.

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`.

<table>
<thead>
<tr>
<th>WLAN Type</th>
<th>Either adhoc or infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSID</td>
<td>Name of network (1-32 ASCII characters)</td>
</tr>
<tr>
<td>Security Type</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• None</td>
</tr>
<tr>
<td></td>
<td>• WEP-40 (5 ASCII characters or 10 hex numbers)</td>
</tr>
<tr>
<td></td>
<td>• WEP-104 (13 ASCII characters or 26 hex numbers)</td>
</tr>
<tr>
<td></td>
<td>• WPA/WPA2 passphrase (8-63 ASCII characters)</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Define</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STACK_USE_EZ_CONFIG</td>
<td>The top level define to enable EasyConfig features.</td>
</tr>
<tr>
<td>EZ_CONFIG_SCAN</td>
<td>Adds additional ability to instruct the MRF24WB0M / MRF24WG0M module to scan for available networks nearby. This can be done when you are already connected to a network.</td>
</tr>
<tr>
<td>EZ_CONFIG_STORE</td>
<td>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.</td>
</tr>
<tr>
<td>EZ_CONFIG_STALL</td>
<td>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 to a new network.</td>
</tr>
</tbody>
</table>

**EZ_CONFIG_SCAN**

The MRF24WB0M / MRF24WG0M 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 / MRF24WG0M 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 / MRF24WG0M 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.

**New demo feature: Mobile Application**

From MLA v5.42.06 March 2013 and future releases, a new mobile application demo is added to Explorer16+PIC32 configuration. The relevant source file is MobileTCP_Server.c. Supported devices run on Android (version 4.0 or later) and iOS6 or later. On Goggle Play store and Apple Apps Store, search for the app "OpenWifiFog".

The functions supported

- Temperature reading
- Potentiometer reading
- LEDs ON/OFF toggling
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.
Google, Inc. has deprecated Google PowerMeter and expressed its intent to remove access to it on September 16, 2011.

Because of this development, Microchip Technology has removed the Google PowerMeter demo projects from the Microchip Application Libraries distribution. To obtain Microchip's Google PowerMeter reference implementation, please download the archived Microchip Application Libraries installation from June, 2011 from www.microchip.com/mla.
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.

Google, Inc. has deprecated Google PowerMeter and expressed its intent to remove access to it on September 16, 2011.

Because of this development, Microchip Technology has removed the Google PowerMeter demo projects from the Microchip Application Libraries distribution. To obtain Microchip's Google PowerMeter reference implementation, please download the archived Microchip Application Libraries installation from June, 2011 from www.microchip.com/mla.

This Energy Monitoring demo has been modified to remove its Google PowerMeter upload features. However, the energy measurements made by the PICtail can still be viewed on the demo board-hosted web page.

The PC GUI for the Energy Monitoring PICtail can be obtained from: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en550456
WiFi G Demo

The Wi-Fi G Demo Board utilizes the EasyConfig mechanism in TCPIP WiFi EZConfig demo app, which allows configuration of an embedded device on a wireless network. It utilizes the web server of the TCP/IP stack to allow the user to input the desired network information from a client browser, and then reset the device to connect to the desired network. By default, Wi-Fi G Demo Board will start up in SoftAP network mode.

Refer to the documentation Wi-Fi G Demo Board User's Guide for more information.
Using the Stack

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

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack Architecture</td>
<td>Describes the TCP/IP stack architecture.</td>
</tr>
<tr>
<td>How the Stack Works</td>
<td>Describes how the stack works and how to start creating your application.</td>
</tr>
</tbody>
</table>
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 used with the stack.
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.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required Files</td>
<td>Describes the base files needed in a TCP/IP project.</td>
</tr>
<tr>
<td>APP_CONFIG Structure</td>
<td>Describes the APP_CONFIG structure.</td>
</tr>
<tr>
<td>Main File</td>
<td>Describes how to construct the main() loop in your application.</td>
</tr>
<tr>
<td>Cooperative Multitasking</td>
<td>Describes how to implement a non-blocking stack task.</td>
</tr>
<tr>
<td>RTOS</td>
<td>Contains information about using the TCP/IP Stack in a Real-Time Operating System.</td>
</tr>
</tbody>
</table>

Using the Stack > How the Stack Works
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** - 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 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 topic in the Protocol.
Configuration topic.

Using the Stack > How the Stack Works > Required Files

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
APP_CONFIG Structure

Most of the stack-related application variables are stored in an APP_CONFIG structure. These include addresses, 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_BYTE[n] macros in TCPIPConfig.h. The Init AppConfig 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.
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.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>Describes some basic initialization calls that should be made.</td>
</tr>
<tr>
<td>Main Loop</td>
<td>Describes how to construct the main loop of your application.</td>
</tr>
</tbody>
</table>

Using the Stack > How the Stack Works > Main File
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()` 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()` function will need to initialize an SPI port to communicate to a memory storage 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` 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()` for the TCP protocol, `HTTPInit()` for HTTP2,...). After `StackInit()` has been called, you can call other application-specific firmware initialization functions.
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 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.
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.

```c
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 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 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.

```c
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** defines the number of ticks in a second,
**TICK_MINUTE** defines the number of ticks in a minute, and **TICK_HOUR** 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.
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.
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.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Configuration</td>
<td>Describes how to configure the hardware options for your project.</td>
</tr>
<tr>
<td>Address</td>
<td>Describes how to set up addressing for your device.</td>
</tr>
<tr>
<td>Protocol Configuration</td>
<td>Describes how to set up protocols for your application.</td>
</tr>
</tbody>
</table>

Configuring the Stack
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.

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.

### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clock Frequency</strong></td>
<td>Describes how to configure the clock for your application.</td>
</tr>
<tr>
<td><strong>External Storage</strong></td>
<td>Describes how to configure external storage options to hold web pages or structures in your application.</td>
</tr>
<tr>
<td><strong>ENC28J60 Config</strong></td>
<td>Describes how to configure your project to use an ENC28J60.</td>
</tr>
<tr>
<td><strong>ENCX24J600 Config</strong></td>
<td>Describes how to configure your project to use an ENCX24J600.</td>
</tr>
<tr>
<td><strong>PIC18F97J60 Config</strong></td>
<td>Describes how to configure your project to use a PIC18F97J60.</td>
</tr>
<tr>
<td><strong>PIC32MX7XX Config</strong></td>
<td>Describes how to configure your project to use a PIC32MX795 family part.</td>
</tr>
</tbody>
</table>

[Configuring the Stack > Hardware Configuration]
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() xxxxxxxxxxxxxxx
```

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 defines as `(GetSystemClock() / 2)` for PIC24F processors).
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 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 (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.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Purpose</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxxx_CS_IO</td>
<td>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.</td>
<td>LATDbits.LATD12</td>
</tr>
<tr>
<td>xxxxx_CS_TRIS</td>
<td>Defines the TRIS bit that corresponds to the chip select pin on the device.</td>
<td>TRISDbits.TRISD12</td>
</tr>
<tr>
<td>xxxxx_SCK_TRIS</td>
<td>Defines the TRIS bit that corresponds to the clock pin of the SPI module connected to the device.</td>
<td>TRISGbits.TRISG6</td>
</tr>
<tr>
<td>xxxxx_SDI_TRIS</td>
<td>Defines the TRIS bit that corresponds to the data-in pin of the SPI module connected to the device.</td>
<td>TRISGbits.TRISG7</td>
</tr>
<tr>
<td>xxxxx_SDO_TRIS</td>
<td>Defines the TRIS bit that corresponds to the data-out pin of the SPI module connected to the device.</td>
<td>TRISGbits.TRISG8</td>
</tr>
<tr>
<td>xxxxx_SPI_IF</td>
<td>Points to the interrupt flag for the SPI module connected to the device.</td>
<td>IFS2bits.SPI2IF</td>
</tr>
<tr>
<td>xxxxx_SSPBUF</td>
<td>Points to the SPI buffer register for the SPI module connected to the device.</td>
<td>SPI2BUF</td>
</tr>
<tr>
<td>xxxxx_SPICON1</td>
<td>Points to the SPI control register for the SPI module connected to the device.</td>
<td>SPI2CON1</td>
</tr>
<tr>
<td>xxxxx_SPICON1bits</td>
<td>Provides bitwise access to the SPI control register for the SPI module connected to the device. The ____bits registers are</td>
<td>SPI2CON1bits</td>
</tr>
<tr>
<td>xxxxxx_SPICON2</td>
<td>typically defined in the processor's header files.</td>
<td></td>
</tr>
<tr>
<td>xxxxxx_SPISTAT</td>
<td>Points to the second SPI control register for the SPI module connected to the device. If your device doesn't have an SPICON2 register (e.g. PIC32) just omit this definition.</td>
<td></td>
</tr>
<tr>
<td>xxxxxx_SPISTATbits</td>
<td>Points to the SPI status register for the SPI module connected to the device.</td>
<td></td>
</tr>
<tr>
<td>xxxxxx_SPIBRG</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Purpose</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENC_CS_IO</td>
<td>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.</td>
<td>LATDbits.LATD14</td>
</tr>
<tr>
<td>ENC_CS_TRIS</td>
<td>Defines the TRIS bit that corresponds to the chip select pin.</td>
<td>TRISDbits.TRISD14</td>
</tr>
<tr>
<td>ENC_RST_IO</td>
<td>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.</td>
<td>LATDbits.LATD15</td>
</tr>
<tr>
<td>ENC_RST_TRIS</td>
<td>Defines the TRIS bit that corresponds to the reset pin.</td>
<td>TRISDbits.TRISD15</td>
</tr>
<tr>
<td>ENC_SPI_IF</td>
<td>Points to the interrupt flag for the SPI module connected to the chip.</td>
<td>IFS0bits.SPI1IF</td>
</tr>
<tr>
<td>ENC_SSPBUF</td>
<td>Points to the SPI buffer register for the SPI module connected to the chip.</td>
<td>SPI1BUF</td>
</tr>
<tr>
<td>ENC_SPISTAT</td>
<td>Points to the SPI status register for the SPI module connected to the chip.</td>
<td>SPI1STAT</td>
</tr>
<tr>
<td>ENC_SPISTATbits</td>
<td>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.</td>
<td>SPI1STATbits</td>
</tr>
<tr>
<td>ENC_SPICON1</td>
<td>Points to the SPI control register for the SPI module connected to the chip.</td>
<td>SPI1CON1</td>
</tr>
<tr>
<td>ENC_SPICON1bits</td>
<td>Provides bitwise access to the SPI control register for the SPI module connected to the</td>
<td>SPI1CON1bits</td>
</tr>
<tr>
<td>ENC_SPICON2</td>
<td>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.</td>
<td>SPI1CON2</td>
</tr>
<tr>
<td>ENC_SPIBRG</td>
<td>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.</td>
<td>SPI1BRG</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SPI mode using CS, SCK, SI, and SO pins</td>
</tr>
<tr>
<td>1</td>
<td>8-bit demultiplexed PSP Mode 1 with RD and WR pins</td>
</tr>
<tr>
<td>2</td>
<td>8-bit demultiplexed PSP Mode 2 with R/~W and EN pins</td>
</tr>
<tr>
<td>3</td>
<td>16-bit demultiplexed PSP Mode 3 with Rd, WRL, and WRH pins</td>
</tr>
<tr>
<td>4</td>
<td>16-bit demultiplexed PSP Mode 4 with R/~W, B0SEL, and B1SEL pins</td>
</tr>
<tr>
<td>5</td>
<td>8-bit multiplexed PSP Mode 5 with RD and WR pins</td>
</tr>
<tr>
<td>6</td>
<td>8-bit multiplexed PSP Mode 6 with R/~W and EN pins</td>
</tr>
<tr>
<td>9</td>
<td>16-bit multiplexed PSP Mode 9 with AL, RD, WRL, and WRH pins</td>
</tr>
<tr>
<td>10</td>
<td>16-bit multiplexed PSP Mode 10 with AL, R/~W, B0SEL, and B1SEL pins</td>
</tr>
</tbody>
</table>

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:
<table>
<thead>
<tr>
<th>Macro</th>
<th>Purpose</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENC100_INTERFACE_MODE</td>
<td>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.</td>
<td>0</td>
</tr>
<tr>
<td>ENC100_PSP_USE_INDIRECT_RAM_ADDRESSING</td>
<td>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.</td>
<td>N/A</td>
</tr>
<tr>
<td>ENC100_TRANSLATE_TO_PIN_ADDR(a)</td>
<td>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).</td>
<td>(((a)&amp;0x100)&lt;&lt;6)</td>
</tr>
<tr>
<td>ENC100_MDIX_IO</td>
<td>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.</td>
<td>LATBbits.LATB</td>
</tr>
<tr>
<td>ENC100_MDIX_TRIS</td>
<td>Defines the TRIS bit to use for the Auto-crossover circuit.</td>
<td>TRISBbits.TRI...</td>
</tr>
<tr>
<td>ENC100_INT_IO</td>
<td>Defines an I/O pin to use for the chip's interrupt signal pin. This feature is currently unused by the stack.</td>
<td>PORTAbits.RA</td>
</tr>
<tr>
<td>ENC100_INT_TRIS</td>
<td>Defines a TRIS bit to use for the chip's interrupt signal pin.</td>
<td>TRISAbits.TRI...</td>
</tr>
<tr>
<td>ENC100_CS_IO</td>
<td>Defines a port bit for use with the chip select pin. Optional in PSP modes.</td>
<td>LATAbits.LATA</td>
</tr>
<tr>
<td>ENC100_CS_TRIS</td>
<td>Defines a TRIS bit to use for the chip select pin.</td>
<td>TRISAbits.TRI...</td>
</tr>
<tr>
<td>ENC100_POR_IO</td>
<td>Defines the port bit to use with a power disconnect circuit. If your application doesn't have this feature implemented, comment out this bit.</td>
<td>LATCbites.LATC...</td>
</tr>
<tr>
<td>ENC100_POR_TRIS</td>
<td>Defines the TRIS bit to use with a power disconnect circuit.</td>
<td>TRISCbits.TRISD4</td>
</tr>
<tr>
<td>ENC100_SO_WR_B0SEL_EN_IO</td>
<td>Defines a pin used for communication. The functionality of this pin depends on which communication mode in selected. It can be equivalent to the ENCX24J600 serial out pin, the parallel mode WR strobe, the B0SEL pin, or the EN pin.</td>
<td>LATDbits.LATD4</td>
</tr>
<tr>
<td>ENC100_SO_WR_B0SEL_EN_TRIS</td>
<td>Defines the TRIS bit to use with the ENC100_SO_WR_B0SEL_EN_IO pin.</td>
<td>TRISDbits.TRISD4</td>
</tr>
<tr>
<td>ENC100_SI_RD_RW_IO</td>
<td>Defines a pin used for communication. The functionality of this pin depends on which communication mode in selected. It can be equivalent to the ENCX24J600 serial in pin, the parallel mode RD strobe, or the R/~W pin.</td>
<td>LATDbits.LATD5</td>
</tr>
<tr>
<td>ENC100_SI_RDWR_RW_TRIS</td>
<td>Defines the TRIS bit to use with the ENC100_SI_RDWR_RW_IO pin.</td>
<td>TRISDbits.TRISD5</td>
</tr>
<tr>
<td>ENC100_SCK_AL_IO</td>
<td>Defines a pin used for communication. The functionality of this pin depends on which communication mode in selected. It can be equivalent to the ENCX24J600 serial clock pin or the parallel mode address latch strobe.</td>
<td>LATDbits.LATD15</td>
</tr>
<tr>
<td>ENC100_SCK_AL_TRIS</td>
<td>Defines the TRIS bit to use with the ENC100_SCK_AL_IO pin.</td>
<td>TRISDbits.TRISD15</td>
</tr>
<tr>
<td>ENC100_ISR_ENABLE</td>
<td>Points to the bit to enable the interrupt for the I/O based ENCX24J600-triggered interrupt. This feature is not currently implemented.</td>
<td>IEC1bits.INT2IE</td>
</tr>
<tr>
<td>ENC100_ISR_FLAG</td>
<td>Points to the interrupt flag bit for the I/O based ENCX24J600-triggered interrupt. This feature is not currently implemented.</td>
<td>IFS1bits.INT2IF</td>
</tr>
<tr>
<td>ENC100_ISR_POLARITY</td>
<td>Points to the interrupt polarity bit</td>
<td></td>
</tr>
<tr>
<td>ENC100_ISR_POLARITY</td>
<td>for the I/O based ENCX24J600-triggered interrupt. This feature is not currently implemented.</td>
<td></td>
</tr>
<tr>
<td>ENC100_ISR_PRIORITY</td>
<td>Points to the interrupt priority bit for the I/O based ENCX24J600-triggered interrupt. This feature is not currently implemented.</td>
<td></td>
</tr>
<tr>
<td>ENC100_SPI_ENABLE</td>
<td>Points to the SPI module enable bit if SPI mode is used.</td>
<td></td>
</tr>
<tr>
<td>ENC100_SPI_IF</td>
<td>Points to the interrupt flag for the SPI module if SPI mode is used.</td>
<td></td>
</tr>
<tr>
<td>ENC100_SSPBUF</td>
<td>Points to the SPI buffer register for the SPI module if SPI mode is used.</td>
<td></td>
</tr>
<tr>
<td>ENC100_SPISTAT</td>
<td>Points to the SPI status register for the SPI module if SPI mode is used.</td>
<td></td>
</tr>
<tr>
<td>ENC100_SPISTATbits</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>ENC100_SPICON1</td>
<td>Points to the SPI control register for the SPI module if SPI mode is used.</td>
<td></td>
</tr>
<tr>
<td>ENC100_SPICON1bits</td>
<td>Provides bitwise access to the SPI control register for the SPI module if SPI mode is used (see ENC_SPISTATbits entry).</td>
<td></td>
</tr>
<tr>
<td>ENC100_SPICON2</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>ENC100_SPIBRG</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>
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.
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`:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Purpose</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_RMII</td>
<td>Define this macro if the external PHY runs in RMII mode. Comment it out if you're using an MII PHY.</td>
<td>-</td>
</tr>
<tr>
<td>PHY_CONFIG_ALTERNATE</td>
<td>Define this symbol if the PIC32MX7XX uses the alternate configuration pins to connect to the PHY. Comment it out for the default configuration pins.</td>
<td>-</td>
</tr>
<tr>
<td>PHY_ADDRESS</td>
<td>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).</td>
<td>0x1</td>
</tr>
</tbody>
</table>

Update the following definitions in `TCPIPConfig.h`:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Purpose</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETH_CFG_LINK</td>
<td>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.</td>
<td>0</td>
</tr>
<tr>
<td>ETH_CFG_AUTO</td>
<td>Set to 1 to use auto negotiation. Strongly recommended.</td>
<td>1</td>
</tr>
<tr>
<td>ETH_CFG_10</td>
<td>Use/advertise 10 Mbps capability.</td>
<td>1</td>
</tr>
<tr>
<td>ETH_CFG_100</td>
<td>Use/advertise 100 Mbps capability.</td>
<td>1</td>
</tr>
<tr>
<td>ETH_CFG_HDUPLEX</td>
<td>Use/advertise half duplex capability.</td>
<td>1</td>
</tr>
<tr>
<td>Configuration</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>ETH_CFG_FDUPLEX</td>
<td>Use/advertise full duplex capability.</td>
<td></td>
</tr>
<tr>
<td>ETH_CFG_AUTO_MDIX</td>
<td>Use/advertise auto MDIX capability (effective only when ETH_CFG_AUTO is enabled).</td>
<td></td>
</tr>
<tr>
<td>ETH_CFG_SWAP_MDIX</td>
<td>Use swapped MDIX if defined. Otherwise, use normal MDIX.</td>
<td></td>
</tr>
</tbody>
</table>
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.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC Address</td>
<td>Describes how to configure the MAC address on your device.</td>
</tr>
<tr>
<td>IP Address</td>
<td>Describes how to configure your application's IP address.</td>
</tr>
</tbody>
</table>

Configuring the Stack > Address
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:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MY_DEFAULT_MAC_BYTE1</td>
<td>(0x00)</td>
</tr>
<tr>
<td>MY_DEFAULT_MAC_BYTE2</td>
<td>(0x04)</td>
</tr>
<tr>
<td>MY_DEFAULT_MAC_BYTE3</td>
<td>(0xA3)</td>
</tr>
<tr>
<td>MY_DEFAULT_MAC_BYTE4</td>
<td>(0x00)</td>
</tr>
<tr>
<td>MY_DEFAULT_MAC_BYTE5</td>
<td>(0x00)</td>
</tr>
<tr>
<td>MY_DEFAULT_MAC_BYTE6</td>
<td>(0x00)</td>
</tr>
</tbody>
</table>

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, MRF24WG0M 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 initialization code so it will obtain the device's MAC address from one of these EEPROMs instead of the default MAC address macros.
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:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Property</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MY_DEFAULT_IP_ADDR_BYTE1</td>
<td>Default IP address byte 1</td>
<td>192ul</td>
</tr>
<tr>
<td>MY_DEFAULT_IP_ADDR_BYTE2</td>
<td>Default IP address byte 2</td>
<td>168ul</td>
</tr>
<tr>
<td>MY_DEFAULT_IP_ADDR_BYTE3</td>
<td>Default IP address byte 3</td>
<td>1ul</td>
</tr>
<tr>
<td>MY_DEFAULT_IP_ADDR_BYTE4</td>
<td>Default IP address byte 4</td>
<td>100ul</td>
</tr>
<tr>
<td>MY_DEFAULT_MASK_BYTE1</td>
<td>Default subnet mask byte 1</td>
<td>255ul</td>
</tr>
<tr>
<td>MY_DEFAULT_MASK_BYTE2</td>
<td>Default subnet mask byte 2</td>
<td>255ul</td>
</tr>
<tr>
<td>MY_DEFAULT_MASK_BYTE3</td>
<td>Default subnet mask byte 3</td>
<td>255ul</td>
</tr>
<tr>
<td>MY_DEFAULT_MASK_BYTE4</td>
<td>Default subnet mask byte 4</td>
<td>0ul</td>
</tr>
<tr>
<td>MY_DEFAULT_GATE_BYTE1</td>
<td>Default gateway byte 1</td>
<td>192ul</td>
</tr>
<tr>
<td>MY_DEFAULT_GATE_BYTE2</td>
<td>Default gateway byte 2</td>
<td>168ul</td>
</tr>
<tr>
<td>MY_DEFAULT_GATE_BYTE3</td>
<td>Default gateway byte 3</td>
<td>1ul</td>
</tr>
<tr>
<td>MY_DEFAULT_GATE_BYTE4</td>
<td>Default gateway byte 4</td>
<td>1ul</td>
</tr>
</tbody>
</table>

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 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
```

in 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
```
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 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.
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.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol Macros and Files</td>
<td>Describes macros and files that must be included to use protocols.</td>
</tr>
<tr>
<td>Additional Features</td>
<td>Describes additional firmware demos and functionality.</td>
</tr>
<tr>
<td>Sockets</td>
<td>Describes how to set up sockets for your application.</td>
</tr>
</tbody>
</table>

Configuring the Stack > Protocol Configuration

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
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:

<table>
<thead>
<tr>
<th>Module</th>
<th>Macro</th>
<th>Function</th>
<th>Required Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMP</td>
<td>STACK_USE_ICMP_SERVER</td>
<td>Provides the ability to query and respond to pings.</td>
<td>ICMP.c, ICMP.h</td>
</tr>
<tr>
<td>ICMP</td>
<td>STACK_USE_ICMP_CLIENT</td>
<td>Provides the ability to transmit pings.</td>
<td>ICMP.c, ICMP.h</td>
</tr>
<tr>
<td>HTTP2</td>
<td>STACK_USE_HTTP2_SERVER</td>
<td>Provides HTTP server functionality with dynamic variables, POST, Cookies, Authentication, and other features</td>
<td>HTTP2.c, HTTP2.h, TCP.c, TCP.h, CustomHTTPApp.c and HTTPPrint.h (see <a href="#">HTTP2 section</a> for information on these files)</td>
</tr>
<tr>
<td>SSL</td>
<td>STACK_USE_SSL_SERVER</td>
<td>Provides support for SSL server sockets.</td>
<td>SSL.c, SSL.h, ARCFOUR.c, ARCFOUR.h, BigInt.c, BigInt.h, Random.c, Random.h, RSA.h</td>
</tr>
<tr>
<td>SSL</td>
<td>STACK_USE_SSL_CLIENT</td>
<td>Provides support for SSL client sockets.</td>
<td>SSL.c, SSL.h, ARCFOUR.c, ARCFOUR.h, BigInt.c, BigInt.h, Random.c, Random.h, RSA.h</td>
</tr>
<tr>
<td>FTP</td>
<td>STACK_USE_FTP_SERVER</td>
<td>Provides ability to remotely upload MPFS2 images to HTTP2 servers via FTP</td>
<td>FTP.c, FTP.h, TCP.c, TCP.h</td>
</tr>
</tbody>
</table>

Provides the...
<table>
<thead>
<tr>
<th>Protocol</th>
<th>Use Stack</th>
<th>Description</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMTP</td>
<td>STACK_USE_SMTP_CLIENT</td>
<td>ability to send email</td>
<td>Helpers.c, Helpers.h</td>
</tr>
<tr>
<td>SNMP</td>
<td>STACK_USE_SNMP_SERVER</td>
<td>Provides a network-based machine control/monitoring protocol</td>
<td>SNMP.c, SNMP.h, UDP.c, UDP.h</td>
</tr>
<tr>
<td>TFTP</td>
<td>STACK_USE_TFTP_CLIENT</td>
<td>Provides unreliable file upload/download services</td>
<td>TFTP.c.c, TFTP.h, TCP.c, TCP.h</td>
</tr>
<tr>
<td>Telnet</td>
<td>STACK_USE_TELNET_SERVER</td>
<td>Provides telnet services.</td>
<td>Telnet.c, Telnet.h, TCP.c, TCP.h</td>
</tr>
<tr>
<td>Announce</td>
<td>STACK_USE_ANNOUNCE</td>
<td>Provides device hostname and IP address discovery on a local Ethernet subnet</td>
<td>Announce.c, Announce.h</td>
</tr>
<tr>
<td>DNS</td>
<td>STACK_USE_DNS</td>
<td>Provides the ability to resolve hostnames to IP addresses</td>
<td>DNS.c, DNS.h, UDP.c, UDP.h</td>
</tr>
<tr>
<td>NBNS</td>
<td>STACK_USE_NBNS</td>
<td>Provides the ability to resolve hostnames to IP addresses on the same subnet.</td>
<td>NBNS.c, NBNS.h, UDP.c, UDP.h</td>
</tr>
<tr>
<td>SNTP</td>
<td>STACK_USE_SNTP_CLIENT</td>
<td>Provides the ability to get the date/time from the internet</td>
<td>SNTP.c, SNTP.h, UDP.c, UDP.h</td>
</tr>
<tr>
<td>Dynamic DNS</td>
<td>STACK_USE_DYNAMICDNS_CLIENT</td>
<td>Provides the ability to resolve hostnames to IP addresses that change frequently.</td>
<td>DynDNS.c, DynDNS.h, TCP.h</td>
</tr>
<tr>
<td>MPFS2</td>
<td>STACK_USE_MPFS2</td>
<td>Provides MPFS2 services for custom applications. This functionality will be</td>
<td>MPFS2.c, MPFS2.h</td>
</tr>
<tr>
<td>Protocol</td>
<td>Macro</td>
<td>Functionality</td>
<td>Source Files</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>---------------</td>
<td>--------------</td>
</tr>
<tr>
<td>TCP</td>
<td>STACK_USE_TCP</td>
<td>enabled/required automatically by stack-based protocols that require MPFS2.</td>
<td>TCP.c, TCP.h</td>
</tr>
<tr>
<td>UDP</td>
<td>STACK_USE_UDP</td>
<td>Provides UDP transport layer services for custom protocols. This functionality is automatically enabled/required by stack-based protocols that require UDP sockets.</td>
<td>UDP.c, UDP.h</td>
</tr>
</tbody>
</table>
## Additional Features

The TCP/IP stack includes some additional functionality that can be enabled in `TCPIPConfig.h`.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Macro</th>
<th>Description</th>
<th>Required Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART Demo</td>
<td>STACK_USE_UART</td>
<td>Application demo using UART for IP address display and stack configuration.</td>
<td>UART.c, UART.h</td>
</tr>
<tr>
<td>UART-to-TCP Bridge</td>
<td>STACK_USE_UART2TCP_BRIDGE</td>
<td>UART to TCP Bridge application example</td>
<td>UART2TCPBridge.c, UART2TCP.h</td>
</tr>
<tr>
<td>IP Gleaning</td>
<td>STACK_USE_IP_GLEANING</td>
<td>Allows assignment of an IP address via reception of an ICMP packet with a valid IP during configuration mode</td>
<td>-</td>
</tr>
<tr>
<td>Reboot Server</td>
<td>STACK_USE_REBOOT_SERVER</td>
<td>Allows the PIC to be reset remotely (useful for bootloaders).</td>
<td>Reboot.c, Reboot.h</td>
</tr>
<tr>
<td>UDP Performance Test</td>
<td>STACK_USE_UDP_PERFORMANCE_TEST</td>
<td>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</td>
<td>UDPPTest.c, UDPPTest.h</td>
</tr>
<tr>
<td>TCP Performance Test</td>
<td>STACK_USE_TCP_PERFORMANCE_TEST</td>
<td>last packets to calculate throughput.</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------</td>
<td>---------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Berkeley API</td>
<td>STACK_USE_BERKELEY_API</td>
<td>Provides a Berkeley Sockets API abstraction layer.</td>
<td></td>
</tr>
</tbody>
</table>

TCP performance test. Connect a demo board to a PC via UART, execute the code, and monitor the throughput on the PC terminal.
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 \texttt{TCP/IPConfig.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 \texttt{TCP/IPConfig.h} that is included with the TCP/IP Demo App project.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Allocation</td>
<td>Describes memory allocation macros.</td>
</tr>
<tr>
<td>Socket Types</td>
<td>Describes socket types.</td>
</tr>
<tr>
<td>Initialization Structure</td>
<td>Describes the structure used to declare and initialize TCP sockets.</td>
</tr>
<tr>
<td>UDP Sockets</td>
<td>Describes how UDP sockets are defined.</td>
</tr>
<tr>
<td>BSD Sockets</td>
<td>Describes the allocation of BSD sockets.</td>
</tr>
</tbody>
</table>
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). You can specify the base address in this RAM chip to use with the TCP_SPI_RAM_BASE_ADDRESS macro.
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:

```c
#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 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` 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` and `TCPConnect` 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 layer (also see `BSD Sockets`).
Initialization Structure

In the TCP/IPConfig.h header file, you must also define an array of structures to declare and initialize any sockets that you need. The sample structure is:

```c
#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_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_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. 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.
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:

```c
#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` 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` 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).
The Berkeley API socket configuration option will require Berkeley sockets. Each one of these internally uses one TCP or UDP socket, defined by the TCPSocketInitializer[] array and the MAX_UDP_SOCKETS 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:

```c
#define BSD_SOCKET_COUNT (5u)
```

Configuring the Stack > Protocol Configuration > Sockets > BSD Sockets
Stack API

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.

## Modules

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Announce</td>
<td>Provides a UDP MAC address announcement feature.</td>
</tr>
<tr>
<td>ARP</td>
<td>Provides Address Resolution Protocol support.</td>
</tr>
<tr>
<td>Berkeley (BSD) Sockets</td>
<td>Provides a BSD socket wrapper to the Microchip TCP/IP Stack.</td>
</tr>
<tr>
<td>DNS Client</td>
<td>Provides Domain Name Service resolution.</td>
</tr>
<tr>
<td>Dynamic DNS Client</td>
<td>Updates an external IP address to a Dynamic DNS service.</td>
</tr>
<tr>
<td>Hashes</td>
<td>Calculates MD5 and SHA-1 hash sums.</td>
</tr>
<tr>
<td>Helpers</td>
<td>Provides several helper function for stack operation.</td>
</tr>
<tr>
<td>HTTP2 Server</td>
<td>Provides an advanced embedded web server.</td>
</tr>
<tr>
<td>ICMP</td>
<td>Provides Ping functionality.</td>
</tr>
<tr>
<td>MPFS2</td>
<td>Provides a light-weight file system.</td>
</tr>
<tr>
<td>NBNS</td>
<td>Describes the NetBIOS Name Service protocol.</td>
</tr>
<tr>
<td>Performance Tests</td>
<td>Tests TCP and UDP performance of an application.</td>
</tr>
<tr>
<td>SMTP Client</td>
<td>Sends e-mail messages across the internet.</td>
</tr>
<tr>
<td>Reboot</td>
<td>Provides a service to remotely reboot the PIC.</td>
</tr>
<tr>
<td>SNTP Client</td>
<td>Obtains absolute time stamps from a pool of network time servers.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>SSL</td>
<td>Implements SSL encryption for TCP connections.</td>
</tr>
<tr>
<td>TCP</td>
<td>Implements the TCP transport layer protocol.</td>
</tr>
<tr>
<td>Telnet</td>
<td>Describes the operation of the Telnet module.</td>
</tr>
<tr>
<td>TFTP</td>
<td>Describes the TFTP module.</td>
</tr>
<tr>
<td>Tick Module</td>
<td>Provides accurate time-keeping capabilities.</td>
</tr>
<tr>
<td>UDP</td>
<td>Implements the UDP transport layer protocol.</td>
</tr>
</tbody>
</table>

**Stack API**

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] | [Index] | [Home]
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 PC program.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Announce Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack.</td>
</tr>
</tbody>
</table>

Stack API > Announce
Announce Stack Members

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.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnnounceIP</td>
<td>Transmits an Announce packet.</td>
</tr>
<tr>
<td>DiscoveryTask</td>
<td>Announce callback task.</td>
</tr>
</tbody>
</table>

### Module

**Announce**

Stack API > Announce > Stack Members

---

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
AnnounceIP Function

C

```c
void AnnounceIP();
```

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.

Preconditions

Stack is initialized()

Returns

None

Side Effects

None

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.
DiscoveryTask Function

C

```c
void DiscoveryTask();
```

Description

Recurring task used to listen 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.

Preconditions

Stack is initialized()

Returns

None

Side Effects

None

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.
ARP

The Address 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 and UDPOpen 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.

### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARP Public Members</td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td>ARP Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td>ARP Internal Members</td>
<td>Functions and variables internal to the ARP module</td>
</tr>
</tbody>
</table>

Stack API > ARP
ARP Public Members

The following functions and variables are available to the stack application.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARPResolve</td>
<td>Transmits an ARP request to resolve an IP address.</td>
</tr>
<tr>
<td>ARPIsResolved</td>
<td>Determines if an ARP request has been resolved yet.</td>
</tr>
<tr>
<td>ARPDeRegisterCallbacks</td>
<td>De-Registering callbacks with ARP module that are registered previously.</td>
</tr>
<tr>
<td>ARPRegisterCallbacks</td>
<td>Registering callback with ARP module to get notified about certain events.</td>
</tr>
<tr>
<td>ARPSendPkt</td>
<td>Transmits an ARP request/Reply initiated by Application or external module.</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARP_REQ</td>
<td>Operation code indicating an ARP Request</td>
</tr>
<tr>
<td>ARP_RESP</td>
<td>Operation code indicating an ARP Response</td>
</tr>
<tr>
<td>MAX_REG_APPS</td>
<td>MAX num allowed registrations of Modules/Apps</td>
</tr>
</tbody>
</table>

Module

ARP

Structures
This is record arp_app_callbacks.
**ARPResolve Function**

```c
void ARPResolve(
    IP_ADDR* IPAddr
);
```

**Description**

This function transmits and ARP request to determine the hardware address of a given IP address.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPAddr</td>
<td>The IP address to be resolved. The address must be specified in network byte order (big endian).</td>
</tr>
</tbody>
</table>

**Returns**

None

**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()` function.
ARPIsResolved Function

C

```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.

Preconditions

ARP packet is ready in the MAC buffer.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPAddr</td>
<td>The IP address to be resolved. This must match the IP address provided to the <code>ARPResolve()</code> function call.</td>
</tr>
<tr>
<td>MACAddr</td>
<td>A buffer to store the corresponding MAC address retrieved from the ARP query.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The IP address has been resolved and MACAddr MAC address field indicates the response.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The IP address is not yet resolved. Try calling <code>ARPIsResolved()</code> again at a later time. If you don't get a response after a application specific timeout period, you may want to call <code>ARPResolve()</code> 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.</td>
</tr>
</tbody>
</table>
Remarks

This function is only required when the stack is a client, and therefore is only enabled when STACK_CLIENT_MODE is enabled.
**ARPDeRegisterCallbacks Function**

```c
BOOL ARPDeRegisterCallbacks(
    CHAR 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**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg_id</td>
<td>Registration-id returned in <a href="#">ARPRegisterCallbacks</a> call</td>
</tr>
</tbody>
</table>

**Returns**

TRUE - On success FALSE - Failure to indicate invalid reg_id
CHAR ARPRegisterCallbacks(
    struct arp_app_callbacks * app
);

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 and IP-address). This is an extension for zeroconf protocol implementation (ZeroconfLL.c)

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>app</td>
<td>ARP-Application callbacks structure supplied by user-application</td>
</tr>
</tbody>
</table>

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 and there is no free slot for registration
ARPSendPkt Function

```c
BOOL ARPSendPkt(
    DWORD SrcIPAddr,
    DWORD DestIPAddr,
    BYTE op_req
);
```

Description

following is the workaround algorithm for the 11Mbps broadcast bugfix

This function transmits and ARP request/reply to determine the hardware address of a given IP address (or) Announce self-address to all nodes in network. Extended for zeroconf protocol.

Preconditions

ARP packet is ready in the MAC buffer.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcIPAddr</td>
<td>The Source IP-address</td>
</tr>
<tr>
<td>DestIPAddr</td>
<td>The Destination IP-Address</td>
</tr>
<tr>
<td>op_req</td>
<td>Operation Request (ARP_REQ/ARP_RESP)</td>
</tr>
</tbody>
</table>

Returns

TRUE - The ARP packet was generated properly FALSE - Not possible return value

Remarks

This API is to give control over AR-packet to external modules.
**arp_app_callbacks Structure**

```c
struct arp_app_callbacks {
    BOOL used;
    void (* ARPPkt_notify)(DWORD SenderIPAddr, DWORD TargetIPAddr, MAC_ADDR* SenderMACAddr, MAC_ADDR* TargetMACAddr);
};
```

**Description**

This is record `arp_app_callbacks`.

[Stack API > ARP > Public Members > arp_app_callbacks Structure](#)
ARP_REQ Macro

C

#define ARP_REQ 0x0001u // Operation code indicating an ARP Request

Description

Operation code indicating an ARP Request
### ARP_RESP Macro

```c
#define ARP_RESP 0x0002u // Operation code indicating an ARP Response
```

**Description**

Operation code indicating an ARP Response
MAX_REG_APPS Macro

```c
#define MAX_REG_APPS 2  // MAX num allowed registrations of Modules/Apps
```

Description

MAX num allowed registrations of Modules/Apps
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.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARPInit</td>
<td>Initializes the ARP module.</td>
</tr>
<tr>
<td>ARPProcess</td>
<td>Processes an incoming ARP packet.</td>
</tr>
</tbody>
</table>

### Module

**ARP**

Stack API > ARP > Stack Members
ARPInit Function

```c
void ARPInit();
```

**Description**

Initializes the ARP module. Call this function once at boot to invalidate the cached lookup.

**Preconditions**

None

**Returns**

None

**Remarks**

This function is only required when the stack is a client, and therefore is only enabled when STACK_CLIENT_MODE is enabled.
ARPP⽬Proce≈ Function

C

BOOL ARPProce≈();

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

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>All processing of this ARP packet is complete. Do not call again until a new ARP packet is waiting in the RX buffer.</td>
</tr>
<tr>
<td>FALSE</td>
<td>This function must be called again. More time is needed to send an ARP response.</td>
</tr>
</tbody>
</table>
ARP Internal Members

The following functions and variables are designated as internal to the ARP module.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARPPut</td>
<td>Writes an ARP packet to the MAC.</td>
</tr>
<tr>
<td>SwapARPPacket</td>
<td>Swaps endian-ness of header information in an ARP packet.</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARP_OPERATION_REQ</td>
<td>Operation code indicating an ARP Request</td>
</tr>
<tr>
<td>ARP_OPERATION_RESP</td>
<td>Operation code indicating an ARP Response</td>
</tr>
<tr>
<td>HW_ETHERNET</td>
<td>ARP Hardware type as defined by IEEE 802.3</td>
</tr>
<tr>
<td>ARP_IP</td>
<td>ARP IP packet type as defined by IEEE 802.3</td>
</tr>
</tbody>
</table>

Module

ARP

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>Cache for one ARP response</td>
</tr>
<tr>
<td>reg_apps</td>
<td>Call-Backs storage for MAX of two Modules/Apps</td>
</tr>
</tbody>
</table>
|       | // 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; |
ARPPut Function

C

static BOOL ARPPut(
    ARP_PACKET* packet
);

Description

Writes an ARP packet to the MAC.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packet</td>
<td>A pointer to an ARP_PACKET structure with correct operation and target preconfigured.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The ARP packet was generated properly</td>
</tr>
<tr>
<td>FALSE</td>
<td>Not a possible return value</td>
</tr>
</tbody>
</table>
SwapARPPacket Function

```c
void SwapARPPacket(
    ARP_PACKET* p
);
```

**Description**

Swaps endian-ness of header information in an ARP packet.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>The ARP packet to be swapped</td>
</tr>
</tbody>
</table>

**Returns**

None
ARP_OPERATION_REQ Macro

C

#define ARP_OPERATION_REQ 0x0001u // Operation code indicating an ARP Request

Description

Operation code indicating an ARP Request
ARP_OPERATION_RESP Macro

C

#define ARP_OPERATION_RESP 0x0002u // Operation code indicating an ARP Response

Description

Operation code indicating an ARP Response
HW_ETHERNET Macro

C

#define HW_ETHERNET (0x0001u)  // ARP Hardware type as defined by IEEE 802.3

Description

ARP Hardware type as defined by IEEE 802.3
ARP_IP Macro

```
#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
Cache Variable

C
NODE_INFO Cache;

Description

Cache for one ARP response
reg_apps Variable

```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;

Stack API > ARP > Internal Members > reg_apps Variable
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.
Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BSD Wrapper Public Members</strong></td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td><strong>BSD Wrapper Stack Members</strong></td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td><strong>BSD Wrapper Internal Members</strong></td>
<td>Functions and variables internal to the module</td>
</tr>
</tbody>
</table>

Stack API > BSD Sockets

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc.  All rights reserved.

Contents | Index | Home
The following functions and variables are available to the stack application.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept</td>
<td>This function accepts connection requests queued for a listening socket.</td>
</tr>
<tr>
<td>bind</td>
<td>This function assigns a name to the socket descriptor.</td>
</tr>
<tr>
<td>closesocket</td>
<td>The closesocket function closes an existing socket.</td>
</tr>
<tr>
<td>connect</td>
<td>This function connects to the peer communications end point.</td>
</tr>
<tr>
<td>gethostname</td>
<td>Returns the standard host name for the system.</td>
</tr>
<tr>
<td>listen</td>
<td>The listen function sets the specified socket in a listen mode</td>
</tr>
<tr>
<td>recv</td>
<td>The recv() function is used to receive incoming data that has been queued for a socket.</td>
</tr>
<tr>
<td>recvfrom</td>
<td>The recvfrom() function is used to receive incoming data that has been queued for a socket.</td>
</tr>
<tr>
<td>send</td>
<td>The send function is used to send outgoing data on an already connected socket.</td>
</tr>
<tr>
<td>sendto</td>
<td>This function used to send the data for both connection oriented and connection-less sockets.</td>
</tr>
<tr>
<td>socket</td>
<td>This function creates a new Berkeley socket.</td>
</tr>
</tbody>
</table>

### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF_INET</td>
<td>Internet Address Family - UDP, TCP, etc.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>INADDR_ANY</td>
<td>IP address for server binding.</td>
</tr>
<tr>
<td>INVALID_TCP_PORT</td>
<td>Invalid TCP port</td>
</tr>
<tr>
<td>IP_ADDR_ANY</td>
<td>IP Address for server binding</td>
</tr>
<tr>
<td>IPPROTO_IP</td>
<td>Indicates IP pseudo-protocol.</td>
</tr>
<tr>
<td>IPPROTO_TCP</td>
<td>Indicates TCP for the internet address family.</td>
</tr>
<tr>
<td>IPPROTO_UDP</td>
<td>Indicates UDP for the internet address family.</td>
</tr>
<tr>
<td>SOCK_DGRAM</td>
<td>Connectionless datagram socket. Use UDP for the internet address family.</td>
</tr>
<tr>
<td>SOCK_STREAM</td>
<td>Connection based byte streams. Use TCP for the internet address family.</td>
</tr>
<tr>
<td>SOCKET_CNXN_IN_PROGRESS</td>
<td>Socket connection state.</td>
</tr>
<tr>
<td>SOCKET_DISCONNECTED</td>
<td>Socket disconnected</td>
</tr>
<tr>
<td>SOCKET_ERROR</td>
<td>Socket error</td>
</tr>
</tbody>
</table>

**Module**

**Berkeley (BSD) Sockets**

**Structures**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSDSocket</td>
<td>Berkeley Socket structure</td>
</tr>
<tr>
<td>in_addr</td>
<td>in_addr structure</td>
</tr>
<tr>
<td>sockaddr</td>
<td>generic address structure for all address families</td>
</tr>
<tr>
<td>sockaddr_in</td>
<td>In the Internet address family</td>
</tr>
</tbody>
</table>

**Types**
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOCKADDR</td>
<td>generic address structure for all address families</td>
</tr>
<tr>
<td>SOCKADDR_IN</td>
<td>In the Internet address family</td>
</tr>
<tr>
<td>SOCKET</td>
<td>Socket descriptor</td>
</tr>
</tbody>
</table>
accept Function

C

```c
SOCKET accept(
    SOCKET s,
    struct sockaddr* addr,
    int* addrlen
);
```

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` type socket.

Preconditions

The `listen` function should be called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Socket descriptor returned from a previous call to <code>socket</code>. Must be bound to a local name and in listening mode.</td>
</tr>
<tr>
<td>addr</td>
<td>Optional pointer to a buffer that receives the address of the connecting entity.</td>
</tr>
<tr>
<td>addrlen</td>
<td>Optional pointer to an integer that contains the length of the address <code>addr</code></td>
</tr>
</tbody>
</table>

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 is returned.

Remarks

None.
AF_INET Macro

C

#define AF_INET 2 // Internet Address Family - UDP, TCP, etc.

Description

Internet Address Family - UDP, TCP, etc.

Stack API > BSD Sockets > Public Members > AF_INET Macro
bind Function

```c
int bind(
    SOCKET s,
    const struct sockaddr* name,
    int namelen
);
```

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`, the name of the remote endpoint is assigned when a `connect` or `accept` function is executed.

Preconditions

socket function should be called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Socket descriptor returned from a previous call to socket.</td>
</tr>
<tr>
<td>name</td>
<td>pointer to the <code>sockaddr</code> structure containing the local address of the socket.</td>
</tr>
<tr>
<td>namelen</td>
<td>length of the <code>sockaddr</code> structure.</td>
</tr>
</tbody>
</table>

Returns

If bind is successful, a value of 0 is returned. A return value of `SOCKET_ERROR` indicates an error.

Remarks

None.
BSDSocket Structure

C

```c
struct BSDSocket {
    int SocketType;
    BSD_SCK_STATE bsdState;
    WORD localPort;
    WORD remotePort;
    DWORD remoteIP;
    int backlog;
    BOOL isServer;
    TCP_SOCKET SocketID;
};
```

Description

Berkeley Socket structure

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int SocketType;</td>
<td>Socket type</td>
</tr>
<tr>
<td>BSD_SCK_STATE bsdState;</td>
<td>Socket state</td>
</tr>
<tr>
<td>WORD localPort;</td>
<td>local port</td>
</tr>
<tr>
<td>WORD remotePort;</td>
<td>remote port</td>
</tr>
<tr>
<td>DWORD remoteIP;</td>
<td>remote IP</td>
</tr>
<tr>
<td>int backlog;</td>
<td>maximum number or client connection</td>
</tr>
<tr>
<td>BOOL isServer;</td>
<td>server/client check</td>
</tr>
<tr>
<td>TCP_SOCKET SocketID;</td>
<td>Socket ID</td>
</tr>
</tbody>
</table>
closesocket Function

```c
int closesocket(SOCKET s);
```

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`.

Preconditions

None.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s</code></td>
<td>Socket descriptor returned from a previous call to <code>socket</code></td>
</tr>
</tbody>
</table>

Returns

If `closesocket` is successful, a value of 0 is returned. A return value of `SOCKET_ERROR` (-1) indicates an error.

Remarks

None.
connect Function

```c
int connect(
    SOCKET s,
    struct sockaddr* name,
    int namelen
);
```

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.

Preconditions

- `socket` function should be called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Socket descriptor returned from a previous call to socket.</td>
</tr>
<tr>
<td>name</td>
<td>pointer to the <code>sockaddr</code> structure containing the peer address and port number.</td>
</tr>
<tr>
<td>namelen</td>
<td>length of the <code>sockaddr</code> structure.</td>
</tr>
</tbody>
</table>

Returns

- If the `connect()` function succeeds, it returns 0. Otherwise, the value `SOCKET_ERROR` is returned to indicate an error condition. For stream based socket, if the connection is not established yet, `connect` returns `SOCKET_CNXN_IN_PROGRESS`.

Remarks
None.
gethostname Function

C

```c
int gethostname(
    char* name,
    int namelen
);
```

Description

This function returns the standard host name of the system which is calling this function. The returned name is null-terminated.

Preconditions

None.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Pointer to a buffer that receives the local host name.</td>
</tr>
<tr>
<td>namelen</td>
<td>size of the name array.</td>
</tr>
</tbody>
</table>

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` (-1) will be returned. On error, *name will be unmodified and no null terminator will be generated.

Remarks

None.
in_addr Structure

```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;
};
```

Description

in_addr structure

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 |
INADDR_ANY Macro

C

#define INADDR_ANY 0x00000000u // IP address for server binding.

Description

IP address for server binding.

Stack API > BSD Sockets > Public Members > INADDR_ANY Macro
INVALID_TCP_PORT Macro

```c
#define INVALID_TCP_PORT (0L)  //Invalid TCP port
```

Description

Invalid TCP port
IP_ADDR_ANY Macro

```c
#define IP_ADDR_ANY 0u // IP Address for server binding
```

**Description**

IP Address for server binding
IPPROTO_IP Macro

```
C

#define IPPROTO_IP 0 // Indicates IP pseudo-protocol.
```

Description

Indicates IP pseudo-protocol.
IPPROTO_TCP Macro

C

#define IPPROTO_TCP 6 // Indicates TCP for the internet address family.

Description

Indicates TCP for the internet address family.
IPPROTO_UDP Macro

C

#define IPPROTO_UDP 17 // Indicates UDP for the internet address family.

Description

Indicates UDP for the internet address family.
**listen Function**

```c
int listen(
    SOCKET s,
    int backlog
);
```

**Description**

This function sets the specified socket in a listen mode. Calling the `listen` function indicates that the application is ready to accept connection requests arriving at a socket of type `SOCK_STREAM`. The connection request is queued (if possible) until accepted with an `accept` function. The backlog parameter defines the maximum number of pending connections that may be queued.

**Preconditions**

`bind()` must have been called on the `s` socket first.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Socket identifier returned from a prior <code>socket()</code> call.</td>
</tr>
<tr>
<td>backlog</td>
<td>Maximum number of connection requests that can be queued. Note that each backlog requires a <code>TCP_PURPOSE_BERKELEY_SERVER</code> type TCP socket to be allocated in the <code>TCPSocketInitializer[]</code> in <code>TCPIPConfig.h</code>. Also, ensure that <code>BSD_SOCKET_COUNT</code> (also in <code>TCPIPConfig.h</code>) is greater than the backlog by at least 1 (more if you have other BSD sockets in use).</td>
</tr>
</tbody>
</table>

**Returns**

Returns 0 on success, else return `SOCKET_ERROR`. 
Remarks

None
recv Function

C

```c
int recv(
    SOCKET s,  
    char* buf, 
    int len,   
    int flags
);
```

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` type sockets. For `SOCK_STREAM` types, the data is buffered internally so the application can retrieve all data by multiple calls of `recvfrom`.

Preconditions

`connect` function should be called for TCP and UDP sockets. Server side, `accept` function should be called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Socket descriptor returned from a previous call to socket.</td>
</tr>
<tr>
<td>buf</td>
<td>application data receive buffer.</td>
</tr>
<tr>
<td>len</td>
<td>buffer length in bytes.</td>
</tr>
<tr>
<td>flags</td>
<td>no significance in this implementation</td>
</tr>
</tbody>
</table>

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 (-1) indicates an error condition. A return value of SOCKET_DISCONNECTED indicates the connection no longer exists.

Remarks

None.
recvfrom Function

C

```c
int recvfrom(
    SOCKET s,
    char* buf,
    int len,
    int flags,
    struct sockaddr* from,
    int* fromlen
);
```

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` type sockets. For `SOCK_STREAM` types, the data is buffered internally so the application can retrieve all data by multiple calls of `recvfrom`.

Preconditions

- `socket` function should be called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Socket descriptor returned from a previous call to <code>socket</code>.</td>
</tr>
<tr>
<td>buf</td>
<td>Application data receive buffer.</td>
</tr>
<tr>
<td>len</td>
<td>Buffer length in bytes.</td>
</tr>
<tr>
<td>flags</td>
<td>Message flags. Currently this is not supported.</td>
</tr>
<tr>
<td>from</td>
<td>Pointer to the <code>sockaddr</code> structure that will be filled in with the destination address.</td>
</tr>
</tbody>
</table>
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** (-1) indicates an error condition.

Remarks

None.
send Function

```c
int send(
    SOCKET s,
    const char* buf,
    int len,
    int flags
);
```

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` but can also be used to send datagrams on a socket of type `SOCK_DGRAM`.

Preconditions

The `connect` function should be called for TCP and UDP sockets. Server side, the `accept` function should be called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Socket descriptor returned from a previous call to <code>socket</code>.</td>
</tr>
<tr>
<td>buf</td>
<td>Application data buffer containing data to transmit.</td>
</tr>
<tr>
<td>len</td>
<td>Length of data in bytes.</td>
</tr>
<tr>
<td>flags</td>
<td>Message flags. Currently this field is not supported.</td>
</tr>
</tbody>
</table>

Returns

On success, `send` returns number of bytes sent. In case of error, returns `SOCKET_ERROR`. A zero indicates no data send.
Remarks

None.
sendto Function

C

```c
int sendto(
    SOCKET s,
    const char* buf,
    int len,
    int flags,
    const struct sockaddr* to,
    int tolen
);
```

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.

Preconditions

socket function should be called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Socket descriptor returned from a previous call to <code>socket</code>.</td>
</tr>
<tr>
<td>buf</td>
<td>Application data buffer containing data to transmit.</td>
</tr>
<tr>
<td>len</td>
<td>Length of data in bytes.</td>
</tr>
<tr>
<td>flags</td>
<td>Message flags. Currently this field is not supported.</td>
</tr>
<tr>
<td>to</td>
<td>Optional pointer to the <code>sockaddr</code> structure containing the destination address. If NULL, the currently bound remote port and IP address are used as the destination.</td>
</tr>
<tr>
<td>tolen</td>
<td>Length of the <code>sockaddr</code> structure.</td>
</tr>
</tbody>
</table>

Returns
On success, `sendto` returns number of bytes sent. In case of error returns `SOCKET_ERROR`.

**Remarks**

None.
#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.
SOCK_STREAM Macro

C

#define SOCK_STREAM 100 //Connection based byte streams. Use TCP for

Description

Connection based byte streams. Use TCP for the internet address family.
sockaddr Structure

```c
struct sockaddr {
    unsigned short sa_family;
    char sa_data[14];
};
```

Description

generic address structure for all address families

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short sa_family;</td>
<td>address family</td>
</tr>
<tr>
<td>char sa_data[14];</td>
<td>up to 14 bytes of direct address</td>
</tr>
</tbody>
</table>

Stack API > BSD Sockets > Public Members > sockaddr Structure
SOCKADDR Type

typedef struct sockaddr SOCKADDR;

Description

generic address structure for all address families
### sockaddr_in Structure

```c
struct sockaddr_in {
    short sin_family;
    WORD sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};
```

#### Description

In the Internet address family

#### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short sin_family;</td>
<td>Address family; must be AF_INET.</td>
</tr>
<tr>
<td>WORD sin_port;</td>
<td>Internet Protocol (IP) port.</td>
</tr>
<tr>
<td>struct in_addr sin_addr;</td>
<td>IP address in network byte order.</td>
</tr>
<tr>
<td>char sin_zero[8];</td>
<td>Padding to make structure the same size as SOCKADDR.</td>
</tr>
</tbody>
</table>

[Stack API] [BSD Sockets] [Public Members] [sockaddr_in Structure]
### SOCKADDR_IN Type

```c
typedef struct sockaddr_in SOCKADDR_IN;
```

#### Description

In the Internet address family

[Stack API > BSD Sockets > Public Members > SOCKADDR_IN Type](#)
socket Function

C

```c
SOCKET socket(
    int af,
    int type,
    int protocol
);
```

Description

This function creates a new BSD socket for the microchip TCPIP stack. The return socket descriptor is used for the subsequent BSD operations.

Preconditions

The `BerkeleySocketInit` function should be called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>af</td>
<td>address family - <code>AF_INET</code>.</td>
</tr>
<tr>
<td>type</td>
<td>socket type <code>SOCK_DGRAM</code> or <code>SOCK_STREAM</code>.</td>
</tr>
<tr>
<td>protocol</td>
<td>IP protocol <code>IPPROTO_UDP</code> or <code>IPPROTO_TCP</code>.</td>
</tr>
</tbody>
</table>

Returns

New socket descriptor. `INVALID_SOCKET` in case of error.

Remarks

None.
SOCKET Type

typedef BYTE SOCKET;

Description

Socket descriptor
#define SOCKET_CNXN_IN_PROGRESS (-2) //Socket connection state.

## Description

Socket connection state.
#define SOCKET_DISCONNECTED (-3) //Socket disconnected

**Description**

Socket disconnected
SOCKET_ERROR Macro

C

#define SOCKET_ERROR (-1) //Socket error

Description

Socket error
BSD Wrapper Stack Members

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.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BerkeleySocketInit</td>
<td>Initializes the Berkeley socket structure array.</td>
</tr>
</tbody>
</table>

Module

Berkeley (BSD) Sockets

Stack API > BSD Sockets > Stack Members

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
BerkeleySocketInit Function

```
C

void BerkeleySocketInit();
```

Description

This function initializes the Berkeley socket array. This function should be called before any BSD socket call.

Preconditions

None.

Returns

None

Remarks

None.

Stack API > BSD Sockets > Stack Members > BerkeleySocketInit Function
BSD Wrapper Internal Members

The following functions and variables are designated as internal to the module.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSD_SCK_STATE</td>
<td>Berkeley Socket (BSD) states</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandlePossibleTCPDisconnection</td>
<td>Internal function that checks for asynchronous TCP connection state changes and resynchs the BSD socket descriptor state to match.</td>
</tr>
</tbody>
</table>

Module

Berkeley (BSD) Sockets

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSDSocketArray</td>
<td>Array of BSDSocket elements; used to track all socket state and connection information.</td>
</tr>
<tr>
<td>gAutoPortNumber</td>
<td>Contains the next local port number to associate with a socket.</td>
</tr>
</tbody>
</table>
### BSD_SCK_STATE Enumeration

```c
typedef enum {
    SKT_CLOSED,
    SKT_CREATED,
    SKT_BOUND,
    SKT_BSD_LISTEN,
    SKT_LISTEN,
    SKT_IN_PROGRESS,
    SKT_EST,
    SKT_DISCONNECTED
} BSD_SCK_STATE;
```

### Description

Berkeley Socket (BSD) states

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKT_CLOSED</td>
<td>Socket closed state indicating a free descriptor</td>
</tr>
<tr>
<td>SKT_CREATED</td>
<td>Socket created state for TCP and UDP sockets</td>
</tr>
<tr>
<td>SKT_BOUND</td>
<td>Socket bound state for TCP and UDP sockets</td>
</tr>
<tr>
<td>SKT_BSD_LISTEN</td>
<td>Listening state for TCP BSD listener handle &quot;socket</td>
</tr>
<tr>
<td>SKT_LISTEN</td>
<td>TCP server listen state</td>
</tr>
<tr>
<td>SKT_IN_PROGRESS</td>
<td>TCP client connection in progress state</td>
</tr>
<tr>
<td>SKT_EST</td>
<td>TCP client or server established state</td>
</tr>
<tr>
<td>SKT_DISCONNECTED</td>
<td>TCP client or server no longer connected to the remote host (but was historically)</td>
</tr>
</tbody>
</table>
**BSDSocketArray Variable**

```
C

struct BSDSocket BSDSocketArray[BSD_SOCKET_COUNT];
```

**Description**

Array of **BSDSocket** elements; used to track all socket state and connection information.

---

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] [Index] [Home]
gAutoPortNumber Variable

C

WORD gAutoPortNumber = 1024;

Description

Contains the next local port number to associate with a socket.
HandlePossibleTCPDisconnection Function

C

```c
static BOOL HandlePossibleTCPDisconnection(
    SOCKET s
);
```

Description

Internal function that checks for asynchronous TCP connection state changes and resynchs the BSD socket descriptor state to match.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

TRUE - Socket is disconnected FALSE - Socket is

Stack API > BSD Sockets > Internal Members > HandlePossibleTCPDisconnection Function
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 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.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DNS Public Members</strong></td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td><strong>DNS Internal Members</strong></td>
<td>Functions and variables internal to the DNS module</td>
</tr>
</tbody>
</table>

Stack API > DNS
DNS Public Members

The following functions and variables are available to the stack application.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNSBeginUsage</td>
<td>Claims access to the DNS module.</td>
</tr>
<tr>
<td>DNSEndUsage</td>
<td>Releases control of the DNS module.</td>
</tr>
<tr>
<td>DNSResolve</td>
<td>Begins resolution of an address.</td>
</tr>
<tr>
<td>DNSResolveROM</td>
<td>Begins resolution of an address.</td>
</tr>
<tr>
<td>DNSIsResolved</td>
<td>Determines if the DNS resolution is complete and provides the IP.</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS_TYPE_A</td>
<td>Constant for record type in DNSResolve. Indicates an A (standard address) record.</td>
</tr>
<tr>
<td>DNS_TYPE_MX</td>
<td>Constant for record type in DNSResolve. Indicates an MX (mail exchanger) record.</td>
</tr>
</tbody>
</table>

Module

DNS Client
DNSBeginUsage Function

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 when this application no longer needs the DNS module so that other applications may make use of it.

Preconditions

Stack is initialized.

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>No other DNS resolutions are in progress and the calling application has sucessfully taken ownership of the DNS module</td>
</tr>
<tr>
<td>FALSE</td>
<td>The DNS module is currently in use. Yield to the stack and attempt this call again later.</td>
</tr>
</tbody>
</table>

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.
**DNSEndUsage Function**

```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.

## Preconditions

- **DNSEbeginUsage** returned TRUE on a previous call.

## Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The address to the host name was successfully resolved.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The DNS failed or the address does not exist.</td>
</tr>
</tbody>
</table>

## 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.
**DNSResolve Function**

```c
void DNSResolve(
    BYTE* HostName,
    BYTE Type
);
```

**Description**

This function attempts to resolve a host name to an IP address. When called, it starts the DNS state machine. Call `DNSIsResolved` 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.

**Preconditions**

`DNSBeginUsage` returned TRUE on a previous call.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostname</td>
<td>A pointer to the null terminated string specifying the host for which to resolve an IP.</td>
</tr>
<tr>
<td>RecordType</td>
<td><code>DNS_TYPE_A</code> or <code>DNS_TYPE_MX</code> depending on what type of record resolution is desired.</td>
</tr>
</tbody>
</table>

**Returns**

None

**Remarks**

This function requires access to one UDP socket. If none are available, `MAX_UDP_SOCKETS` may need to be increased.
## DNSResolveROM Function

### C

```c
void DNSResolveROM(
    ROM BYTE* Hostname,
    BYTE Type
);
```

### Description

This function attempts to resolve a host name to an IP address. When called, it starts the DNS state machine. Call `DNSIsResolved` 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.

### Preconditions

`DNSBeginUsage` returned TRUE on a previous call.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostname</td>
<td>A pointer to the null terminated string specifying the host for which to resolve an IP.</td>
</tr>
<tr>
<td>RecordType</td>
<td><code>DNS_TYPE_A</code> or <code>DNS_TYPE_MX</code> depending on what type of record resolution is desired.</td>
</tr>
</tbody>
</table>

### Returns

None

### 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** on non-PIC18 platforms.
**DNSIsResolved Function**

```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 or DNSResolveROM has been called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HostIP</td>
<td>A pointer to an IP_ADDR structure in which to store the resolved IP address once resolution is complete.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>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.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The resolution process is still in progress.</td>
</tr>
</tbody>
</table>
DNS_TYPE_A Macro

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. Indicates an A (standard address) record.

Stack API > DNS > Public Members > DNS_TYPE_A Macro

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
### DNS_TYPE_MX Macro

```c
#define DNS_TYPE_MX (15u)  // Constant for record type in DNSResolve.
```

#### Description

Constant for record type in **DNSResolve**. Indicates an MX (mail exchanger) record.
## DNS Internal Members

The following functions and variables are designated as internal to the DNS module.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNSPutString</td>
<td>Writes a string to the DNS socket.</td>
</tr>
<tr>
<td>DNSPutROMString</td>
<td>Writes a ROM string to the DNS socket.</td>
</tr>
<tr>
<td>DNSDiscardName</td>
<td>Reads a name string or string pointer from the DNS socket and discards it.</td>
</tr>
</tbody>
</table>

### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS_PORT</td>
<td>Default port for DNS resolutions</td>
</tr>
<tr>
<td>DNS_TIMEOUT</td>
<td>Elapsed time after which a DNS resolution is considered to have timed out</td>
</tr>
</tbody>
</table>

### Module

**DNS Client**

### Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS_HEADER</td>
<td>Structure for the DNS header</td>
</tr>
</tbody>
</table>

### Variables
<table>
<thead>
<tr>
<th></th>
<th>DNSHostName</th>
<th>Host name in RAM to look up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DNSHostNameROM</td>
<td>Host name in ROM to look up</td>
</tr>
<tr>
<td></td>
<td>Flags</td>
<td>Stores various flags for the UDP module</td>
</tr>
<tr>
<td></td>
<td>RecordType</td>
<td>Record type being queried</td>
</tr>
<tr>
<td></td>
<td>ResolvedInfo</td>
<td>Node information about the resolved node</td>
</tr>
<tr>
<td></td>
<td>smDNS</td>
<td>State machine for a DNS query</td>
</tr>
</tbody>
</table>

**Stack API > DNS > Internal Members**
### DNSPutString Function

```c
static void DNSPutString(
    BYTE* String
);
```

### 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>the string to write to the UDP socket.</td>
</tr>
</tbody>
</table>

### Returns

None
C

```c
static void DNSPutROMString(
    ROM BYTE* String
);
```

### 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>the string to write to the UDP socket.</td>
</tr>
</tbody>
</table>

### Returns

None

### Remarks

This function is aliased to `DNSPutString` on non-PIC18 platforms.
**DNS_PORT Macro**

```
#define DNS_PORT 53u // Default port for DNS resolutions
```

**Description**

Default port for DNS resolutions
#define DNS_TIMEOUT (TICK_SECOND*1) // Elapsed time after which a DNS resolution is considered to have timed out

Description

ElapsedTime after which a DNS resolution is considered to have timed out
DNSHostName Variable

C

BYTE * DNSHostName;

Description

Host name in RAM to look up
DNSHostNameROM Variable

C
ROM BYTE * DNSHostNameROM;

Description

Host name in ROM to look up
Flags Variable

C

```c
struct {
    unsigned char bFirstRead : 1;
    unsigned char bWasDiscarded : 1;
} Flags;
```

Description

Stores various flags for the UDP module

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char bFirstRead : 1;</td>
<td>No data has been read from this segment yet</td>
</tr>
<tr>
<td>unsigned char bWasDiscarded : 1;</td>
<td>The data in this segment has been discarded</td>
</tr>
</tbody>
</table>

Stack API > DNS > Internal Members > Flags Variable
RecordType Variable

| C
| BYTE RecordType;

Description

Record type being queried

Stack API > DNS > Internal Members > RecordType Variable
ResolvedInfo Variable

C
NODE_INFO ResolvedInfo;

Description

Node information about the resolved node
### Description

State machine for a DNS query

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS_START = 0</td>
<td>Initial state to reset client state variables</td>
</tr>
<tr>
<td>DNS_ARP_START_RESOLVE</td>
<td>Send ARP resolution of DNS server or gateway MAC address</td>
</tr>
<tr>
<td>DNS_ARP_RESOLVE</td>
<td>Wait for response to ARP request</td>
</tr>
<tr>
<td>DNS_OPEN_SOCKET</td>
<td>Open UDP socket</td>
</tr>
<tr>
<td>DNS_QUERY</td>
<td>Send DNS query to DNS server</td>
</tr>
<tr>
<td>DNS_GET_RESULT</td>
<td>Wait for response from DNS server</td>
</tr>
<tr>
<td>DNS_FAIL</td>
<td>ARP or DNS server not responding</td>
</tr>
<tr>
<td>DNS_DONE</td>
<td>DNS query is finished</td>
</tr>
</tbody>
</table>

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
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
DNSDiscardName Function

C

static void DNSDiscardName();

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

Returns

None
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, No-IP.com, and 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.

### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic DNS Public Members</td>
<td>Functions and variables accessed by the stack application</td>
</tr>
<tr>
<td>Dynamic DNS Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td>Dynamic DNS Internal Members</td>
<td>Functions and variables internal to the Dynamic DNS module</td>
</tr>
</tbody>
</table>

Stack API > Dynamic DNS Client
Dynamic DNS Public Members

These functions and variables are meant to be called by your stack application.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDNS SERVICES</td>
<td>Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to ddnsServiceHosts and ddnsServicePorts in DynDNS.c.</td>
</tr>
<tr>
<td>DDNS STATUS</td>
<td>Status message for DynDNS client. GOOD and NOCHG are ok, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDNSForceUpdate</td>
<td>Forces an immediate DDNS update</td>
</tr>
<tr>
<td>DDNSGetLastIP</td>
<td>Returns the last known external IP address of the device.</td>
</tr>
<tr>
<td>DDNSGetLastStatus</td>
<td>Returns the status of the most recent update.</td>
</tr>
<tr>
<td>DDNSSetService</td>
<td>Selects a pre-configured Dynamic DNS service</td>
</tr>
</tbody>
</table>

Module

Dynamic DNS Client

Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDNS POINTERS</td>
<td>Configuration parameters for the Dynamic DNS Client</td>
</tr>
</tbody>
</table>
# Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDNSClient</td>
<td>Configuration parameters for the module</td>
</tr>
</tbody>
</table>

[Stack API] > [Dynamic DNS Client] > [Public Members]
DDNS_POINTERS Structure

```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 `DDNS_Client.[field name].szRAM` to use a string stored in RAM, or `DDNS_Client.[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, `DDNS_Client.ROMPointers.[field name]` must
also be set to 1 to indicate that this field should be retrieved from ROM instead of RAM.

## Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckIPServer</td>
<td>The server used to determine the external IP address</td>
</tr>
<tr>
<td>CheckIPPort</td>
<td>Port on the above server to <a href="#">connect</a> to</td>
</tr>
<tr>
<td>UpdateServer</td>
<td>The server where updates should be posted</td>
</tr>
<tr>
<td>UpdatePort</td>
<td>Port on the above server to <a href="#">connect</a> to</td>
</tr>
<tr>
<td>Username</td>
<td>The user name for the dynamic DNS server</td>
</tr>
<tr>
<td>Password</td>
<td>The password to supply when making updates</td>
</tr>
<tr>
<td>Host</td>
<td>The host name you wish to update</td>
</tr>
<tr>
<td>ROMPointers</td>
<td>Indicates which parameters to read from ROM instead of RAM.</td>
</tr>
</tbody>
</table>

[Stack API] > [Dynamic DNS Client] > [Public Members] > [DDNS_POINTERS Structure]
typedef enum {
    DYNDNS_ORG = 0u,
    NO_IP_COM,
    DNSOMATIC_COM
} DDNS_SERVICES;

**Description**

Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to `ddnsServiceHosts` and `ddnsServicePorts` in DynDNS.c.

**Members**

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DYNDNS_ORG = 0u</td>
<td><a href="http://www.dyndns.org">www.dyndns.org</a></td>
</tr>
<tr>
<td>NO_IP_COM</td>
<td><a href="http://www.no-ip.com">www.no-ip.com</a></td>
</tr>
<tr>
<td>DNSOMATIC_COM</td>
<td><a href="http://www.dnsomatic.com">www.dnsomatic.com</a></td>
</tr>
</tbody>
</table>

Stack API > Dynamic DNS Client > Public Members > DDNS SERVICES Enumeration
typedef enum {
    DDNS_STATUS_GOOD = 0u,
    DDNS_STATUS_NOCHG,
    DDNS_STATUS_ABUSE,
    DDNS_STATUS_BADSYS,
    DDNS_STATUS_BADAGENT,
    DDNS_STATUS_BADAUTH,
    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;

Description

Status message for DynDNS client. GOOD and NOCHG are ok, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDNS_STATUS_GOOD = 0u</td>
<td>Update successful, hostname is now updated</td>
</tr>
<tr>
<td>DDNS_STATUS_NOCHG</td>
<td>Update changed no setting and is considered abusive. Additional 'nochg' updates will cause hostname to be blocked.</td>
</tr>
<tr>
<td>DDNS_STATUS_ABUSE</td>
<td>The hostname specified is blocked for update abuse.</td>
</tr>
<tr>
<td>DDNS_STATUS_BADSYS</td>
<td>System parameter not valid. Should be dyndns, statdns or custom.</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>DDNS_STATUS_BADAGENT</td>
<td>The user agent was blocked or not sent.</td>
</tr>
<tr>
<td>DDNS_STATUS_BADAUTH</td>
<td>The username and password pair do not match a real user.</td>
</tr>
<tr>
<td>DDNS_STATUS_NOT_DONATOR</td>
<td>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.</td>
</tr>
<tr>
<td>DDNS_STATUS_NOT_FQDN</td>
<td>The hostname specified is not a fully-qualified domain name (not in the form hostname.dyndns.org or domain.com).</td>
</tr>
<tr>
<td>DDNS_STATUS_NOHOST</td>
<td>The hostname specified does not exist in this user account (or is not in the service specified in the system parameter).</td>
</tr>
<tr>
<td>DDNS_STATUS_NOT_YOURS</td>
<td>The hostname specified does not belong to this user account.</td>
</tr>
<tr>
<td>DDNS_STATUS_NUMHOST</td>
<td>Too many hosts specified in an update.</td>
</tr>
<tr>
<td>DDNS_STATUS_DNSERR</td>
<td>Unspecified DNS error encountered by the DDNS service.</td>
</tr>
<tr>
<td>DDNS_STATUS_911</td>
<td>There is a problem or scheduled maintenance with the DDNS service.</td>
</tr>
<tr>
<td>DDNS_STATUS_UPDATE_ERROR</td>
<td>Error communicating with Update service.</td>
</tr>
<tr>
<td>DDNS_STATUS_UNCHANGED</td>
<td>The IP Check indicated that no update was necessary.</td>
</tr>
<tr>
<td>DDNS_STATUS_CHECKIP_ERROR</td>
<td>Error communicating with CheckIP service.</td>
</tr>
<tr>
<td>DDNS_STATUS_INVALID</td>
<td>DDNS Client data is not valid.</td>
</tr>
<tr>
<td>DDNS_STATUS_UNKNOWN</td>
<td>DDNS client has not yet been executed with this configuration.</td>
</tr>
</tbody>
</table>
DDNSClient Variable

C

DDNS_POINTERS DDNSClient;

Description

Configuration parameters for the module
DDNSForceUpdate Function

C

```c
void DDNSForceUpdate();
```

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 parameters have been modified.

Preconditions

DDNSInit must have been called.

Returns

None
DDNSGetLastIP Function

C

IP_ADDR DDNSGetLastIP();

Description

This function returns the last known external IP address of the device.

Preconditions

None

Returns

The last known external IP address of the device.
**DDNSGetLastStatus Function**

```
c DDNS_STATUS DDNSGetLastStatus();
```

**Description**

This function returns the status of the most recent update. See the `DDNS_STATUS` enumeration for possible codes.

**Preconditions**

None

**Returns**

`DDNS_STATUS` indicating the status code for the most recent update.
DD NSSetService Function

C

\texttt{\textbf{void} DD NSSetService( \textbf{DDNS\_SERVICES} \texttt{svc} );}

Description

This function selects a Dynamic DNS service based on parameters configured in \texttt{ddnsServiceHosts} and \texttt{ddnsServicePorts}. These arrays must match the \texttt{DDNS\_SERVICES} enumeration.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{svc}</td>
<td>one of the \texttt{DDNS_SERVICES} elements to indicate the selected service</td>
</tr>
</tbody>
</table>

Returns

None
Dynamic DNS Stack Members

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.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDNSInit</td>
<td>Initializes the Dynamic DNS module.</td>
</tr>
<tr>
<td>DDNSTask</td>
<td>Dynamic DNS client task/state machine.</td>
</tr>
</tbody>
</table>

Module

Dynamic DNS Client

Stack API > Dynamic DNS Client > Stack Members
DDNSInit Function

C


void DDNSInit();

Description

This function initializes the Dynamic DNS client. It clears the DDNSClient 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).

Preconditions

None

Returns

None

Remarks

This function is called only one during lifetime of the application.
DDNSTask Function

C

```c
void DDNSTask();
```

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` elapses, which by default is every 10 minutes, or when an update is forced.

Preconditions

`DDNSInit()` has been called.

Returns

None

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.
Dynamic DNS Internal Members

The following functions and variables are designated as internal to the Dynamic DNS module.

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDNS_CHECKIP_SERVER</td>
<td>Default CheckIP server for determining current IP address</td>
</tr>
<tr>
<td>DDNS_DEFAULT_PORT</td>
<td>Default port for CheckIP server</td>
</tr>
</tbody>
</table>

Module

**Dynamic DNS Client**

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bForceUpdate</td>
<td>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.</td>
</tr>
<tr>
<td>ddnsServiceHosts</td>
<td>Host names for various Dynamic DNS services</td>
</tr>
<tr>
<td>ddnsServicePorts</td>
<td>Port numbers for various Dynamic DNS services</td>
</tr>
<tr>
<td>dwUpdateAt</td>
<td>Indicates when the next CheckIP should be done</td>
</tr>
<tr>
<td>lastKnownIP</td>
<td>Last known IP address of this device</td>
</tr>
<tr>
<td>lastStatus</td>
<td>Status response from last update</td>
</tr>
<tr>
<td>_checkIpSrvrResponse</td>
<td>Delimiter to locate IP address from CheckIP server</td>
</tr>
<tr>
<td>_updateIpSrvrResponse</td>
<td>Response codes from <a href="http://www.dyndns.org">DynDNS</a> Update Server</td>
</tr>
</tbody>
</table>
bForceUpdate Variable

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.
**ddnsServiceHosts Variable**

| C
| ---
| ROM char * ROM ddnsServiceHosts[];

### Description

Host names for various Dynamic DNS services

[Stack API] > [Dynamic DNS Client] > [Internal Members] > ddnsServiceHosts Variable
ddnsServicePorts Variable

C
ROM WORD ddnsServicePorts[] = { 80, 80, 80, };

Description

Port numbers for various Dynamic DNS services

Stack API > Dynamic DNS Client > Internal Members > ddnsServicePorts Variable
**dwUpdateAt Variable**

```
C
DWORD dwUpdateAt;
```

**Description**

Indicates when the next CheckIP should be done
lastKnownIP Variable

C
IP_ADDR lastKnownIP;

Description

Last known IP address of this device
lastStatus Variable

C

DDNS_STATUS lastStatus;

Description

Status response from last update
_checkIpSrvrResponse Variable

C

ROM BYTE _checkIpSrvrResponse[] = "Address:"

Description

Delimiter to locate IP address from CheckIP server
_updateIpSrvrResponse Variable

C

ROM char* _updateIpSrvrResponse[] = { "good", "nochg", "abuse", "badsy

Description

Response codes from DynDNS Update Server

Stack API > Dynamic DNS Client > Internal Members > _updateIpSrvrResponse Variable
#define DDNS_CHECKIP_SERVER (ROM BYTE*)"checkip.dyndns.com" //

Description

Default CheckIP server for determining current IP address
DDNS_DEFAULT_PORT Macro

C

#define DDNS_DEFAULT_PORT (80u)

// Default port for CheckIP server

Description

Default port for CheckIP server
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 structure and pass a pointer to it to either MD5Initialize or SHA1Initialize. Then, call HashAddData or HashAddROMData as many times as are necessary to provide all the data to the hash. Call MD5Calculate or SHA1Calculate 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.

Topics
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hashes Public Members</strong></td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td><strong>Hashes Stack Members</strong></td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td><strong>Hashes Internal Members</strong></td>
<td>Functions and variables internal to the Hashes module</td>
</tr>
</tbody>
</table>
Hashes Public Members

The following functions and variables are available to the stack application.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HashAddData</td>
<td>Adds data to the hash sum.</td>
</tr>
<tr>
<td>HashAddROMData</td>
<td>Adds data to the hash sum.</td>
</tr>
<tr>
<td>MD5Calculate</td>
<td>Calculates an MD5 hash</td>
</tr>
<tr>
<td>MD5Initialize</td>
<td>Initializes a new MD5 hash.</td>
</tr>
<tr>
<td>SHA1Calculate</td>
<td>Calculates a SHA-1 hash</td>
</tr>
<tr>
<td>SHA1Initialize</td>
<td>Initializes a new SHA-1 hash.</td>
</tr>
</tbody>
</table>

Module

Hashes

Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASH_SUM</td>
<td>Context storage for a hash operation</td>
</tr>
</tbody>
</table>

Stack API > Hashes > Public Members
HashAddData Function

C

```c
void HashAddData(
    HASH_SUM* theSum,
    BYTE* data,
    WORD len
);
```

Description

Adds data to the hash sum.

Preconditions

The hash sum has already been initialized

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theSum</td>
<td>hash context state</td>
</tr>
<tr>
<td>data</td>
<td>the data to be added to the hash sum</td>
</tr>
<tr>
<td>len</td>
<td>length of data</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

This function calls the appropriate hashing function based on the hash typed defined in theSum.
HashAddROMData Function

```c
void HashAddROMData(
    HASH_SUM* theSum,
    ROM_BYTE* data,
    WORD len
);
```

Description

Adds data to the hash sum.

Preconditions

The hash sum has already been initialized

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theSum</td>
<td>hash context state</td>
</tr>
<tr>
<td>data</td>
<td>the data to be added to the hash sum</td>
</tr>
<tr>
<td>len</td>
<td>length of data</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

This function calls the appropriate hashing function based on the hash typed defined in theSum.

This function is aliased to HashAddData on non-PIC18 platforms.
MD5Calculate Function

C

```c
void MD5Calculate(  
    HASH_SUM* theSum,  
    BYTE* result
);
```

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theSum</td>
<td>the current hash context</td>
</tr>
<tr>
<td>result</td>
<td>16 byte array in which to store the resulting hash</td>
</tr>
</tbody>
</table>

Returns

None
C

void MD5Initialize(
    HASH_SUM* theSum
);

Description

Initializes a new MD5 hash.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theSum</td>
<td>pointer to the allocated HASH_SUM object to initialize as MD5</td>
</tr>
</tbody>
</table>

Returns

None
SHA1Calculate Function

```c
void SHA1Calculate(
    HASH_SUM* theSum,
    BYTE* result
);
```

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theSum</td>
<td>the current hash context</td>
</tr>
<tr>
<td>result</td>
<td>20 byte array in which to store the resulting hash</td>
</tr>
</tbody>
</table>

Returns

None
SHA1Initialize Function

C

```c
void SHA1Initialize(
    HASH_SUM* theSum
);
```

Description

Initializes a new SHA-1 hash.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theSum</td>
<td>pointer to the allocated HASH_SUM object to initialize as SHA-1</td>
</tr>
</tbody>
</table>

Returns

None

Stack API > Hashes > Public Members > SHA1Initialize Function
HASH_SUM Structure

C

typedef struct {
    DWORD h0;
    DWORD h1;
    DWORD h2;
    DWORD h3;
    DWORD h4;
    DWORD bytesSoFar;
    BYTE partialBlock[64];
    HASH_TYPE hashType;
} HASH_SUM;

Description

Context storage for a hash operation

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD h0;</td>
<td>Hash state h0</td>
</tr>
<tr>
<td>DWORD h1;</td>
<td>Hash state h1</td>
</tr>
<tr>
<td>DWORD h2;</td>
<td>Hash state h2</td>
</tr>
<tr>
<td>DWORD h3;</td>
<td>Hash state h3</td>
</tr>
<tr>
<td>DWORD h4;</td>
<td>Hash state h4</td>
</tr>
<tr>
<td>DWORD bytesSoFar;</td>
<td>Total number of bytes hashed so far</td>
</tr>
<tr>
<td>BYTE partialBlock[64];</td>
<td>Beginning of next 64 byte block</td>
</tr>
<tr>
<td>HASH_TYPE hashType;</td>
<td>Type of hash being calculated</td>
</tr>
</tbody>
</table>
Hashes Stack Members

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.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5AddROMData</td>
<td>Adds data to an MD5 hash calculation.</td>
</tr>
<tr>
<td>SHA1AddROMData</td>
<td>Adds data to a SHA-1 hash calculation.</td>
</tr>
<tr>
<td>SHA1AddData</td>
<td>Adds data to a SHA-1 hash calculation.</td>
</tr>
<tr>
<td>MD5AddData</td>
<td>Adds data to an MD5 hash calculation.</td>
</tr>
</tbody>
</table>

Module

Hashes

Stack API > Hashes > Stack Members
MD5AddROMData Function

C

```c
void MD5AddROMData(
    HASH_SUM* theSum,
    ROM BYTE* data,
    WORD len
);
```

Description

Adds data to an MD5 hash calculation.

Preconditions

The hash context has already been initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theSum</td>
<td>a pointer to the hash context structure</td>
</tr>
<tr>
<td>data</td>
<td>the data to add to the hash</td>
</tr>
<tr>
<td>len</td>
<td>the length of the data to add</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

This function is aliased to MD5AddData on non-PIC18 platforms.
SHA1AddROMData Function

C

```c
void SHA1AddROMData(
    HASH_SUM* theSum,
    ROM_BYTE* data,
    WORD len
);
```

Description

Adds data to a SHA-1 hash calculation.

Preconditions

The hash context has already been initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theSum</td>
<td>a pointer to the hash context structure</td>
</tr>
<tr>
<td>data</td>
<td>the data to add to the hash</td>
</tr>
<tr>
<td>len</td>
<td>the length of the data to add</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

This function is aliased to [SHA1AddData](#) on non-PIC18 platforms.
**SHA1AddData Function**

```
C

void SHA1AddData(
    HASH_SUM* theSum,
    BYTE* data,
    WORD len
);
```

**Description**

Adds data to a SHA-1 hash calculation.

**Preconditions**

The hash context has already been initialized.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theSum</td>
<td>a pointer to the hash context structure</td>
</tr>
<tr>
<td>data</td>
<td>the data to add to the hash</td>
</tr>
<tr>
<td>len</td>
<td>the length of the data to add</td>
</tr>
</tbody>
</table>

**Returns**

None
MD5AddData Function

C

```c
void MD5AddData(
    HASH_SUM* theSum,
    BYTE* data,
    WORD len
);
```

Description

Adds data to an MD5 hash calculation.

Preconditions

The hash context has already been initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theSum</td>
<td>a pointer to the hash context structure</td>
</tr>
<tr>
<td>data</td>
<td>the data to add to the hash</td>
</tr>
<tr>
<td>len</td>
<td>the length of the data to add</td>
</tr>
</tbody>
</table>

Returns

None
The following functions and variables are designated as internal to the Hashes module.

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASH_TYPE</td>
<td>Type of hash being calculated</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA1HashBlock</td>
<td>Calculates the SHA-1 hash sum of a block.</td>
</tr>
<tr>
<td>MD5HashBlock</td>
<td>Calculates the MD5 hash sum of a block.</td>
</tr>
</tbody>
</table>

### Module

Hashes

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_MD5_k</td>
<td>Array of pre-defined K values for MD5</td>
</tr>
<tr>
<td>_MD5_r</td>
<td>Array of pre-defined R values for MD5</td>
</tr>
<tr>
<td>lastBlock</td>
<td>Stores a copy of the last block with the required padding</td>
</tr>
</tbody>
</table>
_MD5_k Variable

```c
ROM DWORD _MD5_k[64] = { 0xD76AA478, 0xE8C7B756, 0x242070DB, 0xC1BDCEE4, 0x103C0C26, 0x1C62C689, 0x1E19B213, 0x6128865B, 0x6701F08D, 0x6EF85299, 0x81CF1088, 0x8BEDE12F, 0x8EBBD476, 0x8F0BBFE0, 0x978FD431, 0x9B04C3D5, 0x9E377257, 0xA0096C42, 0xA65A6338, 0xA8077C86, 0xAE9A8B96, 0xBE1631BD, 0xC1999BDA, 0xC49CDBB2, 0xC0FDDF5D, 0xC58F5CF0, 0 Cd ... D, 0x85845DD1, 0x6FA87E4F, 0xFE2CE6E0, 0xA3014314, 0x4E0811A1, 0xF7537E82, 0xBD3AF235, 0x2AD7D2BB, 0xEB86D391
```

Description

Array of pre-defined K values for MD5
_MD5_r Variable

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, ... 6, 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

Stack API > Hashes > Internal Members > _MD5_r Variable
## lastBlock Variable

```c
BYTE lastBlock[64];
```

### Description

Stores a copy of the last block with the required padding
HASH_TYPE Enumeration

C

typedef enum {
    HASH_MD5 = 0u,
    HASH_SHA1
} HASH_TYPE;

Description

Type of hash being calculated

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASH_MD5 = 0u</td>
<td>MD5 is being calculated</td>
</tr>
<tr>
<td>HASH_SHA1</td>
<td>SHA-1 is being calculated</td>
</tr>
</tbody>
</table>

Stack API > Hashes > Internal Members > HASH_TYPE Enumeration
SHA1HashBlock Function

```c
static void SHA1HashBlock(
    BYTE* data,
    DWORD* h0,
    DWORD* h1,
    DWORD* h2,
    DWORD* h3,
    DWORD* h4
);
```

**Description**

This function calculates the SHA-1 hash sum over a block and updates the values of h0-h3 with the next context.

**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**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>The block of 64 bytes to hash</td>
</tr>
<tr>
<td>h0</td>
<td>the current hash context h0 value</td>
</tr>
<tr>
<td>h1</td>
<td>the current hash context h1 value</td>
</tr>
<tr>
<td>h2</td>
<td>the current hash context h2 value</td>
</tr>
<tr>
<td>h3</td>
<td>the current hash context h3 value</td>
</tr>
<tr>
<td>h4</td>
<td>the current hash context h4 value</td>
</tr>
</tbody>
</table>

**Returns**
None
MD5HashBlock Function

C

```c
static void MD5HashBlock(
    BYTE* data,
    DWORD* h0,
    DWORD* h1,
    DWORD* h2,
    DWORD* h3
);
```

Description

This function calculates the MD5 hash sum over a block and updates the values of h0-h3 with the next context.

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>The block of 64 bytes to hash</td>
</tr>
<tr>
<td>h0</td>
<td>the current hash context h0 value</td>
</tr>
<tr>
<td>h1</td>
<td>the current hash context h1 value</td>
</tr>
<tr>
<td>h2</td>
<td>the current hash context h2 value</td>
</tr>
<tr>
<td>h3</td>
<td>the current hash context h3 value</td>
</tr>
</tbody>
</table>

Returns

None
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.

### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helpers Public Members</td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td>Functions</td>
<td>The following table lists functions in this documentation.</td>
</tr>
<tr>
<td>Variables</td>
<td>The following table lists variables in this documentation.</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFSRRand</td>
<td>Returns a pseudo-random 16-bit unsigned integer in the range from 0 to 65535 (0x0000 to 0xFFFF).</td>
</tr>
<tr>
<td>LFSRSeedRand</td>
<td>Seeds the LFSR random number generator invoked by the LFSRRand() function. The prior seed is returned.</td>
</tr>
<tr>
<td>strncpy_m</td>
<td>Copies multiple strings to a destination</td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwLFSRRandSeed</td>
<td>Default Random Number Generator seed. 0x41FE9F9E corresponds to calling LFSRSeedRand(1)</td>
</tr>
</tbody>
</table>
The following functions and variables are available to the stack application.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base64Decode</td>
<td>Decodes a Base-64 array to its literal representation.</td>
</tr>
<tr>
<td>Base64Encode</td>
<td>Encodes a binary array to Base-64.</td>
</tr>
<tr>
<td>btohexa_high</td>
<td>Converts the upper nibble of a binary value to a hexadecimal ASCII byte.</td>
</tr>
<tr>
<td>btohexa_low</td>
<td>Converts the lower nibble of a binary value to a hexadecimal ASCII byte.</td>
</tr>
<tr>
<td>CalcIPChecksum</td>
<td>Calculates an IP checksum value.</td>
</tr>
<tr>
<td>ExtractURLFields</td>
<td>Extracts all parameters from an URL string (ex:</td>
</tr>
<tr>
<td></td>
<td>&quot;<a href="http://admin:passwd@www.microchip.com:8080/myfile.gif">http://admin:passwd@www.microchip.com:8080/myfile.gif</a>&quot; is split into {</td>
</tr>
<tr>
<td></td>
<td>PROTOCOL_HTTP, &quot;admin&quot;, &quot;passwd&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;www.microchip.com&quot;, 8080, &quot;/myfile.gif&quot;}.</td>
</tr>
<tr>
<td>FormatNetBIOSName</td>
<td>Formats a string to a valid NetBIOS name.</td>
</tr>
<tr>
<td>GenerateRandomDWORD</td>
<td>Generates a random DWORD.</td>
</tr>
<tr>
<td>hexatob</td>
<td>Converts a hex string to a single byte.</td>
</tr>
<tr>
<td>leftRotateDWORD</td>
<td>Left-rotates a DWORD.</td>
</tr>
<tr>
<td>Replace</td>
<td>Replaces all instances of a particular substring with a new string.</td>
</tr>
<tr>
<td>ROMStringToIPAddress</td>
<td>Converts a string to an IP address</td>
</tr>
<tr>
<td>stricmppgm2ram</td>
<td>Case-insensitive comparison of a string in RAM to a string in ROM.</td>
</tr>
<tr>
<td>StringToIPAddress</td>
<td>Converts a string to an IP address.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>strupr</code></td>
<td>Converts a string to uppercase.</td>
</tr>
<tr>
<td><code>strnchr</code></td>
<td>Searches a string up to a specified number of characters for a specific character.</td>
</tr>
<tr>
<td><code>swapl</code></td>
<td>This is function <code>swaps</code>.</td>
</tr>
<tr>
<td><code>swaps</code></td>
<td>Swaps the endian-ness of a WORD.</td>
</tr>
<tr>
<td><code>uitoa</code></td>
<td>Converts an unsigned integer to a decimal string.</td>
</tr>
<tr>
<td><code>ultoa</code></td>
<td>Converts an unsigned integer to a decimal string.</td>
</tr>
<tr>
<td><code>UnencodeURL</code></td>
<td>Decodes a URL-encoded string.</td>
</tr>
</tbody>
</table>

## Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>leftRotateDWORD</code></td>
<td>Rotations are more efficient in C30 and C32</td>
</tr>
<tr>
<td><code>ROMStringToIPAddress</code></td>
<td>Non-ROM variant for C30 and C32</td>
</tr>
</tbody>
</table>

## Module

### Helpers

[Stack API] > [Helpers] > [Public Members]

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
## Base64Decode Function

```c
WORD Base64Decode(
    BYTE* cSourceData,
    WORD wSourceLen,
    BYTE* cDestData,
    WORD wDestLen
);
```

### Description

Decodes a Base-64 array to its literal representation.

### Preconditions

None

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cSourceData</td>
<td>Pointer to a string of Base-64 encoded data</td>
</tr>
<tr>
<td>wSourceLen</td>
<td>Length of the Base-64 source data Maximum length that can be written to cDestData</td>
</tr>
<tr>
<td>cDestData</td>
<td>Pointer to write the decoded data</td>
</tr>
</tbody>
</table>

### Returns

Number of decoded bytes written to cDestData.

### 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.
Base64Encode Function

C

WORD Base64Encode(
    BYTE* cSourceData,
    WORD wSourceLen,
    BYTE* cDestData,
    WORD wDestLen
);  

Description

Encodes a binary array to Base-64.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cSourceData</td>
<td>Pointer to a string of binary data</td>
</tr>
<tr>
<td>wSourceLen</td>
<td>Length of the binary source data Maximum length that can be written to cDestData</td>
</tr>
<tr>
<td>cDestData</td>
<td>Pointer to write the Base-64 encoded data</td>
</tr>
</tbody>
</table>

Returns

Number of encoded bytes written to cDestData. This will always be a multiple of 4.

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.
btohexa_high Function

C

BYTE btohexa_high(
    BYTE b
);

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>the byte to convert</td>
</tr>
</tbody>
</table>

Returns

The upper hexadecimal ASCII byte '0'-'9' or 'A'-'F'.

btohexa_low Function

```c
BYTE btohexa_low(BYTE b);
```

Description

Converts the lower nibble of a binary value to a hexadecimal ASCII byte. For example, `btohexa_high(0xAE)` will return 'E'.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>the byte to convert</td>
</tr>
</tbody>
</table>

Returns

The lower hexadecimal ASCII byte '0'-'9' or 'A'-'F'.

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
CalcIPChecksum Function

C

```c
WORD CalcIPChecksum(
    BYTE* buffer,
    WORD len
);
```

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.

Preconditions

buffer is WORD aligned (even memory address) on 16- and 32-bit PICs.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>pointer to the data to be checksummed</td>
</tr>
<tr>
<td>count</td>
<td>number of bytes to be checksummed</td>
</tr>
</tbody>
</table>

Returns

The calculated checksum.

Stack API > Helpers > Public Members > CalcIPChecksum Function
ExtractURLFields Function

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
);
```

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
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vURL</td>
<td>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).</td>
</tr>
<tr>
<td>protocol</td>
<td>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. For the example URL provided in the function description, PROTOCOL_HTTP would be returned for this field.</td>
</tr>
<tr>
<td>vUsername</td>
<td>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. For the example URL provided in the function description, &quot;admin&quot; would be returned for this field.</td>
</tr>
<tr>
<td>wUsernameLen</td>
<td>On call: Optional pointer to a WORD specifying the maximum length of the vUsername buffer, including the null terminator character. 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. 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: &quot;&quot;). For the example URL provided in the function description, 6 (0x0006) would be returned for this field.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>vPassword</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>For the example URL provided in the function description, &quot;passwd&quot; would be returned for this field.</td>
</tr>
</tbody>
</table>
| wPasswordLen | On call: Optional pointer to a WORD specifying the maximum length of the vPassword buffer, including the null terminator character.  
|              | 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.  
|              | 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: "").  
|              | For the example URL provided in the function description, 7 (0x0007) would be returned for this field.                                                                                                     |
| vHostname    | 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.           |
|              | 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() helper function to do this). |
| wHostnameLen | On call: Optional pointer to a WORD specifying the maximum length of the vHostname buffer, including the null terminator character.  
|              | Upon return: If wHostnameLen and vHostname are non-NULL, the *wHostnameLen WORD is updated with the                                                                   |
| **wHostnameLen** | 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.

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).

| **wPort** | 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.

If the wPort pointer is NULL, then the port field in the URL is ignored, if present.

| **vFilePath** | 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.

For the example URL provided in the function description, "/myfile.gif" would be returned for this field.

| **wFilePathLen** | On call: Optional pointer to a WORD specifying the maximum length of the vFilePath buffer, including the null terminator character.

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.

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.
For the example URL provided in the function description, 12 (0x000C) would be returned for this field.

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.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>1</td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td>URL malformed. Illegal or unknown URL format encountered.</td>
</tr>
<tr>
<td>3</td>
<td>Buffer too small. One of the input buffer sizes is too small to contain the URL parameter.</td>
</tr>
</tbody>
</table>
FormatNetBIOSName Function

```c
void FormatNetBIOSName(
    BYTE Name[16]
);
```

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>the string to format as a NetBIOS name. This parameter must have at least 16 bytes allocated.</td>
</tr>
</tbody>
</table>

Returns

None
GenerateRandomDWORD Function

C
DWORD GenerateRandomDWORD();

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() / LFSRRand() functions, the output is normalized (deskewed) in the hopes of meeting statistical randomness tests.

Preconditions

None

Returns

Random 32-bit number.

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() 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.

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.
hexatob Function

```c
BYTE hexatob(
    WORD_VAL AsciiChars
);
```

Description

Converts a two-character ASCII hex string to a single packed byte.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsciiChars</td>
<td>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'.</td>
</tr>
</tbody>
</table>

Returns

Resulting packed byte 0x00 - 0xFF.
leftRotateDWORD Function

C

```
DWORD leftRotateDWORD(
    DWORD val,
    BYTE bits
);
```

Description

This function rotates the bits in a 32-bit DWORD left by a specific number of bits.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>the DWORD to be rotated</td>
</tr>
<tr>
<td>bits</td>
<td>the number of bits by which to shift</td>
</tr>
</tbody>
</table>

Returns

Rotated DWORD value.

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.
leftRotateDWORD Macro

```c
#define leftRotateDWORD(x, n) (((x) << (n)) | ((x) >> (32-(n))))
```

Description

Rotations are more efficient in C30 and C32
Replace Function

C

SHORT Replace(
    BYTE * vExpression,
    ROM BYTE * vFind,
    ROM BYTE * vReplacement,
    WORD wMaxLen,
    BOOL bSearchCaseInsensitive
);  

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.

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vExpression</td>
<td>Null terminated string to search and make replacements within.</td>
</tr>
<tr>
<td>vFind</td>
<td>Null terminated string to search for.</td>
</tr>
<tr>
<td>vReplacement</td>
<td>Null terminated string to replace all instances of vFind with.</td>
</tr>
<tr>
<td>wMaxLen</td>
<td>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</td>
</tr>
<tr>
<td><strong>wMaxLen</strong></td>
<td>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.</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>bSearchCaseInsensitive</strong></td>
<td>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).</td>
</tr>
</tbody>
</table>

**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.

**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".
ROMStringToIPAddress Function

```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.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Pointer to a dotted-quad IP address string</td>
</tr>
<tr>
<td>IPAddress</td>
<td>Pointer to IP_ADDR in which to store the result</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>an IP address was successfully decoded</td>
</tr>
<tr>
<td>FALSE</td>
<td>no IP address could be found, or the format was incorrect</td>
</tr>
</tbody>
</table>

Remarks

This function is aliased to `StringToIPAddress` on non-PIC18 platforms.
ROMStringToIPAddress Macro

C

#define ROMStringToIPAddress(a,b) StringToIPAddress((BYTE*)a,b)

Description

Non-ROM variant for C30 and C32
stricmppgm2ram Function

C

```c
signed char stricmppgm2ram(
    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 strcmppgm2ram, except that the comparison is not case-sensitive.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Pointer to string in RAM</td>
</tr>
<tr>
<td>b</td>
<td>Pointer to string in ROM</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>a &lt; b</td>
</tr>
<tr>
<td>0</td>
<td>a = b</td>
</tr>
<tr>
<td>1</td>
<td>a &gt; b</td>
</tr>
</tbody>
</table>
StringToIPAddress Function

C

```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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Pointer to a dotted-quad IP address string</td>
</tr>
<tr>
<td>IPAddress</td>
<td>Pointer to IP_ADDR in which to store the result</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>an IP address was successfully decoded</td>
</tr>
<tr>
<td>FALSE</td>
<td>no IP address could be found, or the format was incorrect</td>
</tr>
</tbody>
</table>
strupr Function

```c
char *strupr(char* s);
```

**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**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>the null-terminated string to be converted.</td>
</tr>
</tbody>
</table>

**Returns**

Pointer to the initial string.
strnchr Function

```
c
    char * strnchr(
            const char * searchString,
            size_t count,
            char c
    );
```

**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**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchString</td>
<td>Pointer to a null terminated string to search. If count is less than the string size, then the string need not be null terminated.</td>
</tr>
<tr>
<td>count</td>
<td>Maximum number of characters to search before aborting.</td>
</tr>
<tr>
<td>c</td>
<td>Character to search for</td>
</tr>
</tbody>
</table>

**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.
swapl Function

```
C

DWORD swapl(
    DWORD v
);
```

Description

This is function swapl.
swaps Function

C

WORD swaps(
    WORD v
);

Description

Swaps the endian-ness of a WORD.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>the WORD to swap</td>
</tr>
</tbody>
</table>

Returns

The swapped version of v.
uitoa Function

C

```c
void uitoa(
    WORD Value,
    BYTE* Buffer
);
```

Description

Converts a 16-bit unsigned integer to a null-terminated decimal string.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The number to be converted</td>
</tr>
<tr>
<td>Buffer</td>
<td>Pointer in which to store the converted string</td>
</tr>
</tbody>
</table>

Returns

None
ultoa Function

C

```c
void ultoa(
    DWORD Value,
    BYTE* Buffer
);
```

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The number to be converted</td>
</tr>
<tr>
<td>Buffer</td>
<td>Pointer in which to store the converted string</td>
</tr>
</tbody>
</table>

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

Stack API > Helpers > Public Members > ultoa Function
UnencodeURL Function

```c
void UnencodeURL( BYTE* URL );
```

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td>the null-terminated string to decode</td>
</tr>
</tbody>
</table>

Returns

None

Stack API > Helpers > Public Members > UnencodeURL Function
## Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFSRRand</td>
<td>Returns a pseudo-random 16-bit unsigned integer in the range from 0 to 65535 (0x0000 to 0xFFFF).</td>
</tr>
<tr>
<td>LFSRSeedRand</td>
<td>Seeds the LFSR random number generator invoked by the LFSRRand() function. The prior seed is returned.</td>
</tr>
<tr>
<td>strncpy_m</td>
<td>Copies multiple strings to a destination</td>
</tr>
</tbody>
</table>

## Module

**Helpers**

Stack API > Helpers > Functions
LFSRRand Function

C

```c
WORD LFSRRand();
```

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() 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.

Preconditions

None

Returns

Random 16-bit unsigned integer.

Side Effects

The internal LFSR seed is updated so that the next call to LFSRRand() will return a different random number.

Remarks

None

Stack API > Helpers > Functions > LFSRRand Function
LFSRSeedRand Function

C

```c
DWORD LFSRSeedRand(
    DWORD dwSeed
);
```

Description

 Seeds the LFSR random number generator invoked by the `LFSRRand()` function. The prior seed is returned.

Preconditions

 None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wSeed</td>
<td>The new 32-bit seed value to assign to the LFSR.</td>
</tr>
</tbody>
</table>

Returns

 The last seed in use. This can be saved and restored by a subsequent call to `LFSRSeedRand()` if you wish to use `LFSRRand()` 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()` varies.

Side Effects

 None
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.
strncpy_m Function

```c
size_t strncpy_m(
    char* destStr,
    size_t destSize,
    int nStrings,
    ...
);
```

Description

Copies multiple strings to a destination but doesn't copy more than destSize characters. Useful where the destination is actually an array and an extra 0 won't be appended to overflow the buffer.

Preconditions

- valid string pointers
- destSize should be > 0

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destStr</td>
<td>Pointer to a string to be initialized with the multiple strings provided as arguments.</td>
</tr>
<tr>
<td>destSize</td>
<td>the maximum size of the destStr field, that cannot be exceeded. An 0 won't be appended if the resulting size is &gt; destSize</td>
</tr>
<tr>
<td>nStrings</td>
<td>number of string parameters to be copied into destStr</td>
</tr>
<tr>
<td>...</td>
<td>variable number of arguments</td>
</tr>
</tbody>
</table>

Returns
Length of the destination string, terminating 0 (if exists) not included
# Variables

## Module

### Helpers

## Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dwLFSRRandSeed</code></td>
<td>Default Random Number Generator seed. 0x41FE9F9E corresponds to calling <code>LFSRSeedRand</code>(1)</td>
</tr>
</tbody>
</table>

[Stack API] > [Helpers] > [Variables]
**Description**

Default Random Number Generator seed. 0x41FE9F9E corresponds to calling [LFSRSeedRand](1)
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 callbacks), parses data submitted through forms (in HTTPExecuteGet and HTTPExecutePost) 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 callback functions defined in CustomHTTPApp.c. This file does not require modification by the programmer.

**Topics**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP2 Features</td>
<td>Features available in the HTTP2 module</td>
</tr>
<tr>
<td>HTTP2 Public Members</td>
<td>Functions and variables accessible or implemented by the stack application</td>
</tr>
<tr>
<td>HTTP2 Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td>HTTP2 Internal Members</td>
<td>Functions and variables used internally by the HTTP2 module</td>
</tr>
</tbody>
</table>
HTTP2 Features

The HTTP2 web server module has many capabilities. The following topics will introduce these features and provide examples.

Module

HTTP2 Server

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP2 Dynamic Variables</td>
<td>Displays real-time data in web pages or other outputs</td>
</tr>
<tr>
<td>HTTP2 Form Processing</td>
<td>Allows client input from HTML forms</td>
</tr>
<tr>
<td>HTTP2 Authentication</td>
<td>Verifies user names and passwords for access</td>
</tr>
<tr>
<td>HTTP2 Cookies</td>
<td>Allows cookies to be set/retrieved, adding state to the HTTP protocol</td>
</tr>
<tr>
<td>HTTP2 Compression</td>
<td>Compresses static files for faster throughput</td>
</tr>
</tbody>
</table>

Stack API > HTTP2 Server > Features
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:

```html
Copy Code
<div class="examplebox code">~builddate~</div>
```

The associated callback will print the value into the web page:

```c
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:

```html
<button>btn(3)</button> <button>btn(2)</button> <button>btn(1)</button> <button>btn(0)</button>
```

This associated callback will print the value of the requested button to the web page:

```c
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` 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:

```html
<div class="examplebox code">~lcdtext~</div>
```

The following callback function handles the output, and manages its state for multiple calls:

```c
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;
```
The initial call to `TCPIsPutReady` determines how many bytes can be written to the buffer right now. The `TCPPut` 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:
At this time, dynamic variables are not recursive, so any variables located inside files included in this manner are not parsed.
HTTP2 Form Processing

Many applications need to accept 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 is implemented by the application developer to process this data and perform any necessary actions. The functions HTTPGetArg and HTTPGetROMArg provide an easy method to retrieve submitted values for processing.

The following example demonstrates a form to control several LEDs.

```html
<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:

```plaintext
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` to process this input. To process this data, that callback needs to do several things. First, it should call `MPFSGetFilename` 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');
    }
}
```
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` and `HTTPReadPostValue` 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.

```html
<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" /></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** function will be called with the following data still in the TCP buffer:

```
Copy Code
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** and **HTTPReadPostValue** 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.

```
Copy Code
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(!strcmppgm2ram((char*)temp, (ROM char*)"to"))
    dest = curHTTP.data;

// Save "subject" values to curHTTP.data[50:99]
else if(!strcmppgm2ram((char*)temp, (ROM char*)"subject")
    dest = curHTTP.data + 50;

// When a "msg" is encountered, start sending
else if(!strcmppgm2ram((char*)temp, (ROM char*)"msg"))
{
    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 me

    // 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 hol
    // at once. To avoid errors, read the data piece by
    // piece and send it to the e-mail module. This require
    // 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 and HTTPReadPostValue functions, and also demonstrates using the need to use TCPIsGetReady, TCPGet, and TCPGetArray 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 to accomplish the callback. The HTTPPostLCD 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.
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** functionality is supported by two user-provided callback functions. The first, `HTTPNeedsAuth`, determines if the requested page requires valid credentials to proceed. The second, `HTTPCheckAuth`, 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` 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:

```c
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:

```c
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(!memcmp PROGMEM2ram((void*)cFile, (ROM void*)"treasure", 8))
        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:

```c
#define HTTP_AUTH_ADMIN 0x00
#define HTTP_AUTH_OTHER 0x01

BYTE HTTPNeedsAuth(BYTE* cFile)
{
    // Return a specific code for admin users
    if(!memcmp PROGMEM2ram((void*)cFile, (ROM void*)"admin", 5))
        return HTTP_AUTH_ADMIN;

    return HTTP_AUTH_OTHER;
}
```

**Validating Credentials**

The `HTTPCheckAuth` 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 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 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.
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 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 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 callback using the HTTPGetArg or HTTPGetROMArg 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) 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 can be set in HTTPExecuteGet or HTTPExecutePost. 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:

```c
void HTTPExecuteGet(void)
{
    ...

    // Set a cookie
    strcpypgm2ram((char*)curHTTP.data, (ROM char*)"flavor\oatmeal r"
    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".
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.
The following functions and variables are accessible or implemented by the stack application.

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_IO_RESULT</td>
<td>Result states for execution callbacks</td>
</tr>
<tr>
<td>HTTP_READ_STATUS</td>
<td>Result states for HTTPPostReadName and HTTPPostReadValue</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPCheckAuth</td>
<td>Performs validation on a specific user name and password.</td>
</tr>
<tr>
<td>HTTPExecuteGet</td>
<td>Processes GET form field variables and cookies.</td>
</tr>
<tr>
<td>HTTPExecutePost</td>
<td>Processes POST form variables and data.</td>
</tr>
<tr>
<td>HTTPGetArg</td>
<td>Locates a form field value in a given data array.</td>
</tr>
<tr>
<td>HTTPGetROMArg</td>
<td>Locates a form field value in a given data array.</td>
</tr>
<tr>
<td>HTTPNeedsAuth</td>
<td>Determines if a given file name requires authentication</td>
</tr>
<tr>
<td>HTTPPrint_varname</td>
<td>Inserts dynamic content into a web page</td>
</tr>
<tr>
<td>HTTPReadPostName</td>
<td>Reads a name from a URL encoded string in the TCP buffer.</td>
</tr>
<tr>
<td>HTTPReadPostValue</td>
<td>Reads a value from a URL encoded string in the TCP buffer.</td>
</tr>
<tr>
<td>HTTPURLDecode</td>
<td>Parses a string from URL encoding to plain-text.</td>
</tr>
</tbody>
</table>
Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPReadPostPair</td>
<td>Reads a name and value pair from a URL encoded string in the TCP buffer.</td>
</tr>
<tr>
<td>sktHTTP</td>
<td>Access the current socket</td>
</tr>
</tbody>
</table>

Module

HTTP2 Server

Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_CONN</td>
<td>Stores extended state data for each connection</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curHTTP</td>
<td>Current HTTP connection state</td>
</tr>
</tbody>
</table>

Stack API > HTTP2 Server > Public Members

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
### curHTTP Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>HTTP_CONN curHTTP;</code></td>
</tr>
</tbody>
</table>

**Description**

Current HTTP connection state

[Stack API] > [HTTP2 Server] > [Public Members] > [curHTTP Variable]
HTTP_CONN Structure

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;

Description

Stores extended state data for each connection

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD byteCount;</td>
<td>How many bytes have been read so far</td>
</tr>
<tr>
<td>DWORD nextCallback;</td>
<td>Byte index of the next callback</td>
</tr>
<tr>
<td>DWORD callbackID;</td>
<td>Callback ID to execute, also used as watchdog timer</td>
</tr>
<tr>
<td>DWORD callbackPos;</td>
<td>Callback position indicator</td>
</tr>
<tr>
<td>BYTE * ptrData;</td>
<td>Points to first free byte in data</td>
</tr>
<tr>
<td>BYTE * ptrRead;</td>
<td>Points to current read location</td>
</tr>
<tr>
<td>MPFS_HANDLE file;</td>
<td>File pointer for the file being served</td>
</tr>
<tr>
<td>MPFS_HANDLE offsets;</td>
<td>File pointer for any offset info being used</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>BYTE hasArgs;</td>
<td>True if there were get or cookie arguments</td>
</tr>
<tr>
<td>BYTE isAuthorized;</td>
<td>0x00-0x79 on fail, 0x80-0xff on pass</td>
</tr>
<tr>
<td>HTTP_STATUS httpStatus;</td>
<td>Request method/status</td>
</tr>
<tr>
<td>HTTP_FILE_TYPE fileType;</td>
<td>File type to return with Content-Type</td>
</tr>
<tr>
<td>BYTE data[HTTP_MAX_DATA_LEN];</td>
<td>General purpose data buffer</td>
</tr>
<tr>
<td>BYTE smPost;</td>
<td>POST state machine variable</td>
</tr>
</tbody>
</table>

Stack API > HTTP2 Server > Public Members > HTTP_CONN Structure
HTTP_IO_RESULT Enumeration

C

typedef enum {
    HTTP_IO_DONE = 0u,
    HTTP_IO_NEED_DATA,
    HTTP_IO_WAITING
} HTTP_IO_RESULT;

Description

Result states for execution callbacks

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_IO_DONE = 0u</td>
<td>Finished with procedure</td>
</tr>
<tr>
<td>HTTP_IO_NEED_DATA</td>
<td>More data needed to continue, call again later</td>
</tr>
<tr>
<td>HTTP_IO_WAITING</td>
<td>Waiting for asynchronous process to complete, call again later</td>
</tr>
</tbody>
</table>

Stack API > HTTP2 Server > Public Members > HTTP_IO_RESULT Enumeration
## HTTP_READ_STATUS Enumeration

```c
typedef enum {
    HTTP_READ_OK = 0u,
    HTTP_READ_TRUNCATED,
    HTTP_READ_INCOMPLETE
} HTTP_READ_STATUS;
```

### Description

Result states for HTTPPostReadName and HTTPPostReadValue

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_READ_OK = 0u</td>
<td>Read was successful</td>
</tr>
<tr>
<td>HTTP_READ_TRUNCATED</td>
<td>Buffer overflow prevented by truncating value</td>
</tr>
<tr>
<td>HTTP_READ_INCOMPLETE</td>
<td>Entire object is not yet in the buffer. Try again later.</td>
</tr>
</tbody>
</table>

Related Links:
- [Stack API](#)
- [HTTP2 Server](#)
- [Public Members](#)
- [HTTP_READ_STATUS Enumeration](#)
HTTPCheckAuth Function

```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, HTTPExecutePost, or HTTPPrint_varname callbacks.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 0x79</td>
<td>the credentials were rejected</td>
</tr>
<tr>
<td>&gt;= 0x80</td>
<td>access is granted for this connection</td>
</tr>
</tbody>
</table>

### Remarks

This function is only called when an Authorization header is encountered.

This function may NOT write to the TCP buffer.
HTTPExecuteGet Function

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 and HTTPGetROMArg 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 and any HTTPPrint_varname 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.

Preconditions

None
### Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_IO_DONE</td>
<td>application is done processing</td>
</tr>
<tr>
<td>HTTP_IO_NEED_DATA</td>
<td>this value may not be returned because more data will not become available</td>
</tr>
<tr>
<td>HTTP_IO_WAITING</td>
<td>the application is waiting for an asynchronous process to complete, and this function should be called again later</td>
</tr>
</tbody>
</table>

### 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.
HTTPExecutePost Function

```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` or `TCPGetArray` in order to retrieve bytes. When this is done, `curHTTP.byteCount` MUST be updated to reflect how many bytes now remain. The functions `TCPFind`, `TCPFindString`, `TCPFindROMString`, `TCPFindArray`, and `TCPFindROMArray` may be helpful to locate data in the TCP buffer.

In general, data submitted from web forms via POST is URL encoded. The `HTTPURLDecode` 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` 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.

Preconditions

None

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_IO_DONE</td>
<td>application is done processing</td>
</tr>
<tr>
<td>HTTP_IO_NEED_DATA</td>
<td>more data is needed to continue, and this function should be called again later</td>
</tr>
<tr>
<td>HTTP_IO_WAITING</td>
<td>the application is waiting for an asynchronous process to complete, and this function should be called again later</td>
</tr>
</tbody>
</table>

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.
HTTPGetArg Function

C

```c
BYTE* HTTPGetArg(
    BYTE* cData,
    BYTE* cArg
);
```

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>the buffer to search</td>
</tr>
<tr>
<td>arg</td>
<td>the name of the argument to find</td>
</tr>
</tbody>
</table>

Returns

A pointer to the argument value, or NULL if not found.

Stack API > HTTP2 Server > Public Members > HTTPGetArg Function
HTTPGetROMArg Function

```
C
BYTE* HTTPGetROMArg(
    BYTE* cData,
    ROM BYTE* cArg
);
```

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>the buffer to search</td>
</tr>
<tr>
<td>arg</td>
<td>the name of the argument to find</td>
</tr>
</tbody>
</table>

Returns

A pointer to the argument value, or NULL if not found.

Remarks

This function is aliased to `HTTPGetArg` on non-PIC18 platforms.
HTTPNeedsAuth Function

```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 and any of the HTTPExecuteGet, HTTPExecutePost, or HTTPPrint_varname callbacks.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cFile</td>
<td>the name of the file being requested</td>
</tr>
</tbody>
</table>
Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 0x79</td>
<td>valid authentication is required</td>
</tr>
<tr>
<td>&gt;= 0x80</td>
<td>access is granted for this connection</td>
</tr>
</tbody>
</table>

Remarks

This function may NOT write to the TCP buffer.
HTTPPrint_varname Function

```c
void HTTPPrint_varname(
    WORD wParam1,
    WORD wParam2,
    ...
);
```

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`, `TCPPut`, `TCPPutArray`, `TCPPutString`, `TCPPutROMArray`, and `TCPPutROMString`.

Before calling, the HTTP2 server guarantees that at least `HTTP_MIN_CALLBACK_FREE` 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.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wParam1</td>
<td>first parameter passed in the dynamic variable (if any)</td>
</tr>
<tr>
<td>wParam2</td>
<td>second parameter passed in the dynamic variable (if any)</td>
</tr>
<tr>
<td>...</td>
<td>additional parameters as necessary</td>
</tr>
</tbody>
</table>

**Returns**

None

**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.
HTTPReadPostName Function

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 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cData</td>
<td>where to store the name once it is read</td>
</tr>
<tr>
<td>wLen</td>
<td>how many bytes can be written to cData</td>
</tr>
</tbody>
</table>
## Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_READ_OK</td>
<td>name was successfully read</td>
</tr>
<tr>
<td>HTTP_READ_TRUNCATED</td>
<td>entire name could not fit in the buffer, so the value was truncated and data has been lost</td>
</tr>
<tr>
<td>HTTP_READ_INCOMPLETE</td>
<td>entire name was not yet in the buffer, so call this function again later to retrieve</td>
</tr>
</tbody>
</table>
HTTPReadPostPair Macro

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 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.

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cData, wLen</td>
<td>where to store the name and value strings once they are read</td>
</tr>
<tr>
<td>Return Values</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>HTTP_READ_OK</td>
<td>name and value were successfully read</td>
</tr>
<tr>
<td>HTTP_READ_TRUNCATED</td>
<td>entire name and value could not fit in the buffer, so input was truncated and data has been lost</td>
</tr>
<tr>
<td>HTTP_READ_INCOMPLETE</td>
<td>entire name and value was not yet in the buffer, so call this function again later to retrieve</td>
</tr>
</tbody>
</table>

**Remarks**

This function is aliased to HTTPReadPostValue, since they effectively perform the same task. The name is provided only for completeness.
HTTPReadPostValue Function

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 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cData</td>
<td>where to store the value once it is read</td>
</tr>
<tr>
<td>Return Values</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HTTP_READ_OK</td>
<td>value was successfully read</td>
</tr>
<tr>
<td>HTTP_READ_TRUNCATED</td>
<td>entire value could not fit in the buffer, so the value was truncated and data has been lost</td>
</tr>
<tr>
<td>HTTP_READ_INCOMPLETE</td>
<td>entire value was not yet in the buffer, so call this function again later to retrieve</td>
</tr>
</tbody>
</table>
HTTPURLDecode Function

C

BYTE* HTTPURLDecode(
    BYTE* cData
);  

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.

Preconditions

The data parameter is null terminated and has at least one extra byte free.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cData</td>
<td>The string which is to be decoded in place.</td>
</tr>
</tbody>
</table>

Returns

A pointer to the last null terminator in data, which is also the first free byte for new 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.
**sktHTTP Macro**

```c
#define sktHTTP httpStubs[curHTTPID].socket  // Access the current socket
```

**Description**

Access the current socket

[Stack API > HTTP2 Server > Public Members > sktHTTP Macro](#)
HTTP2 Stack Members

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.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPInit</td>
<td>Initializes the HTTP server module.</td>
</tr>
<tr>
<td>HTTPServer</td>
<td>Performs periodic tasks for the HTTP2 module.</td>
</tr>
</tbody>
</table>

Module

HTTP2 Server

Stack API > HTTP2 Server > Stack Members
HTTPInit Function

C

void HTTPInit();

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.

Preconditions

TCP must already be initialized.

Returns

None

Remarks

This function is called only one during lifetime of the application.
HTTPServer Function

```c
void HTTPServer();
```

Description

Browses through each open connection and attempts to process any pending operations.

Preconditions

HTTPInit() must already be called.

Returns

None

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.
HTTP2 Internal Members

The following functions and variables are designated as internal to the HTTP2 module.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_FILE_TYPE</td>
<td>File type definitions</td>
</tr>
<tr>
<td>HTTP_STATUS</td>
<td>Supported Commands and Server Response Codes</td>
</tr>
<tr>
<td>SM_HTTP2</td>
<td>Basic HTTP Connection State Machine</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPHeaderParseAuthorization</td>
<td>Parses the &quot;Authorization:&quot; header for a request and verifies the credentials.</td>
</tr>
<tr>
<td>HTTPHeaderParseContentLength</td>
<td>Parses the &quot;Content-Length:&quot; header for a request.</td>
</tr>
<tr>
<td>HTTPHeaderParseCookie</td>
<td>Parses the &quot;Cookie:&quot; headers for a request and stores them as GET variables.</td>
</tr>
<tr>
<td>HTTPHeaderParseLookup</td>
<td>Calls the appropriate header parser based on the index of the header that was read from the request.</td>
</tr>
<tr>
<td>HTTPIncFile</td>
<td>Writes a file byte-for-byte to the currently loaded TCP socket.</td>
</tr>
<tr>
<td>HTTPLoadConn</td>
<td>Switches the currently loaded connection for the HTTP2 module.</td>
</tr>
<tr>
<td>HTTPTPFSUpload</td>
<td>Saves a file uploaded via POST as the new MPFS image in EEPROM or external Flash.</td>
</tr>
<tr>
<td>HTTPProcess</td>
<td>Performs any pending operations for the currently loaded HTTP connection.</td>
</tr>
</tbody>
</table>
HTTPReadTo

Reads to a buffer until a specified delimiter character.

HTTPSendFile

Serves up the next chunk of curHTTP’s file, up to a) available TX FIFO space or b) the next callback index, whichever comes first.

### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_CACHE_LEN</td>
<td>Max lifetime (sec) of static responses as string</td>
</tr>
<tr>
<td>HTTP_MAX_DATA_LEN</td>
<td>Define the maximum data length for reading cookie and GET/POST arguments (bytes)</td>
</tr>
<tr>
<td>HTTP_MAX_HEADER_LEN</td>
<td>Set to length of longest string above</td>
</tr>
<tr>
<td>HTTP_MIN_CALLBACK_FREE</td>
<td>Define the minimum number of bytes free in the TX FIFO before executing callbacks</td>
</tr>
<tr>
<td>HTTP_PORT</td>
<td>Define the listening port for the HTTP server</td>
</tr>
<tr>
<td>HTTP_TIMEOUT</td>
<td>Max time (sec) to await more data before timing out and disconnecting the socket</td>
</tr>
<tr>
<td>HTTPS_PORT</td>
<td>Define the listening port for the HTTPS server (if STACK_USE_SSL_SERVER is enabled)</td>
</tr>
<tr>
<td>smHTTP</td>
<td>Access the current state machine</td>
</tr>
<tr>
<td>RESERVED_HTTP_MEMORY</td>
<td>Macro indicating how much RAM to allocate on an ethernet controller to store HTTP state data.</td>
</tr>
</tbody>
</table>

### Module

**HTTP2 Server**

### Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP Connection Struct</td>
<td>Stores partial state data for each</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curHTTPID</td>
<td>ID of the currently loaded HTTP_CONN</td>
</tr>
<tr>
<td>httpContentTypes</td>
<td>Content-type strings corresponding to HTTP_FILE_TYPE</td>
</tr>
<tr>
<td>httpFileExtensions</td>
<td></td>
</tr>
<tr>
<td>HTTPRequestHeaders</td>
<td>Header strings for which we'd like to parse</td>
</tr>
<tr>
<td>HTTPResponseHeaders</td>
<td>Initial response strings (Corresponding to HTTP_STATUS)</td>
</tr>
<tr>
<td>httpStubs</td>
<td>HTTP stubs with state machine and socket</td>
</tr>
</tbody>
</table>
curHTTPID Variable

C
BYTE curHTTPID;

Description

ID of the currently loaded HTTP_CONN Stack API > HTTP2 Server > Internal Members > curHTTPID Variable
HTTP_CACHE_LEN Macro

```c
#define HTTP_CACHE_LEN ("600")  // Max lifetime (sec) of static responses as string
```

Description

Max lifetime (sec) of static responses as string
**HTTP_FILE_TYPE Enumeration**

```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;
```

**Description**

File type definitions

**Members**

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_TXT = 0u</td>
<td>File is a text document</td>
</tr>
<tr>
<td>HTTP_HTM</td>
<td>File is HTML (extension .htm)</td>
</tr>
<tr>
<td>HTTP_HTML</td>
<td>File is HTML (extension .html)</td>
</tr>
<tr>
<td>HTTP_CGI</td>
<td>File is HTML (extension .cgi)</td>
</tr>
<tr>
<td>HTTP_XML</td>
<td>File is XML (extension .xml)</td>
</tr>
<tr>
<td>HTTP_CSS</td>
<td>File is stylesheet (extension .css)</td>
</tr>
<tr>
<td>HTTP_GIF</td>
<td>File is GIF image (extension .gif)</td>
</tr>
<tr>
<td>HTTP_PNG</td>
<td>File is PNG image (extension .png)</td>
</tr>
<tr>
<td>HTTP_JPG</td>
<td>File is JPG image (extension .jpg)</td>
</tr>
<tr>
<td>HTTP_JAVA</td>
<td>File is java (extension .java)</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>HTTP_WAV</td>
<td>File is audio (extension .wav)</td>
</tr>
<tr>
<td>HTTP_UNKNOWN</td>
<td>File type is unknown</td>
</tr>
</tbody>
</table>
HTTP_MAX_DATA_LEN Macro

C

#define HTTP_MAX_DATA_LEN (100u)

Description

Define the maximum data length for reading cookie and GET/POST arguments (bytes)
HTTP_MAX_HEADER_LEN Macro

C

#define HTTP_MAX_HEADER_LEN (15u)

Description

Set to length of longest string above
HTTP_MIN_CALLBACK_FREE Macro

C

#define HTTP_MIN_CALLBACK_FREE (16u)

Description

Define the minimum number of bytes free in the TX FIFO before executing callbacks

Stack API > HTTP2 Server > Internal Members > HTTP_MIN_CALLBACK_FREE Macro

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
HTTP_PORT Macro

C

#define HTTP_PORT (80u)

Description

Define the listening port for the HTTP server

Stack API > HTTP2 Server > Internal Members > HTTP_PORT Macro
HTTP_STATUS Enumeration

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;

Description

Supported Commands and Server Response Codes

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_GET = 0u</td>
<td>GET command is being processed</td>
</tr>
<tr>
<td>HTTP_POST</td>
<td>POST command is being processed</td>
</tr>
<tr>
<td>HTTP_BAD_REQUEST</td>
<td>400 Bad Request will be returned</td>
</tr>
<tr>
<td>HTTP_UNAUTHORIZED</td>
<td>401 Unauthorized will be returned</td>
</tr>
<tr>
<td>HTTP_NOT_FOUND</td>
<td>404 Not Found will be returned</td>
</tr>
<tr>
<td>HTTP_OVERFLOW</td>
<td>414 Request-URI Too Long will be returned</td>
</tr>
<tr>
<td>HTTP_INTERNAL_SERVER_ERROR</td>
<td>500 Internal Server Error will be returned</td>
</tr>
<tr>
<td>HTTP_SSL_REQUIRED</td>
<td>501 Not Implemented (not a GET or POST</td>
</tr>
<tr>
<td>HTTP_STATUS</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>HTTP_NOT_IMPLEMENTED</td>
<td>command)</td>
</tr>
<tr>
<td>HTTP_MPFS_FORM</td>
<td>Show the MPFS Upload form</td>
</tr>
<tr>
<td>HTTP_MPFS_UP</td>
<td>An MPFS Upload is being processed</td>
</tr>
<tr>
<td>HTTP_MPFS_OK</td>
<td>An MPFS Upload was successful</td>
</tr>
<tr>
<td>HTTP_MPFS_ERROR</td>
<td>An MPFS Upload was not a valid image</td>
</tr>
<tr>
<td>HTTP_REDIRECT</td>
<td>302 Redirect will be returned</td>
</tr>
<tr>
<td>HTTP_SSL_REQUIRED</td>
<td>403 Forbidden is returned, indicating SSL is required</td>
</tr>
</tbody>
</table>

Stack API > HTTP2 Server > Internal Members > HTTP_STATUS Enumeration
## HTTP_STUB Structure

```c
typedef struct {
    SM_HTTP2 sm;
    TCP_SOCKET socket;
} HTTP_STUB;
```

### Description

HTTP Connection Struct Stores partial state data for each connection. Meant for storage in fast access RAM.

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM_HTTP2 sm;</td>
<td>Current connection state</td>
</tr>
<tr>
<td>TCP_SOCKET socket;</td>
<td>Socket being served</td>
</tr>
</tbody>
</table>

[Stack API] > [HTTP2 Server] > [Internal Members] > [HTTP_STUB Structure]
HTTP_TIMEOUT Macro

C

#define HTTP_TIMEOUT (45u) // Max time (sec) to await more data before timing out and disconnecting the socket

Description

Max time (sec) to await more data before timing out and disconnecting the socket
httpContentTypes Variable

Description

Content-type strings corresponding to **HTTP_FILE_TYPE**

Stack API > HTTP2 Server > Internal Members > httpContentTypes Variable
httpFileExtensions Variable

ROM char * ROM httpFileExtensions[HTTP_UNKNOWN+1] = { "txt", "htm", "h"
HTTPHeaderParseAuthorization Function

```
C
static void HTTPHeaderParseAuthorization();
```

**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 is called from CustomHTTPApp.c to determine if the credentials are acceptable.

The return value of HTTPCheckAuth is saved in curHTTP.isAuthorized for later use by the application.

**Preconditions**

None

**Returns**

None

**Remarks**

This function is only available when HTTP_USE_AUTHENTICATION is defined.
HTTPHeaderParseContentLength Function

```c
static void HTTPHeaderParseContentLength();
```

**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.

**Preconditions**

None

**Returns**

None

**Remarks**

This function is only available when HTTP_USE_POST is defined.

Stack API > HTTP2 Server > Internal Members > HTTPHeaderParseContentLength Function
HTTPHeaderParseCookie Function

```c
static void HTTPHeaderParseCookie();
```

**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()` and `HTTPGetROMArg()` functions.

**Preconditions**

None

**Returns**

None

**Remarks**

This function is only available when HTTP_USE_COOKIES is defined.

Stack API > HTTP2 Server > Internal Members > HTTPHeaderParseCookie Function
HTTPHeaderParseLookup Function

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>the index of the string found in HTTPRequestHeaders</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>the end of the file was reached and reading is done</td>
</tr>
<tr>
<td>FALSE</td>
<td>more data remains to be read</td>
</tr>
</tbody>
</table>
HTTPIncFile Function

```c
void HTTPIncFile(
    ROM BYTE* cFile
);
```

**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.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cFile</td>
<td>the name of the file to be sent</td>
</tr>
</tbody>
</table>

**Returns**

None

**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.
HTTPLoadConn Function

```c
static void HTTPLoadConn(
    BYTE hHTTP
);
```

**Description**

Saves the currently loaded HTTP connection back to Ethernet buffer RAM, then loads the selected connection into `curHTTP` in local RAM for processing.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hHTTP</td>
<td>the connection ID to load</td>
</tr>
</tbody>
</table>

**Returns**

None

Stack API > HTTP2 Server > Internal Members > HTTPLoadConn Function
HTTPMPFSUpload Function

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.

Preconditions

`MPFSFormat()` has been called.

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_IO_DONE</td>
<td>on success</td>
</tr>
<tr>
<td>HTTP_IO_NEED_DATA</td>
<td>if more data is still expected</td>
</tr>
</tbody>
</table>

Remarks

This function is only available when MPFS uploads are enabled and the MPFS image is stored in EEPROM.
HTTPProcess Function

```c
static void HTTPProcess();
```

Description

Performs any pending operations for the currently loaded HTTP connection.

Preconditions

HTTPInit() and HTTPLoadConn() have been called.

Returns

None
HTTPReadTo Function

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 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cDelim</td>
<td>the character at which to stop reading, or NULL to read to the end of the buffer</td>
</tr>
<tr>
<td>cData</td>
<td>where to store the data being read</td>
</tr>
<tr>
<td>wLen</td>
<td>how many bytes can be written to cData</td>
</tr>
</tbody>
</table>
## Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_READ_OK</td>
<td>data was successfully read</td>
</tr>
<tr>
<td>HTTP_READ_TRUNCATED</td>
<td>entire data could not fit in the buffer, so the data was truncated and data has been lost</td>
</tr>
<tr>
<td>HTTP_READ_INCOMPLETE</td>
<td>delimiter character was not found</td>
</tr>
</tbody>
</table>

Stack API > HTTP2 Server > Internal Members > HTTPReadTo Function
**HTTPRequestHeaders Variable**

```
ROM char * ROM HTTPRequestHeaders[] = { "Cookie:", "Authorization:", "
```

**Description**

Header strings for which we'd like to parse
DESCRIPTION

Initial response strings (Corresponding to HTTP_STATUS)

```c
ROM char * ROM HTTPResponseHeaders[] = {
  "HTTP/1.1 200 OK\r\n\nConnection:",
  "HTTP/1.1 403 Forbidden\r\nConnection:",
  "403 Forbidden: SSL Required - use HTTPS"
};
```

Stack API > HTTP2 Server > Internal Members > HTTPResponseHeaders Variable
HTTPS_PORT Macro

```c
#define HTTPS_PORT (443u)
```

**Description**

Define the listening port for the HTTPS server (if STACK_USE_SSL_SERVER is enabled)
HTTPSendFile Function

C

```c
static BOOL HTTPSendFile();
```

Description

Serves up the next chunk of `curHTTP`'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

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>the end of the file was reached and reading is done</td>
</tr>
<tr>
<td>FALSE</td>
<td>more data remains to be read</td>
</tr>
</tbody>
</table>

Microchip TCP/IP Stack Help | Contents | Index | Home | Previous | Up | Next
---|---|---|---|---|---|---

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
httpStubs Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_STUB</td>
</tr>
</tbody>
</table>

Description

HTTP stubs with state machine and socket
# SM_HTTP2 Enumeration

C
```
typedef enum {
    SM_HTTP_IDLE = 0u,
    SM_HTTPPARSE_REQUEST,
    SM_HTTPPARSE_HEADERS,
    SM_HTTPAUTHENTICATE,
    SM_HTTPPROCESS_GET,
    SM_HTTPPROCESS_POST,
    SM_HTTPPROCESS_REQUEST,
    SM_HTTPSERVE_HEADERS,
    SM_HTTPSERVE_COOKIES,
    SM_HTTPSERVE_BODY,
    SM_HTTP_SEND_FROM_CALLBACK,
    SM_HTTP_DISCONNECT
} SM_HTTP2;
```

## Description

Basic HTTP Connection State Machine

## Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM_HTTP_IDLE = 0u</td>
<td>Socket is idle</td>
</tr>
<tr>
<td>SM_HTTP_PARSE_REQUEST</td>
<td>Parses the first line for a file name and GET args</td>
</tr>
<tr>
<td>SM_HTTP_PARSE_HEADERS</td>
<td>Reads and parses headers one at a time</td>
</tr>
<tr>
<td>SM_HTTP_AUTHENTICATE</td>
<td>Validates the current authorization state</td>
</tr>
<tr>
<td>SM_HTTP_PROCESS_GET</td>
<td>Invokes user callback for GET args or cookies</td>
</tr>
<tr>
<td>SM_HTTP_PROCESS_POST</td>
<td>Invokes user callback for POSTed data</td>
</tr>
<tr>
<td>SM_HTTP_PROCESS_REQUEST</td>
<td>Begins the process of returning data</td>
</tr>
<tr>
<td>SM_HTTP_SERVE_HEADERS</td>
<td>Sends any required headers for the response</td>
</tr>
<tr>
<td>SM_HTTP_SERVE_COOKIES</td>
<td>Adds any cookies to the response</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>SM_HTTP_SERVE_BODY</td>
<td>Serves the actual content</td>
</tr>
<tr>
<td>SM_HTTP_SEND_FROM_CALLBACK</td>
<td>Invokes a dynamic variable callback</td>
</tr>
<tr>
<td>SM_HTTP_DISCONNECT</td>
<td>Disconnects the server and closes all files</td>
</tr>
</tbody>
</table>
smHTTP Macro

C

#define smHTTP httpStubs[curHTTPID].sm  // Access the current state machine

Description

Access the current state machine
RESERVED_HTTP_MEMORY Macro

C

#define RESERVED_HTTP_MEMORY ((DWORD)MAX_HTTP_CONNECTIONS * (DWORD)size

Description

Macro indicating how much RAM to allocate on an ethernet controller to store HTTP state data.
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.

ICMP Ping Process

NOTES:
Denotes a required task break
where StackTask() can execute

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMP Public Members</td>
<td>Functions and variables accessible or implemented by the stack application</td>
</tr>
<tr>
<td>ICMP Internal Members</td>
<td>Functions and variables used internally by the ICMP module</td>
</tr>
</tbody>
</table>

Stack API > ICMP
ICMP Public Members

The following functions and variables are accessible or implemented by the stack application.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMPBeginUsage</td>
<td>Claims ownership of the ICMP module.</td>
</tr>
<tr>
<td>ICMPGetReply</td>
<td>None</td>
</tr>
<tr>
<td>ICMPSendPing</td>
<td>None</td>
</tr>
<tr>
<td>ICMPSendPingToHost</td>
<td>None</td>
</tr>
<tr>
<td>ICMPSendPingToHostROM</td>
<td>Begins the process of transmitting an ICMP echo request. This normally involves an ARP resolution procedure first.</td>
</tr>
<tr>
<td>ICMPGetReply</td>
<td>None</td>
</tr>
<tr>
<td>ICMPEndUsage</td>
<td>Gives up ownership of the ICMP module.</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMPSendPingToHostROM</td>
<td>This is macro ICMPSendPingToHostROM.</td>
</tr>
</tbody>
</table>

Module

ICMP

Stack API > ICMP > Public Members
ICMPBeginUsage Function

C

BOOL ICMPBeginUsage();

Description

Claims ownership of the ICMP module.

Preconditions

None

Returns

TRUE: You have successfully gained ownership of the ICMP client module and can now use the ICMPSendPing() and ICMPGetReply() functions. FALSE: Some other application is using the ICMP client module. Calling ICMPSendPing() will corrupt the other application's ping result.

Side Effects

None

Remarks

None
ICMPSendPing Function

C

```c
void ICMPSendPing(
    DWORD dwRemoteIP
);
```

Description

None

Preconditions

ICMPBeginUsage() returned TRUE

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwRemoteIP</td>
<td>IP Address to ping. Must be stored big endian. Ex. 192.168.0.1 should be passed as 0x0100A8C0.</td>
</tr>
</tbody>
</table>

Returns

Begins the process of transmitting an ICMP echo request. This normally involves an ARP resolution procedure first.

Side Effects

None

Remarks

None

Stack API > ICMP > Public Members > ICMPSendPing Function
ICMPSendPingToHost Function

```c
void ICMPSendPingToHost( BYTE * szRemoteHost );
```

**Description**

None

**Preconditions**

ICMPBeginUsage() returned TRUE

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szRemoteHost</td>
<td>Host name to ping. Must be stored in RAM if being called by PIC18. Ex. <a href="http://www.microchip.com">www.microchip.com</a></td>
</tr>
</tbody>
</table>

**Returns**

Begins the process of transmitting an ICMP echo request. This normally involves an ARP resolution procedure first.

**Side Effects**

None

**Remarks**

None
ICMPSendPingToHostROM Function

C

```c
void ICMPSendPingToHostROM(
    ROM BYTE * szRemoteHost
);
```

Description

Begins the process of transmitting an ICMP echo request. This normally involves an ARP resolution procedure first.

Preconditions

ICMPBeginUsage() returned TRUE

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szRemoteHost</td>
<td>Host name to ping. Must be stored in ROM. Should only be called by PIC18. Ex. <a href="http://www.microchip.com">www.microchip.com</a></td>
</tr>
</tbody>
</table>

Returns

None

Side Effects

None

Remarks

None

Stack API > ICMP > Public Members > ICMPSendPingToHostROM Function
ICMPGetReply Function

C

LONG ICMPGetReply();

Description

None

Preconditions

ICMPBeginUsage() returned TRUE and ICMPSendPing() was called

Returns

-3: Could not resolve hostname (DNS timeout or hostname invalid) -2: No response received yet -1: Operation timed out (longer than ICMP_TIMEOUT) has elapsed. >=0: Number of TICKs that elapsed between initial ICMP transmission and reception of a valid echo.

Side Effects

None

Remarks

None
ICMPEndUsage Function

C

```c
void ICMPEndUsage();
```

Description

Gives up ownership of the ICMP module.

Preconditions

ICMPBeginUsage() was called by you and it returned TRUE.

Returns

Your ownership of the ICMP module is released. You can no longer use ICMPSendPing().

Side Effects

None

Remarks

None
ICMPSendPingToHostROM Macro

```c
#define ICMPSendPingToHostROM(a) ICMPSendPingToHost((BYTE*)(a))
```

Description

This is macro ICMPSendPingToHostROM.
ICMP Internal Members

The following functions and variables are designated as internal to the ICMP module.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMPProcess</td>
<td>None</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMP_TIMEOUT</td>
<td>ICMP Timeout Value</td>
</tr>
</tbody>
</table>

Module

ICMP

Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMP_PACKET</td>
<td>ICMP Packet Structure</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMPFlags</td>
<td>ICMP Flag structure</td>
</tr>
<tr>
<td>ICMPState</td>
<td>ICMP State Machine Enumeration</td>
</tr>
<tr>
<td>ICMPTimer</td>
<td>ICMP tick timer variable</td>
</tr>
<tr>
<td>StaticVars</td>
<td>ICMP Static Variables</td>
</tr>
</tbody>
</table>
Stack API > ICMP > Internal Members

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
ICMPProcess Function

C

void ICMPProcess(
    NODE_INFO * remote,
    WORD len
);

Description

None

Preconditions

MAC buffer contains ICMP type packet.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*remote</td>
<td>Pointer to a NODE_INFO structure of the ping requester</td>
</tr>
<tr>
<td>len</td>
<td>Count of how many bytes the ping header and payload are in this IP packet</td>
</tr>
</tbody>
</table>

Returns

Generates an echo reply, if requested Validates and sets ICMPFlags.bReplyValid if a correct ping response to one of ours is received.

Side Effects

None

Remarks

None
ICMPFlags Variable

C

```c
struct {
    unsigned char bICMPInUse : 1;
    unsigned char bReplyValid : 1;
    unsigned char bRemoteHostIsROM : 1;
} ICMPFlags;
```

Description

ICMP Flag structure

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char bICMPInUse : 1;</td>
<td>Indicates that the ICMP Client is in use</td>
</tr>
<tr>
<td>unsigned char bReplyValid : 1;</td>
<td>Indicates that a correct Ping response to one of our pings was received</td>
</tr>
<tr>
<td>unsigned char bRemoteHostIsROM : 1;</td>
<td>Indicates that a remote host name was passed as a ROM pointer argument</td>
</tr>
</tbody>
</table>
ICMP_PACKET Structure

C

typedef struct {
    BYTE vType;
    BYTE vCode;
    WORD wChecksum;
    WORD wIdentifier;
    WORD wSequenceNumber;
    WORD wData;
} ICMP_PACKET;

Description

ICMP Packet Structure

Stack API > ICMP > Internal Members > ICMP_PACKET Structure
ICMPState Variable

```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

Stack API > ICMP > Internal Members > ICMPState Variable
ICMP_TIMEOUT Macro

C

#define ICMP_TIMEOUT (4ul*TICK_SECOND)

Description

ICMP Timeout Value
ICMPTimer Variable

```c
DWORD ICMPTimer;
```

Description

ICMP tick timer variable

Stack API > ICMP > Internal Members > ICMPTimer Variable
**StaticVars Variable**

```c
union {
    union {
        ROM BYTE * szROM;
        BYTE * szRAM;
    } RemoteHost;
    NODE_INFO ICMPRemote;
} StaticVars;
```

**Description**

ICMP Static Variables

[Stack API] > [ICMP] > [Internal Members] > [StaticVars Variable]
c

WORD wICMPSequenceNumber;

Description

ICMP Sequence Number
**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** 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 MFPS_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** is described in the MPFS2 Utility documentation. Uploading images via HTTP can be accomplished as described in the **Getting Started** 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 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.

**Topics**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MPFS2 Public Members</strong></td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td><strong>MPFS2 Stack Members</strong></td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td><strong>MPFS2 Internal Members</strong></td>
<td>Functions and variables internal to the MPFS2 module</td>
</tr>
</tbody>
</table>

Stack API > MPFS2

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] [Index] [Home]
MPFS2 Public Members

The following functions and variables are accessible by the stack application.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPFS.Seek.MODE</td>
<td>Indicates the method for MPFSSend</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPFSClose</td>
<td>Closes a file.</td>
</tr>
<tr>
<td>MPFSFormat</td>
<td>Prepares the MPFS image for writing.</td>
</tr>
<tr>
<td>MPFSGet</td>
<td>Reads a byte from a file.</td>
</tr>
<tr>
<td>MPFSGetArray</td>
<td>Reads a series of bytes from a file.</td>
</tr>
<tr>
<td>MPFS.GetBytesRem</td>
<td>Determines how many bytes remain to be read.</td>
</tr>
<tr>
<td>MPFSGetEndAddr</td>
<td>Determines the ending address of a file.</td>
</tr>
<tr>
<td>MPFSGetFilename</td>
<td>Reads the file name of a file that is already open.</td>
</tr>
<tr>
<td>MPFSGetFlags</td>
<td>Reads a file's flags.</td>
</tr>
<tr>
<td>MPFSGetID</td>
<td>Determines the ID in the FAT for a file.</td>
</tr>
<tr>
<td>MPFSGetLong</td>
<td>Reads a DWORD or Long value from the MPFS.</td>
</tr>
<tr>
<td>MPFSGetMicrotime</td>
<td>Reads the microtime portion of a file's timestamp.</td>
</tr>
<tr>
<td>MPFSGetPosition</td>
<td>Determines the current position in the file</td>
</tr>
<tr>
<td>MPFSGetSize</td>
<td>Reads the size of a file.</td>
</tr>
</tbody>
</table>
### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPFS_INVALID</td>
<td>Indicates a position pointer is invalid</td>
</tr>
<tr>
<td>MPFS_INVALID.Handle</td>
<td>Indicates that a handle is not valid</td>
</tr>
</tbody>
</table>

### Module

**MPFS2**

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPFS_HANDLE</td>
<td>MPFS Handles are currently stored as BYTEs</td>
</tr>
</tbody>
</table>

---

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
MPFS_HANDLE Type

C

```c
typedef BYTE MPFS_HANDLE;
```

Description

MPFS Handles are currently stored as BYTEs
MPFS_INVALID Macro

C

#define MPFS_INVALID (0xffffffffu) // Indicates a position pointer

Description

Indicates a position pointer is invalid

Stack API > MPFS2 > Public Members > MPFS_INVALID Macro
MPFS_INVALID_HANDLE Macro

```c
#define MPFS_INVALID_HANDLE (0xffu)  // Indicates that a handle is not valid
```

Description

Indicates that a handle is not valid
typedef enum {
    MPFS_SEEK_START = 0u,
    MPFS_SEEK_END,
    MPFS_SEEK_FORWARD,
    MPFS_SEEK_REWIND
} MPFS_SEEK_MODE;

Description

Indicates the method for MPFSSeek

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPFS_SEEK_START = 0u</td>
<td>Seek forwards from the front of the file</td>
</tr>
<tr>
<td>MPFS_SEEK_END</td>
<td>Seek backwards from the end of the file</td>
</tr>
<tr>
<td>MPFS_SEEK_FORWARD</td>
<td>Seek forward from the current position</td>
</tr>
<tr>
<td>MPFS_SEEK_REWIND</td>
<td>See backwards from the current position</td>
</tr>
</tbody>
</table>
MPFSClose Function

```c
void MPFSClose(MPFS_HANDLE hMPFS);
```

Description

Closes a file and releases its stub back to the pool of available handles.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle to be closed</td>
</tr>
</tbody>
</table>

Returns

None
MPFSFormat Function

C

MPFS_HANDLE MPFSFormat();

Description

Prepares the MPFS image for writing and locks the image so that other processes may not access it.

Preconditions

None

Returns

An MPFS handle that can be used for MPFSPut commands, or MPFS_INVALID_HANDLE when the EEPROM failed to initialize for writing.

Remarks

In order to prevent misreads, the MPFS will be inaccessible until MPFSClose is called. This function is not available when the MPFS is stored in internal Flash program memory.
**MPFSGet Function**

```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**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle from which to read</td>
</tr>
<tr>
<td>c</td>
<td>Where to store the byte that was read</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The byte was successfully read</td>
</tr>
<tr>
<td>FALSE</td>
<td>No byte was read because either the handle was invalid or the end of the file has been reached.</td>
</tr>
</tbody>
</table>

*Stack API > MPFS2 > Public Members > MPFSGet Function*
MPFSGetArray Function

C

```c
WORD MPFSGetArray(
    MPFS_HANDLE hMPFS,
    BYTE* cData,
    WORD wLen
);
```

Description

Reads a series of bytes from a file.

Preconditions

The file handle referenced by hMPFS is already open.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle from which to read</td>
</tr>
<tr>
<td>cData</td>
<td>where to store the bytes that were read</td>
</tr>
<tr>
<td>wLen</td>
<td>how many bytes to read</td>
</tr>
</tbody>
</table>

Returns

The number of bytes successfully read. If this is less than wLen, an EOF occurred while attempting to read.
MPFSGetStringRem Function

C

```
DWORD MPFSGetStringRem(
    MPFS_HANDLE hMPFS
);
```

Description

Determines how many bytes remain to be read.

Preconditions

The file handle referenced by hMPFS is already open.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle from which to read the metadata</td>
</tr>
</tbody>
</table>

Returns

The number of bytes remaining in the file as a DWORD
MPFSGetEndAddr Function

C

```c
MPFS_PTR MPFSGetEndAddr(
    MPFS_HANDLE hMPFS
);
```

Description

Determines the ending address of a file.

Preconditions

The file handle referenced by hMPFS is already open.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle from which to read the metadata</td>
</tr>
</tbody>
</table>

Returns

The address just after the file ends (start address of next file)
MPFSGetFilename Function

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle from which to determine the file name</td>
</tr>
<tr>
<td>cName</td>
<td>where to store the name of the file</td>
</tr>
<tr>
<td>wLen</td>
<td>the maximum length of data to store in cName</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>the file name was successfully located</td>
</tr>
<tr>
<td>FALSE</td>
<td>the file handle provided is not currently open</td>
</tr>
</tbody>
</table>
MPFSGetFlags Function

```c
WORD MPFSGetFlags(MPFS_HANDLE hMPFS);
```

**Description**

Reads a file's flags.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle from which to read the metadata</td>
</tr>
</tbody>
</table>

**Returns**

The flags that were associated with the file
MPFSGetID Function

C

```c
WORD MPFSGetID(
    MPFS_HANDLE hMPFS
);
```

Description

Determines the ID in the FAT for a file.

Preconditions

The file handle referenced by hMPFS is already open.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle from which to read the metadata</td>
</tr>
</tbody>
</table>

Returns

The ID in the FAT for this file

Remarks

Use this function in association with MPFSOpenID to quickly access file without permanently reserving a file handle.
MPFSGetLong Function

C

BOOL MPFSGetLong(
    MPFS_HANDLE hMPFS,
    DWORD* ul
);

Description

Reads a DWORD or Long value from the MPFS.

Preconditions

The file handle referenced by hMPFS is already open.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle from which to read</td>
</tr>
<tr>
<td>ul</td>
<td>where to store the DWORD or long value that was read</td>
</tr>
</tbody>
</table>

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.

Stack API > MPFS > Public Members > MPFSGetLong Function
MPFSGetMicrotime Function

C

```c
DWORD MPFSGetMicrotime(
    MPFS_HANDLE hMPFS
);
```

Description

Reads the microtime portion of a file's timestamp.

Preconditions

The file handle referenced by hMPFS is already open.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle from which to read the metadata</td>
</tr>
</tbody>
</table>

Returns

The microtime that was read as a DWORD
MPFSGetPosition Function

C

DWORD MPFSGetPosition(
    MPFS_HANDLE hMPFS
);

Description

Determines the current position in the file

Preconditions

The file handle referenced by hMPFS is already open.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle for which to determine position</td>
</tr>
</tbody>
</table>

Returns

The position in the file as a DWORD (or MPFS_PTR)

Remarks

Calling MPFSSeek(hMPFS, pos, MPFSSEEK_START) will return the pointer to this position at a later time. (Where pos is the value returned by this function.)
**MPFSGetSize Function**

```c
DWORD MPFSGetSize(
    MPFS_HANDLE hMPFS
);
```

**Description**

Reads the size of a file.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle from which to read the metadata</td>
</tr>
</tbody>
</table>

**Returns**

The size that was read as a DWORD
MPFSGetStartAddr Function

C

`MPFS_PTR MPFSGetStartAddr(
    MPFS_HANDLE hMPFS
);`

Description

Reads the starting address of a file.

Preconditions

The file handle referenced by hMPFS is already open.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle from which to read the metadata</td>
</tr>
</tbody>
</table>

Returns

The starting address of the file in the MPFS image
MPFSGetTimestamp Function

C

```c
DWORD MPFSGetTimestamp(
   MPFS_HANDLE hMPFS
);
```

Description

Reads the timestamp for the specified file.

Preconditions

The file handle referenced by hMPFS is already open.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle from which to read the metadata</td>
</tr>
</tbody>
</table>

Returns

The timestamp that was read as a DWORD
**MPFSOpen Function**

```c
MPFS_HANDLE MPFSOpen(
    BYTE* cFile
);
```

**Description**

Opens a file in the MPFS2 file system.

**Preconditions**

MPFSInit has been called

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cFile</td>
<td>a null terminated file name to open</td>
</tr>
</tbody>
</table>

**Returns**

An MPFS_HANDLE to the opened file if found, or MPFS_INVALID_HANDLE if the file could not be found or no free handles exist.
**MPFSOpenID Function**

C

```c
MPFS_HANDLE MPFSOpenID(
    WORD hFatID
);
```

**Description**

Quickly re-opens a file in the MPFS2 file system. Use this function along with **MPFSGetID()** to quickly re-open a file without tying up a permanent MPFSStub.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hFatID</td>
<td>the ID of a previous opened file in the FAT</td>
</tr>
</tbody>
</table>

**Returns**

An **MPFS_HANDLE** to the opened file if found, or **MPFS_INVALID_HANDLE** if the file could not be found or no free handles exist.

Stack API > MPFS2 > Public Members > MPFSOpenID Function
MPFSOpenROM Function

C

```
MPFS_HANDLE MPFSOpenROM(
    ROM BYTE* cFile
);
```

Description

Opens a file in the MPFS2 file system.

 Preconditions

None

 Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cFile</td>
<td>a null terminated file name to open</td>
</tr>
</tbody>
</table>

Returns

An MPFS_HANDLE to the opened file if found, or MPFS_INVALID_HANDLE if the file could not be found or no free handles exist.

Remarks

This function is aliased to MPFSOpen on non-PIC18 platforms.
MPFSPutArray Function

C

```c
WORD MPFSPutArray(
    MPFS_HANDLE hMPFS,
    BYTE* cData,
    WORD wLen
);```

Description

Writes an array of data to the MPFS image.

Preconditions

MPFSFormat was successfully called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle for writing</td>
</tr>
<tr>
<td>cData</td>
<td>the array of bytes to write</td>
</tr>
<tr>
<td>wLen</td>
<td>how many bytes to write</td>
</tr>
</tbody>
</table>

Returns

The number of bytes successfully written.

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 must be called after the last call to MPFSPutArray.
**MPFSSeek Function**

```c
BOOL MPFSSeek(
    MPFS_HANDLE hMPFS,
    DWORD dwOffset,
    MPFS_SEEK_MODE tMode
);
```

**Description**

Moves the current read pointer to a new location.

**Preconditions**

The file handle referenced by hMPFS is already open.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMPFS</td>
<td>the file handle to seek with</td>
</tr>
<tr>
<td>dwOffset</td>
<td>offset from the specified position in the specified direction</td>
</tr>
<tr>
<td>tMode</td>
<td>one of the MPFS_SEEK_MODE constants</td>
</tr>
</tbody>
</table>

**Returns**

TRUE - the seek was successful FALSE - either the new location or the handle itself was invalid
MPFSPutEnd Function

C

```c
void MPFSPutEnd(  
    BOOL final
);
```

Description

Finalizes an MPFS writing operation.

Preconditions

MPFSFormat and MPFSPutArray were successfully called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>final</td>
<td>TRUE if the application is done writing, FALSE if MPFS2 called this function locally.</td>
</tr>
</tbody>
</table>

Returns

None

Stack API > MPFS2 > Public Members > MPFSPutEnd Function
MPFS2 Stack Members

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.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPFSInit</td>
<td>Initializes the MPFS module.</td>
</tr>
</tbody>
</table>

Module

MPFS2

Stack API > MPFS2 > Stack Members

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
MPFSInit Function

C

```c
void MPFSInit();
```

Description

Sets all MPFS handles to closed, and initializes access to the EEPROM if necessary.

Preconditions

None

Returns

None

Remarks

This function is called only one during lifetime of the application.
The following functions and variables are designated as internal to the MPFS2 module.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadProgramMemory</td>
<td>Assembly function to read all three bytes of program memory for 16-bit parts</td>
</tr>
<tr>
<td>_LoadFATRecord</td>
<td>Loads the FAT record for a specified handle.</td>
</tr>
<tr>
<td>_Validate</td>
<td>Validates the MPFS Image</td>
</tr>
</tbody>
</table>

### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_FILE_NAME_LEN</td>
<td>Supports long file names to 64 characters</td>
</tr>
<tr>
<td>MPFS_WRITE_PAGE_SIZE</td>
<td>Defines the size of a page in EEPROM</td>
</tr>
<tr>
<td>MPFS2_FLAG_HASINDEX</td>
<td>Indicates a file has an associated index of dynamic variables</td>
</tr>
<tr>
<td>MPFS2_FLAG_ISZIPPED</td>
<td>Indicates a file is compressed with GZIP compression</td>
</tr>
<tr>
<td>MPFSTell</td>
<td>Alias of MPFSGetPosition</td>
</tr>
<tr>
<td>MPFS_INVALID_FAT</td>
<td>Indicates an invalid FAT cache</td>
</tr>
</tbody>
</table>

### Module

**MPFS2**

### Structures
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPFS_STUB</td>
<td>Stores each file handle’s information Handles are free when addr = MPFS_INVALID</td>
</tr>
<tr>
<td>MPFS_FAT_RECORD</td>
<td>Stores the data for an MPFS2 FAT record</td>
</tr>
</tbody>
</table>

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPFS_PTR</td>
<td>MPFS Pointers are currently DWORDs</td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isMPFSLocked</td>
<td>Allows the MPFS to be locked, preventing access during updates</td>
</tr>
<tr>
<td>lastRead</td>
<td>Track the last read address to prevent unnecessary data overhead to switch locations.</td>
</tr>
<tr>
<td>MPFSStubs</td>
<td>Track the MPFS File Handles MPFSStubs[0] is reserved for internal use (FAT access)</td>
</tr>
<tr>
<td>fatCache</td>
<td>FAT record cache</td>
</tr>
<tr>
<td>fatCacheID</td>
<td>ID of currently loaded fatCache</td>
</tr>
<tr>
<td>numFiles</td>
<td>Number of files in this MPFS image</td>
</tr>
</tbody>
</table>
isMPFSLocked Variable

C

BOOL isMPFSLocked;

Description

Allows the MPFS to be locked, preventing access during updates
lastRead Variable

C

MPFS_PTR lastRead;

Description

Track the last read address to prevent unnecessary data overhead to switch locations.
MAX_FILE_NAME_LEN Macro

C

```
#define MAX_FILE_NAME_LEN (64u)
```

Description

Supports long file names to 64 characters

Stack API > MPFS2 > Internal Members > MAX_FILE_NAME_LEN Macro
MPFS_PTR Type

C

typedef DWORD MPFS_PTR;

Description

MPFS Pointers are currently DWORDs
MPFS_STUB Structure

C

```c
typedef struct {
    MPFS_PTR addr;
    DWORD bytesRem;
    WORD fatID;
} MPFS_STUB;
```

Description

Stores each file handle's information. Handles are free when addr = MPFS_INVALID

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPFS_PTR addr;</td>
<td>Current address in the file system</td>
</tr>
<tr>
<td>DWORD bytesRem;</td>
<td>How many bytes remain in this file</td>
</tr>
<tr>
<td>WORD fatID;</td>
<td>ID of which file in the FAT was accessed</td>
</tr>
</tbody>
</table>

Stack API > MPFS2 > Internal Members > MPFS_STUB Structure
MPFS_WRITE_PAGE_SIZE Macro

C

#define MPFS_WRITE_PAGE_SIZE (64u) // Defines the size of a page in EEPROM

Description

Defines the size of a page in EEPROM

Stack API > MPFS2 > Internal Members > MPFS_WRITE_PAGE_SIZE Macro
MPFS2_FLAG_HASINDEX Macro

```c
#define MPFS2_FLAG_HASINDEX ((WORD)0x0002)   // Indicates a file has
```

Description

Indicates a file has an associated index of dynamic variables
#define MPFS2_FLAG_ISZIPPED ((WORD)0x0001)  // Indicates a file is compressed with GZIP compression

## Description

Indicates a file is compressed with GZIP compression
C
MPFS_STUB MPFSStubs[MAX_MPFS_HANDLES+1];

Description

Track the MPFS File Handles MPFSStubs[0] is reserved for internal use (FAT access)
MPFSTell Macro

C

```c
#define MPFSTell(a) MPFSGetPosition(a)
```

Description

Alias of MPFSGetPosition

Stack API > MPFS2 > Internal Members > MPFSTell Macro

Microchip TCP/IP Stack 5.42.08 - June 15, 2013  
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
ReadProgramMemory Function

C

DWORD ReadProgramMemory(
    DWORD address
);

Description

Assembly function to read all three bytes of program memory for 16-bit parts

Stack API > MPFS2 > Internal Members > ReadProgramMemory Function
__LoadFATRecord Function

C

```c
static void _LoadFATRecord(
    WORD fatID
);
```

Description

Loads the FAT record for a specified handle.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fatID</td>
<td>the ID of the file whose FAT is to be loaded</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

The FAT record will be stored in `fatCache`.
Validate Function

```c
static void _Validate();
```

**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

**Returns**

None
**C**

```c
typedef struct {
    DWORD string;
    DWORD data;
    DWORD len;
    DWORD timestamp;
    DWORD microtime;
    WORD flags;
} MPFS_FAT_RECORD;
```

**Description**

Stores the data for an MPFS2 FAT record

**Members**

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD string;</td>
<td>Pointer to the file name</td>
</tr>
<tr>
<td>DWORD data;</td>
<td>Address of the file data</td>
</tr>
<tr>
<td>DWORD len;</td>
<td>Length of file data</td>
</tr>
<tr>
<td>DWORD timestamp;</td>
<td>Timestamp of file</td>
</tr>
<tr>
<td>DWORD microtime;</td>
<td>Microtime stamp of file</td>
</tr>
<tr>
<td>WORD flags;</td>
<td>Flags for this file</td>
</tr>
</tbody>
</table>

**Stack API** > **MPFS2** > **Internal Members** > **MPFS_FAT_RECORD Structure**
fatCache Variable

```c
MPFS_FAT_RECORD fatCache;
```

Description

FAT record cache

Stack API > MPFS2 > Internal Members > fatCache Variable
fatCacheID Variable

C
WORD fatCacheID;

Description

ID of currently loaded fatCache

Stack API > MPFS2 > Internal Members > fatCacheID Variable
numFiles Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD numFiles;</td>
</tr>
</tbody>
</table>

**Description**

Number of files in this MPFS image
#define MPFS_INVALID_FAT (0xffffu) // Indicates an invalid FAT cache

Description

Indicates an invalid FAT cache
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 "TCP/IP 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.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NBNS Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack.</td>
</tr>
</tbody>
</table>

Stack API > NBNS
NBNS Stack Members

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.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NBNSGetName</td>
<td>Reads the NetBIOS name from a UDP socket and copies it into a user-specified buffer.</td>
</tr>
<tr>
<td>NBNSPutName</td>
<td>Transmits the NetBIOS name across an open UDP socket.</td>
</tr>
<tr>
<td>NBNSTask</td>
<td>Sends responses to NetBIOS name requests</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NBNS_PORT</td>
<td>NetBIOS Name Service port</td>
</tr>
</tbody>
</table>

Module

NBNS

Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NBNS_HEADER</td>
<td>NBNS Header structure</td>
</tr>
<tr>
<td>_NBNS_HEADER</td>
<td>NBNS Header structure</td>
</tr>
</tbody>
</table>
NBNSGetName Function

```
static void NBNSGetName(
    BYTE * String
);
```

Description

Reads the NetBIOS name from a UDP socket and copies it into a user-specified buffer.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Pointer to an array into which a received NetBIOS name should be copied.</td>
</tr>
</tbody>
</table>

Returns

None

Side Effects

None

Remarks

None
NBNSPutName Function

C

```c
static void NBNSPutName(
    BYTE * String
);
```

Description

Transmits the NetBIOS name across an open UDP socket.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>The name to transmit</td>
</tr>
</tbody>
</table>

Returns

None

Side Effects

None

Remarks

None

Stack API > NBNS > Stack Members > NBNSPutName Function
NBNSTask Function

```c
void NBNSTask();
```

Description

Sends responses to NetBIOS name requests

Preconditions

None

Returns

None

Side Effects

None

Remarks

None
NBNS_HEADER Structure

```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
NBNS_PORT Macro

C
#define NBNS_PORT (137u)

Description

NetBIOS Name Service port

Stack API > NBNS > Stack Members > NBNS_PORT Macro

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
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, TCPTXPerformanceTask, and TCPRXPerformanceTask.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Test Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td>Performance Test Internal Members</td>
<td>Functions and variables internal to the module</td>
</tr>
</tbody>
</table>

Stack API > Performance Tests
## Performance Test Stack Members

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.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPPerformanceTask</td>
<td>Tests the performance of the TCP module.</td>
</tr>
<tr>
<td>UDPPerformanceTask</td>
<td>Tests the transmit performance of the UDP module.</td>
</tr>
</tbody>
</table>

### Module

**Performance Tests**

Stack API > Performance Tests > Stack Members

---

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
TCPPerformanceTask Function

C

```c
void TCPPerformanceTask();
```

Description

This function calls both `TCPTXPerformanceTask` and `TCPRXPerformanceTask` to perform the performance task functions. Refer to the documentation for each of those functions for details.

Preconditions

TCP is initialized.

Returns

None
UDPPerformanceTask Function

```c
void UDPPerformanceTask();
```

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.

Returns

None
Performance Test Internal Members

The following functions and variables are designated as internal to the module.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPRXPerformanceTask</td>
<td>Tests the receive performance of the TCP module.</td>
</tr>
<tr>
<td>TCPTXPerformanceTask</td>
<td>Tests the transmit performance of the TCP module.</td>
</tr>
</tbody>
</table>

### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERFORMANCE_PORT</td>
<td>Which UDP port to broadcast from for the UDP tests</td>
</tr>
<tr>
<td>RX_PERFORMANCE_PORT</td>
<td>The TCP port to <a href="#">listen</a> on for TCP receive tests</td>
</tr>
<tr>
<td>TX_PERFORMANCE_PORT</td>
<td>The TCP port to <a href="#">listen</a> on for TCP transmit tests</td>
</tr>
</tbody>
</table>

### Module

**Performance Tests**

[Plugin Stack API > Performance Tests > Internal Members](#)
void TCPRXPerformanceTask();

Description

This function tests the receive performance of the TCP module. To use, open a telnet connection to the device on RX_PERFORMANCE_PORT (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.

Returns

None
TCPTXPerformanceTask Function

C

void TCPTXPerformanceTask();

Description

This function tests the transmit performance of the TCP module. To use, open a telnet connection to the device on TX_PERFORMANCE_PORT (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.

Returns

None

Stack API > Performance Tests > Internal Members > TCPTXPerformanceTask Function
PERFORMANCE_PORT Macro

```c
#define PERFORMANCE_PORT 9
```

Description

Which UDP port to broadcast from for the UDP tests
RX_PERFORMANCE_PORT Macro

C

```c
#define RX_PERFORMANCE_PORT 9763
```

Description

The TCP port to listen on for TCP receive tests

Stack API > Performance Tests > Internal Members > RX_PERFORMANCE_PORT Macro
The TCP port to listen on for TCP transmit tests

C

#define TX_PERFORMANCE_PORT 9762

Description

The TCP port to listen on for TCP transmit tests
SMTP Client

The SMTP client module in the TCP/IP Stack lets applications send e-mails to any recipient worldwide. These messages 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.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMTP Client Examples</td>
<td>Examples for using the SMTP Client</td>
</tr>
<tr>
<td>SMTP Client Public Members</td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td>SMTP Client Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td>SMTP Client Internal Members</td>
<td>Functions and variables used internally by the SMTP Client</td>
</tr>
</tbody>
</table>

Stack API > SMTP Client
SMTP Client Examples

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.

Module

SMTP Client

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMTP Client Short Message Example</td>
<td>Send a message whose contents can be read from memory</td>
</tr>
<tr>
<td>SMTP Client Long Message Example</td>
<td>Sends a long message built in pieces by the application</td>
</tr>
</tbody>
</table>

Stack API > SMTP Client > Examples

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
SMTP Client Short Message Example

The SMTP client API is simplified when messages can be buffered entirely in RAM. The SMTPDemo 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 TCPIPConfig.h before continuing.

The diagram below provides an overview of the process:

![Diagram](image)

First, call `SMTPBeginUsage` 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` to instruct the SMTP client to begin transmission.

The application must now call `SMTPIsBusy` 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` 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` 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.
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 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 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` 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` to verify that the connection to the remote server is active and has not been lost. If the call succeeds, call `SMTPIsPutReady` 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`, `SMTPPutArray`, `SMTPPutROMArray`, `SMTPPutString`, and `SMTPPutROMString` 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` to track how much free space remains in the TX buffer. Once the buffer is depleted, call `SMTPFlush` to force the data written to be sent.
The SMTP client module can accept as much data as the TCP TX FIFO can hold. This is determined by the socket initializer for TCP_PURPOSE_DEFAULT type sockets in TCPIPConfig.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 to inform the SMTP client that no more data remains. Then call SMTPIsBusy 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 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, and ensure that these fields are being properly assigned to SMTPClient 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.
The following functions and variables are available to the stack application.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMTPBeginUsage</td>
<td>Requests control of the SMTP client module.</td>
</tr>
<tr>
<td>SMTPEndUsage</td>
<td>Releases control of the SMTP client module.</td>
</tr>
<tr>
<td>SMTPFlush</td>
<td>Flushes the SMTP socket and forces all data to be sent.</td>
</tr>
<tr>
<td>SMTPIsBusy</td>
<td>Determines if the SMTP client is busy.</td>
</tr>
<tr>
<td>SMTPIsPutReady</td>
<td>Determines how much data can be written to the SMTP client.</td>
</tr>
<tr>
<td>SMTPPut</td>
<td>Writes a single byte to the SMTP client.</td>
</tr>
<tr>
<td>SMTPPutArray</td>
<td>Writes a series of bytes to the SMTP client.</td>
</tr>
<tr>
<td>SMTPPutDone</td>
<td>Indicates that the on-the-fly message is complete.</td>
</tr>
<tr>
<td>SMTPPutROMArray</td>
<td>Writes a series of bytes from ROM to the SMTP client.</td>
</tr>
<tr>
<td>SMTPPutROMString</td>
<td>Writes a string from ROM to the SMTP client.</td>
</tr>
<tr>
<td>SMTPPutString</td>
<td>Writes a string to the SMTP client.</td>
</tr>
<tr>
<td>SMTPSendMail</td>
<td>Initializes the message sending process.</td>
</tr>
</tbody>
</table>

### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMTP_CONNECT_ERROR</td>
<td>Connection to SMTP server failed</td>
</tr>
<tr>
<td>SMTP_RESOLVE_ERROR</td>
<td>DNS lookup for SMTP server failed</td>
</tr>
</tbody>
</table>
SMTP_SUCCESS

Message was successfully sent

Module

SMTP Client

Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMTP_POINTS</td>
<td>Configures the SMTP client to send a message</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMTPClient</td>
<td></td>
</tr>
</tbody>
</table>

Stack API > SMTP Client > Public Members
SMTP_CONNECT_ERROR Macro

```c
#define SMTP_CONNECT_ERROR (0x8001u) // Connection to SMTP server failed
```

**Description**

Connection to SMTP server failed

Stack API > SMTP Client > Public Members > SMTP_CONNECT_ERROR Macro
SMTP_POINTERS Structure

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;
    }
Description

This structure of pointers configures the SMTP Client to send an e-mail message. Initially, all pointers will be null. Set `SMTPClient.[field name].szRAM` to use a string stored in RAM, or `SMTPClient.[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.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>the SMTP server to relay the message through</td>
</tr>
<tr>
<td>Username</td>
<td>the user name to use when logging into the SMTP server, if any is required</td>
</tr>
<tr>
<td>Password</td>
<td>the password to supply when logging in, if any is required</td>
</tr>
<tr>
<td>To</td>
<td>the destination address for this message. May be a comma-separated list of addresss, and/or formatted.</td>
</tr>
<tr>
<td>CC</td>
<td>The CC addresses for this message, if any. May be a comma-separated list of addresss, and/or formatted.</td>
</tr>
<tr>
<td>BCC</td>
<td>The BCC addresses for this message, if any. May be a comma-separated list of addresss, and/or formatted.</td>
</tr>
<tr>
<td>From</td>
<td>The From address for this message. May be formatted.</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>Subject</td>
<td>The Subject header for this message.</td>
</tr>
<tr>
<td>OtherHeaders</td>
<td>Any additional headers for this message. Each additional header, including the last one, must be terminated with a CRLF pair.</td>
</tr>
<tr>
<td>Body</td>
<td>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.</td>
</tr>
<tr>
<td>ROMPointers</td>
<td>Indicates which parameters to read from ROM instead of RAM.</td>
</tr>
<tr>
<td>UseSSL</td>
<td>When STACK_USE_SSL_CLIENT is enabled, this flag causes the SMTP client to make an SSL connection to the server.</td>
</tr>
<tr>
<td>ServerPort</td>
<td>(WORD value) Indicates the port on which to connect to the remote SMTP server.</td>
</tr>
</tbody>
</table>

### 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>
SMTP_RESOLVE_ERROR Macro

C

#define SMTP_RESOLVE_ERROR (0x8000u) // DNS lookup for SMTP server

Description

DNS lookup for SMTP server failed

Stack API > SMTP Client > Public Members > SMTP_RESOLVE_ERROR Macro
SMTP_SUCCESS Macro

```c
#define SMTP_SUCCESS (0x0000u) // Message was successfully sent
```

Description

Message was successfully sent

Stack API > SMTP Client > Public Members > SMTP_SUCCESS Macro
SMTPBeginUsage Function

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 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

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The application has successfully obtained control of the module</td>
</tr>
<tr>
<td>FALSE</td>
<td>The SMTP module is in use by another application. Call SMTPBeginUsage again later, after returning to the main program loop</td>
</tr>
</tbody>
</table>

Stack API > SMTP Client > Public Members > SMTPBeginUsage Function
SMTPClient Variable

```c
SMTP_POINTERS SMTPClient;
```
SMTPEndUsage Function

C

\texttt{WORD \textbf{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 \texttt{SMTPBeginUsage}, and frees the SMTP client to be used by another application.

**Preconditions**

\texttt{SMTPBeginUsage} returned TRUE on a previous call.

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{SMTP_SUCCESS}</td>
<td>A message was successfully sent</td>
</tr>
<tr>
<td>\texttt{SMTP_RESOLVE_ERROR}</td>
<td>The SMTP server could not be resolved</td>
</tr>
<tr>
<td>\texttt{SMTP_CONNECT_ERROR}</td>
<td>The connection to the SMTP server failed or was prematurely terminated</td>
</tr>
<tr>
<td>1-199 and 300-399</td>
<td>The last SMTP server response code</td>
</tr>
</tbody>
</table>

Stack API  >  SMTP Client  >  Public Members  >  SMTPEndUsage Function
A function named `SMTPFlush` in C is defined as:

```c
void SMTPFlush();
```

### Description

Flushes the SMTP socket and forces all data to be sent.

### Preconditions

- `SMTPBeginUsage` returned TRUE on a previous call.

### Returns

None

### Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, `SMTPSendMail` was called with `SMTPClient.Body` set to NULL.)
SMTPIsBusy Function

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, SMTPPutDone to determine if the stack has finished performing its internal tasks. It should also be called prior to any call to SMTPIsPutReady to verify that the SMTP client has not prematurely disconnected. When this function returns FALSE, the next call should be to SMTPEndUsage to release the module and obtain the status code for the operation.

Preconditions

SMTPBeginUsage returned TRUE on a previous call.

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The SMTP Client is busy with internal tasks or sending an on-the-fly message.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The SMTP Client is terminated and is ready to be released.</td>
</tr>
</tbody>
</table>

Stack API > SMTP Client > Public Members > SMTPIsBusy Function
SMTPIsPutReady Function

C

WORD SMTPIsPutReady();

Description

Use this function to determine how much data can be written to the SMTP client when generating an on-the-fly message.

Preconditions

SMTMPBeginUsage returned TRUE on a previous call, and an on-the-fly message is being generated. This requires that SMTPSendMail was called with SMTPClient.Body set to NULL.

Returns

The number of free bytes the SMTP TX FIFO.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, SMTPSendMail was called with SMTPClient.Body set to NULL.)
SMTPPut Function

```c
BOOL SMTPPut ( BYTE c );
```

**Description**

Writes a single byte to the SMTP client.

**Preconditions**

`SMTPBeginUsage` returned TRUE on a previous call.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>The byte to be written</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The byte was successfully written</td>
</tr>
<tr>
<td>FALSE</td>
<td>The byte was not written, most likely because the buffer was full</td>
</tr>
</tbody>
</table>

**Remarks**

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, `SMTPSendMail` was called with SMTPClient.Body set to NULL.)
SMTPPutArray Function

C

WORD SMTPPutArray(  
    BYTE* Data,  
    WORD Len  
);

Description

Writes a series of bytes to the SMTP client.

Preconditions

SMTPBeginUsage returned TRUE on a previous call.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>The data to be written</td>
</tr>
<tr>
<td>Len</td>
<td>How many bytes should be written</td>
</tr>
</tbody>
</table>

Returns

The number of bytes written. If less than Len, then the TX FIFO became full before all bytes could be written.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, SMTPSendMail was called with SMTPClient.Body set to NULL.)
SMTPPutDone Function

C

```c
void SMTPPutDone();
```

Description

Indicates that the on-the-fly message is complete.

Preconditions

SMTPBeginUsage returned TRUE on a previous call, and the SMTP client is generated an on-the-fly message. (That is, SMTPSendMail was called with SMTPClient.Body set to NULL.)

Returns

None
SMTPPutROMArray Function

C

WORD SMTPPutROMArray(
    ROM BYTE* Data,
    WORD Len
);

Description

Writes a series of bytes from ROM to the SMTP client.

Preconditions

SMTPBeginUsage returned TRUE on a previous call.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>The data to be written</td>
</tr>
<tr>
<td>Len</td>
<td>How many bytes should be written</td>
</tr>
</tbody>
</table>

Returns

The number of bytes written. If less than Len, then the TX FIFO became full before all bytes could be written.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, SMTPSendMail was called with SMTPClient.Body set to NULL.)

This function is aliased to SMTPPutArray on non-PIC18 platforms.
**SMTPPutROMString Function**

```c
WORD SMTPPutROMString(
    ROM BYTE* Data
);
```

**Description**

Writes a string from ROM to the SMTP client.

**Preconditions**

- **SMTPBeginUsage** returned TRUE on a previous call.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>The data to be written</td>
</tr>
</tbody>
</table>

**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.

**Remarks**

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, **SMTPSendMail** was called with SMTPClient.Body set to NULL.)

This function is aliased to **SMTPPutString** on non-PIC18 platforms.
SMTPPutString Function

```c
WORD SMTPPutString(
    BYTE* Data
);
```

Description

Writes a string to the SMTP client.

Preconditions

SMTPBeginUsage returned TRUE on a previous call.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>The data to be written</td>
</tr>
</tbody>
</table>

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.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, SMTPSendMail was called with SMTPClient.Body set to NULL.)
SMTPSendMail Function

C

```c
void SMTPSendMail();
```

Description

This function starts the state machine that performs the actual transmission of the message. Call this function after all the fields in `SMTPClient` have been set.

Preconditions

`SMTPBeginUsage` returned TRUE on a previous call.

Returns

None
SMTP Client Stack Members

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.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMTPTask</td>
<td>Performs any pending SMTP client tasks</td>
</tr>
</tbody>
</table>

Module

SMTP Client

Stack API > SMTP Client > Stack Members
SMTPTask Function

C

void SMPTask();

Description

This function handles periodic tasks associated with the SMTP client, such as processing initial connections and command sequences.

Preconditions

None

Returns

None

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.
The following functions and variables are designated for internal use by the SMTP Client module.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindEmailAddress</td>
<td>Searches a string for an e-mail address.</td>
</tr>
<tr>
<td>FindROMEmailAddress</td>
<td>Searches a ROM string for an e-mail address.</td>
</tr>
</tbody>
</table>

### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMTP_PORT</td>
<td>Default port to use when unspecified</td>
</tr>
<tr>
<td>SMTP_SERVER_REPLY_TIMEOUT</td>
<td>How long to wait before assuming the connection has been dropped (default 8 seconds)</td>
</tr>
</tbody>
</table>

### Module

**SMTP Client**

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRPeriod</td>
<td>State machine for the CR LF Period replacement Used by SMTPPut to transparently replace &quot;rn.&quot; with &quot;rn..&quot;</td>
</tr>
<tr>
<td>MySocket</td>
<td>Socket currently in use by the SMTP client</td>
</tr>
<tr>
<td>PutHeadersState</td>
<td>State machine for writing the SMTP message headers</td>
</tr>
<tr>
<td>ResponseCode</td>
<td>Response code from server when an error exists</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>RXParserState</td>
<td>State machine for parsing incoming responses</td>
</tr>
<tr>
<td>SMTPFlags</td>
<td>Internal flags used by the SMTP Client</td>
</tr>
<tr>
<td>SMTPServer</td>
<td>IP address of the remote SMTP server</td>
</tr>
<tr>
<td>SMTPState</td>
<td>Message state machine for the SMTP Client</td>
</tr>
<tr>
<td>TransportState</td>
<td>State of the transport for the SMTP Client</td>
</tr>
</tbody>
</table>

Stack API > SMTP Client > Internal Members

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
CRPeriod Variable

C

union {
    BYTE * Pos;
    enum {
        CR_PERIOD_SEEK_CR = 0,
        CR_PERIOD_SEEK_LF,
        CR_PERIOD_SEEK_PERIOD,
        CR_PERIOD_NEED_INSERTION
    } State;
} CRPeriod;

Description

State machine for the CR LF Period replacement Used by SMTPPut to transparently replace "rn." with "rn.."

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR_PERIOD_SEEK_CR = 0</td>
<td>Idle state, waiting for 'r</td>
</tr>
<tr>
<td>CR_PERIOD_SEEK_LF</td>
<td>r&quot; has been written, so check next byte for 'n</td>
</tr>
<tr>
<td>CR_PERIOD_SEEK_PERIOD</td>
<td>rn&quot; has been written, so check next byte for '.'.</td>
</tr>
<tr>
<td>CR_PERIOD_NEED_INSERTION</td>
<td>&quot;rn.&quot; has been written, so an additional '.' must be written before the next byte.</td>
</tr>
</tbody>
</table>

Stack API > SMTP Client > Internal Members > CRPeriod Variable
FindEmailAddress Function

```c
static BYTE * FindEmailAddress(
    BYTE * str,
    WORD * wLen
);
```

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 returned TRUE on a previous call.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>The string in which to search for an e-mail address</td>
</tr>
<tr>
<td>wLen</td>
<td>the length of str</td>
</tr>
</tbody>
</table>

Returns

A pointer to the e-mail address
FindROMEmailAddress Function

C

```
static ROM BYTE * FindROMEmailAddress(
    ROM BYTE * str,
    WORD * wLen
);
```

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* returned TRUE on a previous call.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>The ROM string in which to search for an e-mail address</td>
</tr>
<tr>
<td>wLen</td>
<td>the length of str</td>
</tr>
</tbody>
</table>

Returns

A pointer to the e-mail address
### MySocket Variable

```c
TCP_SOCKET MySocket = INVALID_SOCKET;
```

**Description**

Socket currently in use by the SMTP client
PutHeadersState Variable

```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;
```

Description

State machine for writing the SMTP message headers

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUTHEADERS_FROM_INIT = 0</td>
<td>Preparing to send From header</td>
</tr>
<tr>
<td>PUTHEADERS_FROM</td>
<td>Sending From header</td>
</tr>
<tr>
<td>PUTHEADERS_TO_INIT</td>
<td>Preparing to send To header</td>
</tr>
<tr>
<td>PUTHEADERS_TO</td>
<td>Sending To header</td>
</tr>
<tr>
<td>PUTHEADERS_CC_INIT</td>
<td>Preparing to send CC header</td>
</tr>
<tr>
<td>PUTHEADERS_CC</td>
<td>Sending CC header</td>
</tr>
<tr>
<td>PUTHEADERS_SUBJECT_INIT</td>
<td>Preparing to send Subject header</td>
</tr>
<tr>
<td>PUTHEADERS_SUBJECT</td>
<td>Sending Subject header</td>
</tr>
<tr>
<td>PUTHEADERS_OTHER_INIT</td>
<td>Preparing to send additional headers</td>
</tr>
<tr>
<td>State</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>PUTHEADERS_OTHER</td>
<td>Sending additional headers</td>
</tr>
<tr>
<td>PUTHEADERS_DONE</td>
<td>Done writing all headers</td>
</tr>
</tbody>
</table>
ResponseCode Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD ResponseCode;</td>
</tr>
</tbody>
</table>

**Description**

Response code from server when an error exists
RXParserState Variable

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

Stack API > SMTP Client > Internal Members > RXParserState Variable
SMTP_PORT Macro

C
#define SMTP_PORT 25 // Default port to use when unspecified

Description

Default port to use when unspecified
SMTP_SERVER_REPLY_TIMEOUT Macro

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)
SMTPFlags Variable

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
SMTPServer Variable

```
C
IP_ADDR SMTPServer;
```

Description

IP address of the remote SMTP server
SMTPState Variable

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;
```

Description

Message state machine for the SMTP Client

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SMTP_STATE</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>SMTP_HOME = 0</td>
<td>Idle start state for SMTP client (application is preparing message)</td>
</tr>
<tr>
<td>SMTP_HELO</td>
<td>HELO is being sent to server</td>
</tr>
<tr>
<td>SMTP_HELO_ACK</td>
<td>Received an ACK for the HELO</td>
</tr>
<tr>
<td>SMTP_AUTH_LOGIN</td>
<td>Requesting to log in</td>
</tr>
<tr>
<td>SMTP_AUTH_LOGIN_ACK</td>
<td>Log in request accepted</td>
</tr>
<tr>
<td>SMTP_AUTH_USERNAME</td>
<td>Sending user name</td>
</tr>
<tr>
<td>SMTP_AUTH_USERNAME_ACK</td>
<td>User name accepted</td>
</tr>
<tr>
<td>SMTP_AUTH_PASSWORD</td>
<td>Sending password</td>
</tr>
<tr>
<td>SMTP_AUTH_PASSWORD_ACK</td>
<td>Password was accepted</td>
</tr>
<tr>
<td>SMTP_MAILFROM</td>
<td>Sending initial MAIL FROM command</td>
</tr>
<tr>
<td>SMTP_MAILFROM_ACK</td>
<td>MAIL FROM was accepted</td>
</tr>
<tr>
<td>SMTP_RCPTTO_INIT</td>
<td>Preparing to send RCPT TO</td>
</tr>
<tr>
<td>SMTP_RCPTTO</td>
<td>Sending RCPT TO command</td>
</tr>
<tr>
<td>SMTP_RCPTTO_ACK</td>
<td>RCPT TO was accepted</td>
</tr>
<tr>
<td>SMTP_RCPTTO_ISDONE</td>
<td>Done sending RCPT TO commands</td>
</tr>
<tr>
<td>SMTP_RCPTTOCC_INIT</td>
<td>Preparing to send RCPT TO CC commands</td>
</tr>
<tr>
<td>SMTP_RCPTTOCC</td>
<td>Sending RCPT TO CC commands</td>
</tr>
<tr>
<td>SMTP_RCPTTOCC_ACK</td>
<td>RCPT TO CC was accepted</td>
</tr>
<tr>
<td>SMTP_RCPTTOCC_ISDONE</td>
<td>Done sending RCPT TO CC</td>
</tr>
<tr>
<td>SMTP_RCPTTOBCC_INIT</td>
<td>Preparing to send RCPT TO BCC commands</td>
</tr>
<tr>
<td>SMTP_RCPTTOBCC</td>
<td>Sending RCPT TO BCC commands</td>
</tr>
<tr>
<td>SMTP_RCPTTOBCC_ACK</td>
<td>RCPT TO BCC was accepted</td>
</tr>
<tr>
<td>SMTP_RCPTTOBCC_ISDONE</td>
<td>Done sending RCPT TO BCC</td>
</tr>
<tr>
<td>SMTP_DATA</td>
<td>Sending DATA command</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>SMTP_DATA_ACK</td>
<td>DATA command accepted</td>
</tr>
<tr>
<td>SMTP_DATA_HEADER</td>
<td>Sending message headers</td>
</tr>
<tr>
<td>SMTP_DATA_BODY_INIT</td>
<td>Preparing for message body</td>
</tr>
<tr>
<td>SMTP_DATA_BODY</td>
<td>Sending message body</td>
</tr>
<tr>
<td>SMTP_DATA_BODY_ACK</td>
<td>Message body accepted</td>
</tr>
<tr>
<td>SMTP_QUIT_INIT</td>
<td>Sending QUIT command</td>
</tr>
<tr>
<td>SMTP_QUIT</td>
<td>QUIT accepted, connection closing</td>
</tr>
</tbody>
</table>
TransportState Variable

```c
enum {
    TRANSPORT_HOME = 0,
    TRANSPORT_BEGIN,
    TRANSPORT_NAME_RESOLVE,
    TRANSPORT_OBTAIN_SOCKET,
    TRANSPORT_SECURE_SOCKET,
    TRANSPORT_SOCKET_OBTAINED,
    TRANSPORT_CLOSE
} TransportState;
```

Description

State of the transport for the SMTP Client

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSPORT_HOME = 0</td>
<td>Idle state</td>
</tr>
<tr>
<td>TRANSPORT_BEGIN</td>
<td>Preparing to make connection</td>
</tr>
<tr>
<td>TRANSPORT_NAME_RESOLVE</td>
<td>Resolving the SMTP server address</td>
</tr>
<tr>
<td>TRANSPORT_OBTAIN_SOCKET</td>
<td>Obtaining a socket for the SMTP connection</td>
</tr>
<tr>
<td>TRANSPORT_SECURE_SOCKET</td>
<td>Securing the socket for the SMTP over SSL connection</td>
</tr>
<tr>
<td>TRANSPORT_SOCKET_OBTAINED</td>
<td>SMTP connection successful</td>
</tr>
<tr>
<td>TRANSPORT_CLOSE</td>
<td>SMTP socket is closed</td>
</tr>
</tbody>
</table>
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
```

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reboot Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
</tbody>
</table>

Stack API > Reboot
Reboot Stack Members

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.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RebootTask</td>
<td>Checks for incoming traffic on port 69. Resets the PIC if a 'R' is received.</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REBOOT_PORT</td>
<td>UDP TFTP port</td>
</tr>
<tr>
<td>REBOOTSAME_SUBNET_ONLY</td>
<td>For improved security, you might want to limit reboot capabilities to only users on the same IP subnet. Define REBOOTSAME_SUBNET_ONLY to enable this access restriction.</td>
</tr>
</tbody>
</table>

Module

Reboot

Stack API > Reboot > Stack Members
RebootTask Function

```c
void RebootTask();
```

Description

Checks for incoming traffic on port 69. Resets the PIC if a 'R' is received.

Preconditions

Stack is initialized()

Returns

None

Side Effects

None

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.
REBOOT_PORT Macro

C

#define REBOOT_PORT 69       // UDP TFTP port

Description

UDP TFTP port

Stack API > Reboot > Stack Members > REBOOT_PORT Macro
REBOOT_SAME_SUBNET_ONLY Macro

```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.
Simple Network Management Protocol V2c (community) agent implementation of RFC 3416.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP Public Members</td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td>SNMP Internal Members</td>
<td>Functions and variables internal to the SNMP module</td>
</tr>
<tr>
<td>SNMP Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td>Functions</td>
<td>The following table lists functions in this documentation.</td>
</tr>
<tr>
<td>Structs, Records, Enums</td>
<td>The following table lists structs, records, enums in this documentation.</td>
</tr>
<tr>
<td>Types</td>
<td>The following table lists types in this documentation.</td>
</tr>
<tr>
<td>Variables</td>
<td>The following table lists variables in this documentation.</td>
</tr>
<tr>
<td>Macros</td>
<td>The following table lists macros in this documentation.</td>
</tr>
<tr>
<td>Files</td>
<td>The following table lists files in this documentation.</td>
</tr>
</tbody>
</table>

Files

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMPv3.c</td>
<td>- Simple Network Management Protocol (SNMP) Version 3 Agent</td>
</tr>
<tr>
<td></td>
<td>- Module for Microchip TCP/IP Stack</td>
</tr>
<tr>
<td></td>
<td>- Provides SNMPv3 API for doing stuff</td>
</tr>
<tr>
<td>File</td>
<td>Notes</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SNMPv3.h</td>
<td>- Simple Network Management Protocol (SNMP) Version 3 Agent</td>
</tr>
<tr>
<td></td>
<td>- Module for Microchip TCP/IP Stack</td>
</tr>
<tr>
<td></td>
<td>- Provides SNMPv3 API for doing stuff</td>
</tr>
<tr>
<td></td>
<td>- Reference: RFCs 3410, 3411, 3412, 3413, 3414</td>
</tr>
<tr>
<td>SNMPv3USM.c</td>
<td>- Simple Network Management Protocol (SNMP) Version 3 Agent</td>
</tr>
<tr>
<td></td>
<td>- Module for Microchip TCP/IP Stack</td>
</tr>
<tr>
<td></td>
<td>- Provides SNMPv3 API for doing stuff</td>
</tr>
<tr>
<td></td>
<td>- Reference: RFCs 3410, 3411, 3412, 3413, 3414</td>
</tr>
<tr>
<td>SNMP.c</td>
<td>- Simple Network Management Protocol (SNMP) Version 1 Agent</td>
</tr>
<tr>
<td></td>
<td>- Simple Network Management Protocol (SNMP) Version 2 community based</td>
</tr>
<tr>
<td></td>
<td>- Module for Microchip TCP/IP Stack</td>
</tr>
<tr>
<td></td>
<td>- Provides SNMP API for doing stuff</td>
</tr>
</tbody>
</table>
## Reference
- RFC 1157 (for SNMP V1)
- RFC 3416 (for SNMPv2C)

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IsSNMPv3ValidStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>_Snmpv3IsValidAuthStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>_Snmpv3IsValidInt</td>
<td>Verifies variable datatype as INT and retrieves its value.</td>
</tr>
<tr>
<td>FindOIDsFromSnmpV3Request</td>
<td>Finds number of varbinds in the varbind list received in a SNMPv3 pdu.</td>
</tr>
<tr>
<td>getSnmpV2GenTrapOid</td>
<td>Resolves generic trap code to generic trap OID.</td>
</tr>
<tr>
<td>IsSnmpV3ASNNull</td>
<td>Verifies the value type as <strong>ASN_NULL</strong>.</td>
</tr>
<tr>
<td>IsSnmpv3ValidOID</td>
<td>Populates OID type, length and oid string from the received pdu.</td>
</tr>
<tr>
<td>IsSNMPv3ValidStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>ProcessGetBulkVar</td>
<td>This routine process the SNMPv2c Get Bulk Request.</td>
</tr>
<tr>
<td>ProcessGetNextVar</td>
<td>Retrieves next node from the MIB database.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ProcessGetVar</td>
<td>Processes snmp Get request pdu.</td>
</tr>
<tr>
<td>ProcessSnmpv3MsgData</td>
<td>This routine processes the snmpv3 request and parallely creates the response pdu.</td>
</tr>
<tr>
<td>SNMPGetExactIndex</td>
<td>To search for exact index node in case of a Sequence variable.</td>
</tr>
<tr>
<td>SNMPGetTrapTime</td>
<td>Returns trap resolve get time.</td>
</tr>
<tr>
<td>SNMPIdRecrdValidation</td>
<td>Used to Restrict the access dynamic and non dynamic OID string for A particular SNMP Version.</td>
</tr>
<tr>
<td>SNMPIsValidSetLen</td>
<td>Validates the set variable data length to data type.</td>
</tr>
<tr>
<td>Snmpv3AESDecryptRxed Scoped Pdu</td>
<td>Incoming SNMPv3 scoped PDU decryption using AES decryption protocol.</td>
</tr>
<tr>
<td>Snmpv3AESEncryptResponseScoped Pdu</td>
<td>outGoing SNMPv3 scoped PDU Encryption using AES encryption protocol.</td>
</tr>
<tr>
<td>Snmpv3AuthenticateRxed Pdu For Data Integrity</td>
<td>Authenticate an incoming SNMPV3 USM PDU using MD5 or SHA</td>
</tr>
<tr>
<td>Snmpv3AuthenticateTxPdu For Data Integrity</td>
<td>Authenticate to an outgoing SNMPV3 USM PDU using MD5 or SHA</td>
</tr>
<tr>
<td>Snmpv3AuthKeyZeroing2 Hmac Buf Len 64</td>
<td>Pad zero to the authentication key localized buffer.</td>
</tr>
<tr>
<td>Snmpv3BufferPut</td>
<td>Copies BYTE data to dynamically allocated memory buffer.</td>
</tr>
<tr>
<td>Snmpv3CmprTrapSecNameAndSecLvlWithUSM Db</td>
<td>Routine to find the index of the user name in the user</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Snmpv3ComputeHMACIpadOpadForAuthLocalzedKey</td>
<td>Compute HMAC inner and outer pad for authorization localized key.</td>
</tr>
<tr>
<td>Snmpv3ComputeHmacMD5Digest</td>
<td>Compute HMAC - MD5 authentication code.</td>
</tr>
<tr>
<td>Snmpv3ComputeHmacShaDigest</td>
<td>Compute HMAC - SHA authentication code.</td>
</tr>
<tr>
<td>Snmpv3ComputeMd5HmacCode</td>
<td>Compute HMAC - MD5 authentication code.</td>
</tr>
<tr>
<td>Snmpv3ComputeShaHmacCode</td>
<td>Compute HMAC - SHA authentication code.</td>
</tr>
<tr>
<td>Snmpv3FormulateEngineID</td>
<td>Formulates the snmpEngineID for the SNMPV3 engine.</td>
</tr>
<tr>
<td>Snmpv3FreeDynAllocMem</td>
<td>Allocated dynamic memory freeing is done by this routine.</td>
</tr>
<tr>
<td>Snmpv3GetAuthEngineTime</td>
<td>Updates the snmp engine time variable 'snmpEngineTime' for the SNMPV3 engine.</td>
</tr>
<tr>
<td>Snmpv3GetBufferData</td>
<td>Reads BYTE data from dynamically allocated memory buffer.</td>
</tr>
<tr>
<td>Snmpv3GetSecurityLevel</td>
<td>Get Security level from authentication and Privacy type.</td>
</tr>
<tr>
<td>Snmpv3GetTrapSecurityLevel</td>
<td>Routine to find the report, auth and privacy flags settings in the TRAP.</td>
</tr>
<tr>
<td>Snmpv3Init</td>
<td>SNMPv3 initialization.</td>
</tr>
<tr>
<td>Snmpv3InitializeUserDataBase</td>
<td>Initialize default SNMPv3 global user database.</td>
</tr>
<tr>
<td>Snmpv3IsValidAuthStructure</td>
<td>Decode variable length</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Snmpv3IsValidInt</td>
<td>Verifies variable datatype as INT and retrieves its value.</td>
</tr>
<tr>
<td>Snmpv3MsgProcessingModelProcessPDU</td>
<td>This routine collects or populates the message processing model information from the received SNMPv3 request PDU or to the response PDU respectively.</td>
</tr>
<tr>
<td>Snmpv3Notify</td>
<td>Creates and Sends SNMPv3 TRAP pdu.</td>
</tr>
<tr>
<td>Snmpv3Pswd2LocalizedAuthKeyMD5Hashing</td>
<td>Convert MD5 Auth password to the localized Key using SNMPEngineID.</td>
</tr>
<tr>
<td>Snmpv3Pswd2LocalizedAuthKeySHAHashing</td>
<td>Convert SHA Auth password to the localized Key using SNMPEngineID.</td>
</tr>
<tr>
<td>Snmpv3ScopedPduProcessing</td>
<td>This routine collects the scoped pdu header information from the received SNMPv3 request PDU or populates to the response PDU respectively.</td>
</tr>
<tr>
<td>Snmpv3SetErrorStatus</td>
<td>Set snmpv3 error status in the response pdu.</td>
</tr>
<tr>
<td>Snmpv3TrapScopedpdu</td>
<td>TRAP PDU scoped pdu header construction.</td>
</tr>
<tr>
<td>Snmpv3UserSecurityModelProcessPDU</td>
<td>This routine collects or populates the security model parameters information from the received SNMPv3 request PDU or to the response PDU respectively.</td>
</tr>
<tr>
<td></td>
<td>AES Encryption and</td>
</tr>
</tbody>
</table>
### Snmpv3UsmAesEncryptDecryptInitVector
- decryption init vector. (RFC 3826)

### Snmpv3UsmOutMsgAuthenticationParam
- Both MD5 and SHA1 is used for the outgoing message authentication.

### Snmpv3USMOutMsgPrivParam
- SNMP USM out message uses Privacy protocol (RFC 3826)

### Snmpv3UsmSnmpEngnAuthPrivPsbdLocalization
- Convert Auth and Priv password to the localized Key using SNMPEngineID.

### Snmpv3ValidateEngineId
- Validate engine ID.

### Snmpv3ValidateSecNameAndSecLvl
- Validate security name with Security level.

### Snmpv3ValidateSecurityName
- Validate SNMPV3 user name or security name.

---

### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH_LOCALIZED_PASSWORD_KEY_LEN</td>
<td>SNMPv3 Authentication length size</td>
</tr>
<tr>
<td>INVALID_INDEX</td>
<td>This is macro INVALID_INDEX.</td>
</tr>
<tr>
<td>IS_SNMPV3_AUTH_STRUCTURE</td>
<td>This is macro IS_SNMPV3_AUTH_STRUCTURE.</td>
</tr>
<tr>
<td>MSG_AUTHORITATIVE_HEADER_LEN</td>
<td>Length of SNMPv3 authoritative msg header. (Header length = \text{Header length} (2 + 2 \text{bytes}) + \text{engineID} (\text{snmpEngnIDLength} engine boot (4 \text{bytes}) + \text{engine time (4 \text{bytes}}) + \text{security name} (\text{securityPrimitivesOfIncomingPdu + authentication parameters} (\text{snmpOutMsgAuthParamLen}) + ...))</td>
</tr>
</tbody>
</table>

- engine boot: 4 bytes
- engine time: 4 bytes
- security name: \text{securityPrimitivesOfIncomingPdu} + authentication parameters (\text{snmpOutMsgAuthParamLen})
<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSGGLOBAL_HEADER_LEN</td>
<td>Length of the SNMPv3 msg header (2 bytes)</td>
</tr>
<tr>
<td></td>
<td>• MSGID size (length of value)</td>
</tr>
<tr>
<td></td>
<td>• msgMAXSIZE (type + length of value + 4 bytes value)</td>
</tr>
<tr>
<td></td>
<td>• msg flag (type + length of value + 1 byte value)</td>
</tr>
<tr>
<td></td>
<td>• security model type (type + length of value + 1 byte value)</td>
</tr>
<tr>
<td>PRIV_LOCALIZED_PASSWORD_KEY_LEN</td>
<td>#define PRIV_LOCALIZED_PASSWORD_KEY_LEN 16</td>
</tr>
<tr>
<td>REPORT_RESPONSE</td>
<td>This is macro REPORT_RESPONSE.</td>
</tr>
<tr>
<td>SNMP_ENGINE_MAX_MSG_SIZE</td>
<td>SNMP ENGINE_MAX_MSG_SIZE is determined as the minimum of the max msg size</td>
</tr>
<tr>
<td></td>
<td>values supported among all transports available to and supported by the engine.</td>
</tr>
<tr>
<td>SNMP_MAX_MSG_SIZE</td>
<td>SNMP_MIN and MAX message size 484 bytes. As per RFC 3411 and RFC 1157 (section 4 - protocol specification) and implementation supports more than 484 whenever feasible.</td>
</tr>
<tr>
<td>SNMP_MAX_OID_LEN_MEM_USE</td>
<td>This macro will be used to avoid SNMP OID memory buffer corruption</td>
</tr>
<tr>
<td>SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE</td>
<td>This macro will be used to avoid SNMP OID memory buffer corruption</td>
</tr>
<tr>
<td>SNMP_V3</td>
<td>This is macro SNMP_V3.</td>
</tr>
<tr>
<td>SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE</td>
<td>SNMPv3 authentication localized Key length for memory validation</td>
</tr>
<tr>
<td>SNMPV3_H</td>
<td>This is macro SNMPV3_H.</td>
</tr>
<tr>
<td></td>
<td>SNMPv3 privacy key length size for memory validation</td>
</tr>
</tbody>
</table>
### Structs, Records, Enums

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AccessCtrlSubSysIsAllowed</strong></td>
<td>Applications are the typical clients of the service(s) of the Access Control Subsystem. The following primitive is provided by the Access Control Subsystem to check if access is allowed: statusInformation = -- success or errorIndication</td>
</tr>
<tr>
<td><strong>dispatcherProcessPdu</strong></td>
<td>Process Incoming Request or Notification PDU. Dispatcher provides the following primitive to pass an incoming snmp pdu to an application.</td>
</tr>
<tr>
<td><strong>dispatcherStatusInfo</strong></td>
<td>Generate Outgoing Request or Notification statusInformation = -- sendPduHandle if success -- errorIndication if failure</td>
</tr>
<tr>
<td><strong>dispatcherReturnResponsePdu</strong></td>
<td>Generate Outgoing Response The PDU Dispatcher provides the following primitive for an application to return an SNMP Response PDU to the PDU Dispatcher: result = SUCCESS or FAILURE</td>
</tr>
<tr>
<td>Procedure Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| **MsgProcModPrepareDataElements**                   | Prepare Data Elements from an Incoming SNMP Message  
The Message Processing Subsystem provides this service primitive for preparing the abstract data elements from an incoming SNMP message: result = -- SUCCESS or errorIndication                                                                                           |
| **MsgProcModPrepareOutgoingMessage**                | Prepare Outgoing SNMP Request or Notification Message  
The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Request or Notification Message                                                                                                                                 |
| **MsgProcModPrepareResponseMessage**                 | Prepare an Outgoing SNMP Response Message  
The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Response Message: result = -- SUCCESS or FAILURE                                                                                                                                     |
| **processResponsePdu**                              | Process Incoming Response PDU  
The PDU Dispatcher provides the following primitive to pass an incoming SNMP Response PDU to an application:                                                                                                           |
| **registerContextEngineID**                          | success or errorIndication                                                                                                                                                                                                                                                                                                               |
| **SecuritySysGenerateRequestMsg**                    | This is record SecuritySysGenerateRequestMsg.                                                                                                                                                                                                                                                                                           |
| **SecuritySysGenerateResponseMsg**                   | Generate a Response Message  
The Security Subsystem provides the following primitive to generate a Response message:                                                                                                                                                                                                                   |
| **StateRelease**                                    | Release State Reference Information  
All Subsystems which pass stateReference information also provide a primitive to release the memory that holds the referenced state information                                                                                                                                                 |
<p>| <strong>unregisterContextEngineID</strong>                        | This is record unregisterContextEngineID.                                                                                                                                                                                                                                                                                               |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INOUT_SNMP_PDU</td>
<td>This is type INOUT_SNMP_PDU</td>
</tr>
<tr>
<td>REPORT_FLAG_AND_SECURITY_LEVEL_FLAGS</td>
<td>This is type REPORT_FLAG_AND_SECURITY_LEVEL_FLAGS</td>
</tr>
<tr>
<td>SecuritySysProcessIncomingMsg</td>
<td>This is type SecuritySysProcessIncomingMsg</td>
</tr>
<tr>
<td>SNMP_ENGNID_OCTET_IDENTIFIER_VAL</td>
<td>The fifth octet indicates how the rest (6th and following octets) are formatted. Refer to RFC3411 section 5 Page #41</td>
</tr>
<tr>
<td>SNMPNONMIBRECDINFO</td>
<td>This is type SNMPNONMIBRECDINFO</td>
</tr>
<tr>
<td>SNMPV3_HMAC_HASH_TYPE</td>
<td>Type of hash being calculated</td>
</tr>
<tr>
<td>SNMPV3_MSG_AUTH_SEC_PARAM_RESULT</td>
<td>This is type SNMPV3_MSG_AUTH_SEC_PARAM_RESULT</td>
</tr>
<tr>
<td>SNMPV3_MSG_PRIV_SEC_PARAM_RESULT</td>
<td>This is type SNMPV3_MSG_PRIV_SEC_PARAM_RESULT</td>
</tr>
<tr>
<td>SNMPV3_PRIV_PROT_TYPE</td>
<td>This is type SNMPV3_PRIV_PROT_TYPE</td>
</tr>
<tr>
<td>SNMPV3_REQUEST_WHOLEMSG</td>
<td>This is type SNMPV3_REQUEST_WHOLEMSG</td>
</tr>
<tr>
<td>SNMPV3_RESPONSE_WHOLEMSG</td>
<td>This is type SNMPV3_RESPONSE_WHOLEMSG</td>
</tr>
<tr>
<td>snmpV3EngnUserDataBase</td>
<td>This is type snmpV3EngnUserDataBase</td>
</tr>
<tr>
<td>SNMPV3MSGDATA</td>
<td>SNMPv3</td>
</tr>
<tr>
<td>snmpV3TrapConfigDataBase</td>
<td>snmpv3 target configuration</td>
</tr>
<tr>
<td>statusInformation</td>
<td>success or errorIndication</td>
</tr>
<tr>
<td>STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL</td>
<td>This is type STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL</td>
</tr>
<tr>
<td>STD_BASED_SNMP_SECURITY_MODEL</td>
<td>Snmp Message Processing Model</td>
</tr>
<tr>
<td>STD_BASED_SNMPV3_SECURITY_LEVEL</td>
<td>This is type STD_BASED_SNMPV3_SECURITY_LEVEL</td>
</tr>
<tr>
<td>USM_SECURITY_LEVEL</td>
<td>This is type USM_SECURITY LEVEL</td>
</tr>
</tbody>
</table>

**Variables**
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>authKey_iPad</strong></td>
<td>This is variable authKey_iPad.</td>
</tr>
<tr>
<td><strong>authKey_oPad</strong></td>
<td>This is variable authKey_oPad.</td>
</tr>
<tr>
<td><strong>authoritativeSnmpEngineBoots</strong></td>
<td>The number of times that the authoritative SNMP engine has (re-)initialized itself since its <code>snmpEngineID</code> was last configured.</td>
</tr>
<tr>
<td><strong>authoritativeSnmpEngineTime</strong></td>
<td>The number of seconds since the value of the <code>authoritativeSnmpEngineBoots</code> object last changed</td>
</tr>
<tr>
<td><strong>cipher_text</strong></td>
<td>This is variable cipher_text.</td>
</tr>
<tr>
<td><strong>deciphered_text</strong></td>
<td>This is variable deciphered_text.</td>
</tr>
<tr>
<td><strong>getZeroInstance</strong></td>
<td>This variable is used for gext next request for zero instance</td>
</tr>
<tr>
<td><strong>gSnmpV3InPduWholeMsgBuf</strong></td>
<td>Dynamic memory stub and PDU details for Incoming stored PDU</td>
</tr>
<tr>
<td><strong>gSnmpV3OUTPduWholeMsgBuf</strong></td>
<td>Dynamic memory stub details and constructed outgoing stored PDU details</td>
</tr>
<tr>
<td><strong>gSNMPv3PduHeaderBuf</strong></td>
<td>Response PDU construction offset details</td>
</tr>
<tr>
<td><strong>gSNMPv3ScopedPduDataPos</strong></td>
<td>Offset to read scoped PDU data bytes for processing from dynamic memory stub</td>
</tr>
<tr>
<td><strong>gSNMPv3ScopedPduRequestBuf</strong></td>
<td>Stored request scoped pdu details</td>
</tr>
<tr>
<td><strong>gSNMPv3ScopedPduResponseBuf</strong></td>
<td>Processed response scoped pdu details</td>
</tr>
<tr>
<td><strong>gSnmpv3TrapConfigData</strong></td>
<td>SNMPv3 global configuration database to be used for trap notification</td>
</tr>
<tr>
<td><strong>gSNMPv3TrapMsgHeaderBuf</strong></td>
<td>TRAP message PDU header construction offset details</td>
</tr>
<tr>
<td><strong>gSnmpV3TrapOUTPduWholeMsgBuf</strong></td>
<td>Dynamic memory stub details and constructed trap PDU details</td>
</tr>
<tr>
<td><strong>gSNMPv3TrapScopedPduResponseBuf</strong></td>
<td>TRAP scoped PDU construction offset details</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>gSNMPV3TrapSecurityLevel</td>
<td>This is variable gSNMPV3TrapSecurityLevel.</td>
</tr>
<tr>
<td>gSnmpv3UserDBIndex</td>
<td>Index to the particular reference configured in User security model data base snmpV3UserDataBase.</td>
</tr>
<tr>
<td>gUsmStatsEngineID</td>
<td>Global variable to find out how many times SNMPv3 engine id has been validated.</td>
</tr>
<tr>
<td>hmacAuthKeyBuf</td>
<td>This is variable hmacAuthKeyBuf.</td>
</tr>
<tr>
<td>HmacMd5Digest</td>
<td>This is variable HmacMd5Digest.</td>
</tr>
<tr>
<td>HmacSHADigest</td>
<td>This is variable HmacSHADigest.</td>
</tr>
<tr>
<td>incomingPdu</td>
<td>Incoming PDU details.</td>
</tr>
<tr>
<td>incomingSnmpPDUmsgID</td>
<td>Retrieved Incoming Msg ID value from PDU.</td>
</tr>
<tr>
<td>ivEncrptKeyOut</td>
<td>This is variable ivEncrptKeyOut.</td>
</tr>
<tr>
<td>md5LocalizedAuthKey</td>
<td>This is variable md5LocalizedAuthKey.</td>
</tr>
<tr>
<td>msgSecrtyParamLenOffset</td>
<td>This is variable msgSecrtyParamLenOffset.</td>
</tr>
<tr>
<td>securityPrimitivesOfIncomingPdu</td>
<td>Incoming PDU Security primitive details.</td>
</tr>
<tr>
<td>session_key</td>
<td>This is variable session_key.</td>
</tr>
<tr>
<td>sha1LocalizedAuthKey</td>
<td>This is variable sha1LocalizedAuthKey.</td>
</tr>
<tr>
<td>snmpEngineBoots</td>
<td>The number of times that the SNMP engine has (re-)initialized itself since snmpEngineID was last configured.</td>
</tr>
<tr>
<td>snmpEngineID</td>
<td>Reserving 32 bytes for the snmpEngineID as the octet string length can vary form 5 to 32.</td>
</tr>
<tr>
<td>snmpEngineMaxMessageSize</td>
<td>The maximum message size the SNMP engine can handle.</td>
</tr>
<tr>
<td>snmpEngineMsgProcessModel</td>
<td>Type of Message processing model used. Value Maximum range (2^31-1), RFC3411</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>snmpEngineSecurityModel</td>
<td>Type of security model used. Value Maximum range (2^31-1), RFC3411</td>
</tr>
<tr>
<td>snmpEngineTime</td>
<td>The number of seconds since the value of the snmpEngineBoots object last changed</td>
</tr>
<tr>
<td>snmpEngineTimeOffset</td>
<td>Stores the time value in seconds since SNMP Engine reset</td>
</tr>
<tr>
<td>snmpEngnIDLength</td>
<td>Engine ID length of the SNMP Engine</td>
</tr>
<tr>
<td>snmpInMsgAuthParamLen</td>
<td>Incoming SNMPv3 msg authentication parameters string is 12 bytes long.</td>
</tr>
<tr>
<td>snmpInMsgAuthParamString</td>
<td>Reserving 12 bytes for the incoming SNMPv3 msg authentication parameters.</td>
</tr>
<tr>
<td>snmpInMsgPrivParamLen</td>
<td>Incoming SNMPv3 msg privacy parameters string is 8 bytes long.</td>
</tr>
<tr>
<td>snmpInMsgPrvParamStrng</td>
<td>Reserving 8 bytes for the incoming SNMPv3 msg privacy parameters.</td>
</tr>
<tr>
<td>snmpMsgBufSeekPos</td>
<td>Offset to read PDU data bytes for processing from dynamic memory stub</td>
</tr>
<tr>
<td>snmpOutMsgAuthParamLen</td>
<td>Outgoing SNMPv3 msg authentication parameters string is 12 bytes long.</td>
</tr>
<tr>
<td>snmpOutMsgAuthParamStrng</td>
<td>Reserving 12 bytes for the outgoing SNMPv3 msg authentication parameters.</td>
</tr>
<tr>
<td>snmpOutMsgPrivParamLen</td>
<td>Outgoing SNMPv3 msg privacy parameters string is 8 bytes long.</td>
</tr>
<tr>
<td>snmpOutMsgPrvParamStrng</td>
<td>Reserving 8 bytes for the outgoing SNMPv3 msg privacy parameters.</td>
</tr>
<tr>
<td>snmpResponseSecurityFlag</td>
<td>Type of Security for outgoing message in response to the incoming message.</td>
</tr>
<tr>
<td>snmpSecurityLevel</td>
<td>Type of security. noAuthNoPriv(0), AuthNoPriv(1), AuthPriv(3)</td>
</tr>
<tr>
<td>snmpTrapTimer</td>
<td>This is variable snmpTrapTimer.</td>
</tr>
<tr>
<td>snmpV3AesDecryptInitVector</td>
<td>128 Bit</td>
</tr>
<tr>
<td><strong>snmpV3AesEncryptInitVector</strong></td>
<td>128 Bit</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>snmpV3UserDataBase</strong></td>
<td>This is variable snmpV3UserDataBase.</td>
</tr>
</tbody>
</table>

**Stack API > SNMP**

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] [Index] [Home]
SNMP Public Members

The following functions and variables are available to the stack application.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERIC_TRAP_NOTIFICATION_TYPE</td>
<td>This is type GENERIC_TRAP_NOTIFICATION_TYPE.</td>
</tr>
<tr>
<td>VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE</td>
<td>This is type VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE.</td>
</tr>
<tr>
<td>SNMP_ACTION</td>
<td>This is the list of SNMP action a remote NMS can perform. This information is passed to application via callback SNMPValidateCommunity should validate the action for given community string.</td>
</tr>
<tr>
<td>COMMUNITY_TYPE</td>
<td>This is type COMMUNITY_TYPE.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMPSendTrap</td>
<td>Prepare, validate remote node which will receive trap and send trap pdu.</td>
</tr>
<tr>
<td>SNMPNotify</td>
<td>Creates and Sends TRAP pdu.</td>
</tr>
<tr>
<td>SNMPSetVar</td>
<td>This routine Set the mib variable with the requested value.</td>
</tr>
<tr>
<td>SNMPGetVar</td>
<td>Used to Get/collect OID variable information.</td>
</tr>
<tr>
<td>SNMPIsNotifyReady</td>
<td>Resolves given remoteHost IP address into MAC address.</td>
</tr>
<tr>
<td>SNMPNotifyPrepare</td>
<td>Collects trap notification info and send ARP to remote host.</td>
</tr>
<tr>
<td>To search for next index node in case of a Sequence</td>
<td></td>
</tr>
</tbody>
</table>
### SNMPGetNextIndex

variable.

### SNMPValidateCommunity

Validates community name for access control.

---

## Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SNMP_COMMUNITY_MAX_LEN</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>OID_MAX_LEN</strong></td>
<td>Change this to match your OID string length.</td>
</tr>
<tr>
<td><strong>SNMP_START_OF_VAR</strong></td>
<td>This is macro SNMP_START_OF_VAR.</td>
</tr>
<tr>
<td><strong>SNMP_END_OF_VAR</strong></td>
<td>This is macro SNMP_END_OF_VAR.</td>
</tr>
<tr>
<td><strong>SNMP_INDEX_INVALID</strong></td>
<td>This is macro SNMP_INDEX_INVALID.</td>
</tr>
<tr>
<td><strong>TRAP_TABLE_SIZE</strong></td>
<td>This is macro TRAP_TABLE_SIZE.</td>
</tr>
<tr>
<td><strong>TRAP_COMMUNITY_MAX_LEN</strong></td>
<td>This is macro TRAP_COMMUNITY_MAX_LEN.</td>
</tr>
<tr>
<td><strong>NOTIFY_COMMUNITY_LEN</strong></td>
<td>This is macro NOTIFY_COMMUNITY_LEN.</td>
</tr>
</tbody>
</table>

---

## Module

### SNMP

## Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TRAP_INFO</strong></td>
<td>This is type TRAP_INFO.</td>
</tr>
</tbody>
</table>

## Types
### SNMP_ID

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 rule when BIB was generated.

### SNMP_INDEX

This is type SNMP_INDEX.

## Unions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP_VAL</td>
<td>This is type SNMP_VAL.</td>
</tr>
</tbody>
</table>

## Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gSendTrapFlag</td>
<td>global flag to send Trap</td>
</tr>
<tr>
<td>gSetTrapSendFlag</td>
<td>#if defined(SNMP_STACK_USE_V2_TRAP)</td>
</tr>
<tr>
<td>gGenericTrapNotification</td>
<td>Global flag for Generic trap notification</td>
</tr>
<tr>
<td>gSpecificTrapNotification</td>
<td>Vendor specific trap code</td>
</tr>
<tr>
<td>gOIDCorrespondingSnmpMibID</td>
<td>Global var to store SNMP ID of var for OID received in SNMP request.</td>
</tr>
</tbody>
</table>
**C**

typedef enum {
    COLD_START = 0x0,
    WARM_START = 0x1,
    LINK_DOWN = 0x2,
    LINK_UP = 0x3,
    AUTH_FAILURE = 0x4,
    EGP_NEIGHBOR LOSS = 0x5,
    ENTERPRISE SPECIFIC = 0x6
} GENERIC_TRAP_NOTIFICATION_TYPE;

**Description**

This is type GENERIC_TRAP_NOTIFICATION_TYPE.
VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE Enumeration

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.
**SNMP_ACTION Enumeration**

```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;
```

**Description**

This is the list of SNMP action a remote NMS can perform. This information is passed to application via callback `SNMPValidateCommunity()`. Application should validate the action for given community string.

**Members**

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP_GET = 0xa0</td>
<td>Snmp GET identifier</td>
</tr>
<tr>
<td>SNMP_GET_NEXT = 0xa1</td>
<td>Snmp GET_NEXT identifier</td>
</tr>
<tr>
<td>SNMP_GET_RESPONSE = 0xa2</td>
<td>Snmp GET_RESPONSE identifier</td>
</tr>
<tr>
<td>SNMP_SET = 0xa3</td>
<td>Snmp SET identifier</td>
</tr>
<tr>
<td>SNMP_TRAP = 0xa4</td>
<td>Snmp TRAP identifier</td>
</tr>
<tr>
<td>SNMP_V2C_GET_BULK = 0xa5</td>
<td>Snmp GET_BULK identifier</td>
</tr>
<tr>
<td>SNMP_V2_TRAP = 0xa7</td>
<td>Snmp v2 Trap Identifier</td>
</tr>
<tr>
<td>SNMP_ACTION_UNKNOWN</td>
<td>Snmp requested action unknown</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>= 0</td>
<td>------------------------------</td>
</tr>
</tbody>
</table>

Stack API > SNMP > Public Members > SNMP_ACTION Enumeration

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
COMMUNITY_TYPE Enumeration

```
typedef enum {
    READ_COMMUNITY = 1,
    WRITE_COMMUNITY = 2,
    INVALID_COMMUNITY = 3
} COMMUNITY_TYPE;
```

Description

This is type COMMUNITY_TYPE.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_COMMUNITY = 1</td>
<td>Read only community</td>
</tr>
<tr>
<td>WRITE_COMMUNITY = 2</td>
<td>Read write community</td>
</tr>
<tr>
<td>INVALID_COMMUNITY = 3</td>
<td>Community invalid</td>
</tr>
</tbody>
</table>

Stack API > SNMP > Public Members > COMMUNITY_TYPE Enumeration
SNMP_VAL Union

C

typedef union {
    DWORD dword;
    WORD word;
    BYTE byte;
    BYTE v[sizeof(DWORD)];
} SNMP_VAL;

Description

This is type SNMP_VAL.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD dword;</td>
<td>double word value</td>
</tr>
<tr>
<td>WORD word;</td>
<td>word value</td>
</tr>
<tr>
<td>BYTE byte;</td>
<td>byte value</td>
</tr>
<tr>
<td>BYTE v[sizeof(DWORD)];</td>
<td>byte array</td>
</tr>
</tbody>
</table>
TRAP_INFO Structure

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;

Description

This is type TRAP_INFO.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE communityLen;</td>
<td>Community name length</td>
</tr>
<tr>
<td>char community[TRAP_COMMUNITY_MAX_LEN];</td>
<td>Community name array</td>
</tr>
<tr>
<td>IP_ADDR IPAddress;</td>
<td>IP address to which trap to be sent</td>
</tr>
<tr>
<td>unsigned int bEnabled : 1;</td>
<td>Trap enabled flag</td>
</tr>
</tbody>
</table>

Stack API > SNMP > Public Members > TRAP_INFO Structure
gSendTrapFlag Variable

```c
BYTE gSendTrapFlag = FALSE;
```

**Description**

Global flag to send Trap

**Stack API** > **SNMP** > **Public Members** > **gSendTrapFlag Variable**
#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 Variable

BYTE gGenericTrapNotification = ENTERPRISE_SPECIFIC;

Description

Global flag for Generic trap notification
gSpecificTrapNotification Variable

```c
BYTE gSpecificTrapNotification = VENDOR_TRAP_DEFAULT;
```

Description

Vendor specific trap code
gOIDCorrespondingSnmpMibID Variable

C

BYTE gOIDCorrespondingSnmpMibID = MICROCHIP;

Description

Global var to store SNMP ID of var for OID received in SNMP request.
SNMPSendTrap Function

C

```c
void SNMPSendTrap();
```

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).

Preconditions

If application defined event occurs to send the trap.

Returns

None.

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 `SNMPIsNotifyReady()` returns FALSE, this routine times out in 5 seconds. This routine should be modified according to event occured and should update corrsponding OID and notification type to the trap pdu.
SNMPNotify Function

```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

For ASCII STR trap VAL(argument) contains the pointer address of the string variable.

Preconditions

SNMPIsNotifyReady() is already called and returned TRUE.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>SNMP var ID that is to be used in notification</td>
</tr>
<tr>
<td>val</td>
<td>Value of var. Only value of BYTE, WORD or DWORD can be sent.</td>
</tr>
<tr>
<td>index</td>
<td>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</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>if SNMP notification was successful sent. This does not guarantee that remoteHost recieved it.</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FALSE</td>
<td>Notification sent failed.</td>
</tr>
<tr>
<td></td>
<td>This would fail under following conditions</td>
</tr>
<tr>
<td></td>
<td>1) Given SNMP_BIB_FILE does not exist in MPFS</td>
</tr>
<tr>
<td></td>
<td>2) Given var does not exist.</td>
</tr>
<tr>
<td></td>
<td>3) Previously given agentID does not exist</td>
</tr>
<tr>
<td></td>
<td>4) Data type of given var is unknown</td>
</tr>
<tr>
<td></td>
<td>only possible if MPFS itself was corrupted.</td>
</tr>
</tbody>
</table>

**Remarks**

This would fail if there were not UDP socket to open.

[Stack API] > [SNMP] > [Public Members] > [SNMPNotify Function]
SNMPSetVar Function

```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.

Preconditions

ProcessVariables() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>Variable id whose value is to be set</td>
</tr>
<tr>
<td>ref</td>
<td>Variable reference used to transfer multi-byte data. 0 if first byte is set otherwise nonzero value to indicate corresponding byte being set.</td>
</tr>
<tr>
<td>val</td>
<td>Up to 4 byte data value. If var data type is BYTE, variable value is in val-&gt;byte. If var data type is WORD, variable value is in val-&gt;word. If var data type is DWORD, variable value is in val-&gt;dword. If var data type is IP_ADDRESS, COUNTER32, or GAUGE32, value is in val-&gt;dword. If var data type is OCTET_STRING, ASCII_STRING value is in val-&gt;byte; multi-byte transfer will be performed to transfer remaining bytes of data.</td>
</tr>
</tbody>
</table>

Return Values
Return Values | Description
---|---
TRUE | if it is OK to set more byte(s).
FALSE | if otherwise.

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.

Stack API > SNMP > Public Members > SNMPSetVar Function
**SNMPGetVar Function**

```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.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>Variable id whose value is to be returned</td>
</tr>
<tr>
<td>index</td>
<td>Index of variable that should be transferred</td>
</tr>
<tr>
<td>ref</td>
<td>Variable reference used to transfer multi-byte data It is always <code>SNMP_START_OF_VAR</code> when very first byte is requested. Otherwise, use this as a reference to keep track of multi-byte transfers.</td>
</tr>
<tr>
<td>val</td>
<td>Pointer to up to 4 byte buffer. If var data type is BYTE, transfer data in val-&gt;byte If var data type is WORD, transfer data in val-&gt;word If var data type is DWORD, transfer data in val-&gt;dword If var data type is IP_ADDRESS, transfer data in val-&gt;v[] or val-&gt;dword If var data type is COUNTER32, TIME_TICKS or GAUGE32, transfer data in val-&gt;dword If var data type is ASCII_STRING or OCTET_STRING transfer data in val-&gt;byte using multi-byte transfer mechanism.</td>
</tr>
</tbody>
</table>
### Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If a value exists for given variable at given index.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Otherwise.</td>
</tr>
</tbody>
</table>

### Remarks

None.

Stack API > SNMP > Public Members > SNMPGetVar Function

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
SNMPIsNotifyReady Function

C

```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.

Preconditions

SNMPNotifyPrepare() is already called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remoteHost</td>
<td>Pointer to remote Host IP address</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If remoteHost IP address is resolved and SNMPNotify may be called.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If remoteHost IP address is not resolved.</td>
</tr>
</tbody>
</table>

Remarks

This would fail if there were not UDP socket to open.
SNMPNotifyPrepare Function

```c
void SNMPNotifyPrepare(
    IP_ADDR* remoteHost,
    char* community,
    BYTE communityLen,
    SNMP_ID agentIDVar,
    BYTE notificationCode,
    DWORD timestamp
);
```

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.

Preconditions

- **SNMPInit()** is already called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remoteHost</td>
<td>pointer to remote Host IP address</td>
</tr>
<tr>
<td>community</td>
<td>Community string to use to notify</td>
</tr>
<tr>
<td>communityLen</td>
<td>Community string length</td>
</tr>
<tr>
<td>agentIDVar</td>
<td>System ID to use identify this agent</td>
</tr>
<tr>
<td>notificationCode</td>
<td>Notification Code to use</td>
</tr>
<tr>
<td>timestamp</td>
<td>Notification timestamp in 100th of second.</td>
</tr>
</tbody>
</table>

Returns
None

Remarks

This is first of series of functions to complete SNMP notification.
SNMPGetNextIndex Function

```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.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>Variable id whose value is to be returned</td>
</tr>
<tr>
<td>index</td>
<td>Next Index of variable that should be transferred</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If a next index value exists for given variable at given index and index parameter contains next valid index.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Otherwise.</td>
</tr>
</tbody>
</table>

**Remarks**
Only sequence index needs to be handled in this function.
SNMPValidateCommunity Function

C

```c
BYTE SNMPValidateCommunity(
    BYTE * community
);
```

Description

This function validates the community name for the mib access to NMS. The snmp community name received in the request pdu is validated for read and write community names. The agent gives an access to the mib variables only if the community matches with the predefined values. This routine also sets a global flag to send trap if authentication failure occurs.

Preconditions

SNMPInit is already called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>community</td>
<td>Pointer to community string as sent by NMS.</td>
</tr>
</tbody>
</table>

Returns

This routine returns the community validation result as READ_COMMUNITY or WRITE_COMMUNITY or INVALID_COMMUNITY

Remarks

This is a callback function called by module. User application must implement this function and verify that community matches with predefined value. This validation occurs for each NMS request.
SNMP_ID Type

C

```cpp
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.
SNMP_INDEX Type

typedef BYTE SNMP_INDEX;

Description

This is type SNMP_INDEX.
SNMP_COMMUNITY_MAX_LEN Macro

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.
#define OID_MAX_LEN (SNMP_MAX_OID_LEN_MEM_USE + 1)

Description

Change this to match your OID string length.
SNMP_START_OF_VAR Macro

C

#define SNMP_START_OF_VAR (0)

Description

This is macro SNMP_START_OF_VAR.
SNMP_END_OF_VAR Macro

```c
#define SNMP_END_OF_VAR (0xff)
```

Description

This is macro SNMP_END_OF_VAR.
SNMP_INDEX_INVALID Macro

```
C
#define SNMP_INDEX_INVALID (0xff)
```

Description

This is macro SNMP_INDEX_INVALID.
TRAP_TABLE_SIZE Macro

C

#define TRAP_TABLE_SIZE (2)

Description

This is macro TRAP_TABLE_SIZE.
TRAP_COMMUNITY_MAX_LEN Macro

```
#define TRAP_COMMUNITY_MAX_LEN (SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE+1)
```

Description

This is macro TRAP_COMMUNITY_MAX_LEN.
NOTIFY_COMMUNITY_LEN Macro

#define NOTIFY_COMMUNITY_LEN (SNMP_COMMUNITY_MAX_LEN)

Description

This is macro NOTIFY_COMMUNITY_LEN.
SNMP Internal Members

The following functions and variables are designated as internal to the SNMP module.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_TYPE</td>
<td></td>
</tr>
<tr>
<td>SNMP_ERR_STATUS</td>
<td></td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SNMPDuplexInit</td>
<td>Prepare for full duplex transfer.</td>
</tr>
<tr>
<td>SNMPGet</td>
<td>Read byte from snmp udp socket rx buffer.</td>
</tr>
<tr>
<td>_SNMPPut</td>
<td>Copy byte to tx buffer.</td>
</tr>
<tr>
<td>FindOIDsInRequest</td>
<td>Finds number of varbinds in the varbind list received in a pdu.</td>
</tr>
<tr>
<td>GetDataTypeInfo</td>
<td>Get ASN data type info.</td>
</tr>
<tr>
<td>IsASNNull</td>
<td>Verifies the value type as ASN_NULL.</td>
</tr>
<tr>
<td>SetErrorStatus</td>
<td>Set snmp error status in the response pdu.</td>
</tr>
<tr>
<td>IsValidLength</td>
<td>Retrieves the packet length and actual pdu length.</td>
</tr>
<tr>
<td>getNextLeaf</td>
<td>Searches for the next leaf node in the MIP tree.</td>
</tr>
<tr>
<td>GetOIDStringByAddr</td>
<td>Get OID string from MPFS using the node address.</td>
</tr>
<tr>
<td></td>
<td>Get complete notification variable OID string</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>GetOIDStringByID</td>
<td>from MPFS using var id.</td>
</tr>
<tr>
<td>IsValidCommunity</td>
<td>Verifies for the community string datatype and the max community name and length, this agent can process.</td>
</tr>
<tr>
<td>IsValidInt</td>
<td>Verifies variable datatype as INT and retrieves its value.</td>
</tr>
<tr>
<td>IsValidOID</td>
<td>Populates OID type, length and oid string from the received pdu.</td>
</tr>
<tr>
<td>IsValidPDU</td>
<td>Verifies for the snmp request type.</td>
</tr>
<tr>
<td>IsValidStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>OIDLookup</td>
<td>To search and validate whether the requested OID is in the MIB database.</td>
</tr>
<tr>
<td>ProcessGetSetHeader</td>
<td>Validates the received udp packet Get/Set request header.</td>
</tr>
<tr>
<td>ProcessHeader</td>
<td>Validates the received udp packet Snmp header.</td>
</tr>
<tr>
<td>ProcessSetVar</td>
<td>Processes snmp Set request pdu.</td>
</tr>
<tr>
<td>ProcessVariables</td>
<td>This routine processes the snmp request and parallely creates the response pdu.</td>
</tr>
<tr>
<td>ReadMIBRecord</td>
<td>Get OID string from MPFS using the node address.</td>
</tr>
<tr>
<td>SNMPCheckIfPvtMibObjRequested</td>
<td>To find whether requested OID is only for private access.</td>
</tr>
</tbody>
</table>

**Macros**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SNMPGetTxOffset</td>
<td>This is macro _SNMPGetTxOffset.</td>
</tr>
<tr>
<td>_SNMPSetTxOffset</td>
<td>This is macro _SNMPSetTxOffset.</td>
</tr>
<tr>
<td>AGENT_NOTIFY_PORT</td>
<td>This is macro AGENT_NOTIFY_PORT.</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>ASN_INT</td>
<td>This is macro ASN_INT.</td>
</tr>
<tr>
<td>ASN_NULL</td>
<td>This is macro ASN_NULL.</td>
</tr>
<tr>
<td>ASN_OID</td>
<td>This is macro ASN_OID.</td>
</tr>
<tr>
<td>DATA_TYPE_TABLE_SIZE</td>
<td></td>
</tr>
<tr>
<td>GET_BULK_REQUEST</td>
<td>This is macro GET_BULK_REQUEST.</td>
</tr>
<tr>
<td>GET_NEXT_REQUEST</td>
<td>This is macro GET_NEXT_REQUEST.</td>
</tr>
<tr>
<td>GET_REQUEST</td>
<td></td>
</tr>
<tr>
<td>GET_RESPONSE</td>
<td>This is macro GET_RESPONSE.</td>
</tr>
<tr>
<td>IS_AGENT_PDU</td>
<td>This is macro IS_AGENT_PDU.</td>
</tr>
<tr>
<td>IS ASN_INT</td>
<td>This is macro IS ASN_INT.</td>
</tr>
<tr>
<td>IS ASN_NULL</td>
<td>This is macro IS ASN_NULL.</td>
</tr>
<tr>
<td>IS_GET_NEXT_REQUEST</td>
<td>This is macro IS_GET_NEXT_REQUEST.</td>
</tr>
<tr>
<td>IS_GET_REQUEST</td>
<td>This is macro IS_GET_REQUEST.</td>
</tr>
<tr>
<td>IS_GET_RESPONSE</td>
<td>This is macro IS_GET_RESPONSE.</td>
</tr>
<tr>
<td>IS OCTET_STRING</td>
<td>This is macro IS OCTET_STRING.</td>
</tr>
<tr>
<td>IS_OID</td>
<td>This is macro IS_OID.</td>
</tr>
<tr>
<td>IS_SET_REQUEST</td>
<td>This is macro IS_SET_REQUEST.</td>
</tr>
<tr>
<td>IS_STRUCTURE</td>
<td></td>
</tr>
<tr>
<td>IS_TRAP</td>
<td>This is macro IS_TRAP.</td>
</tr>
<tr>
<td>OCTET_STRING</td>
<td>This is macro OCTET_STRING.</td>
</tr>
<tr>
<td>SET_REQUEST</td>
<td>This is macro SET_REQUEST.</td>
</tr>
<tr>
<td>SNMP_AGENT_PORT</td>
<td></td>
</tr>
</tbody>
</table>
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.

This is macro SNMP_COUNTER32.

This is macro SNMP_GAUGE32.

This is macro SNMP_IP_ADDR.

This is macro SNMP_NMS_PORT.

This is macro SNMP_NSAP_ADDR.

This is macro SNMP_OPAQUE.

This is macro SNMP_TIME_TICKS.

This is macro SNMP_V1.

This is macro SNMP_V2C.

This is macro TRAP.

Module

**SNMP**

Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_TYPE_INFO</td>
<td></td>
</tr>
<tr>
<td>OID_INFO</td>
<td></td>
</tr>
<tr>
<td>PDU_INFO</td>
<td></td>
</tr>
<tr>
<td>reqVarErrStatus</td>
<td></td>
</tr>
<tr>
<td>SNMP_NOTIFY_INFO</td>
<td></td>
</tr>
</tbody>
</table>
### Unions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX_INFO</td>
<td></td>
</tr>
<tr>
<td>MIB_INFO</td>
<td></td>
</tr>
<tr>
<td>SNMP_STATUS</td>
<td></td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appendZeroToOID</td>
<td>global flag to modify OID by appending zero</td>
</tr>
<tr>
<td>dataTypeTable</td>
<td>ASN format datatype for snmp v1 and v2c</td>
</tr>
<tr>
<td>hMPFS</td>
<td>MPFS file handler</td>
</tr>
<tr>
<td>SNMPAgentSocket</td>
<td>Snmp udp socket</td>
</tr>
<tr>
<td>SNMPNotifyInfo</td>
<td>notify info for trap</td>
</tr>
<tr>
<td>snmpReqVarErrStatus</td>
<td>vars from req list processing err status</td>
</tr>
<tr>
<td>SNMPRxOffset</td>
<td>Snmp udp buffer rx offset</td>
</tr>
<tr>
<td>SNMPStatus</td>
<td>MIB file access status</td>
</tr>
<tr>
<td>SNMPTxOffset</td>
<td>Snmp udp buffer tx offset</td>
</tr>
<tr>
<td>trapInfo</td>
<td>Initialize trap table with no entries.</td>
</tr>
</tbody>
</table>
_SNMPDuplexInit Function

C

```c
void _SNMPDuplexInit(
    UDP_SOCKET socket
);
```

Description

As we process SNMP variables, we will prepare response on-the-fly creating full duplex transfer. Current MAC layer does not support full duplex transfer, so SNMP needs to manage its own full duplex connection. Prepare for full duplex transfer. Set the Tx and Rx offset to start of the buffer.

Preconditions

`SNMPTask()` is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>socket</code></td>
<td>An active udp socket for which tx and rx offset to be set.</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

This routine should be called for every new snmp packet received.
Description

This function reads a single byte from the currently active UDP socket, receive buffer, while incrementing the buffer offset from where the next byte will be read.

Preconditions

SNMPTask() is called. A active udp socket is available to read from.

Returns

None.

Remarks

None.

Stack API > SNMP > Internal Members > __SNMPGet Function
_SNMPGetTxOffset Macro

C

#define _SNMPGetTxOffset SNMPTxOffset

Description

This is macro _SNMPGetTxOffset.
void _SNMPPut(BYTE v);

Description

This function writes a single byte to the currently active UDP socket, transmit buffer, while incrementing the buffer offset for the next write operation.

Preconditions

SNMPTask() is called. A active udp socket is availabe to tx from.

Returns

None.

Remarks

None.
```c
#define _SNMPSetTxOffset(o) (SNMPTxOffset = o)
```
## AGENT_NOTIFY_PORT Macro

```
#define AGENT_NOTIFY_PORT (0xfffe)
```

### Description

This is macro `AGENT_NOTIFY_PORT`.
appendZeroToOID Variable

```c
BYTE appendZeroToOID = FALSE;
```

**Description**

Global flag to modify OID by appending zero

Stack API > SNMP > Internal Members > appendZeroToOID Variable
ASN_INT Macro

C
#define ASN_INT (0x02u)

Description

This is macro ASN_INT.

Stack API > SNMP > Internal Members > ASN_INT Macro
ASN_NULL Macro

```c
#define ASN_NULL (0x05u)
```

Description

This is macro ASN_NULL.
ASN_OID Macro

```
#define ASN_OID (0x06u)
```

Description

This is macro ASN_OID.

Stack API > SNMP > Internal Members > ASN_OID Macro
## DATA_TYPE Enumeration

```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

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT8_VAL = 0x00</td>
<td>8 bit integer value</td>
</tr>
<tr>
<td>INT16_VAL = 0x01</td>
<td>16 bit integer value</td>
</tr>
<tr>
<td>INT32_VAL = 0x02</td>
<td>32 bit integer value</td>
</tr>
<tr>
<td>BYTE_ARRAY = 0x03</td>
<td>Array of bytes</td>
</tr>
<tr>
<td>ASCII_STRING = 0x04</td>
<td>Ascii string type</td>
</tr>
<tr>
<td>IP_ADDRESS = 0x05</td>
<td>IP address variable</td>
</tr>
<tr>
<td>COUNTER32 = 0x06</td>
<td>32 bit counter variable</td>
</tr>
<tr>
<td>TIME_TICKS_VAL = 0x07</td>
<td>Timer value counter variable</td>
</tr>
<tr>
<td>GAUGE32 = 0x08</td>
<td>32 bit guage variable</td>
</tr>
<tr>
<td>OID_VAL = 0x09</td>
<td>Object id value var</td>
</tr>
<tr>
<td>DATA_TYPE_UNKNOWN</td>
<td>Unknown data type</td>
</tr>
</tbody>
</table>
**DATA_TYPE_INFO Structure**

```c
typedef struct {
    BYTE asnType;
    BYTE asnLen;
} DATA_TYPE_INFO;
```

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE asnType;</td>
<td>ASN data type</td>
</tr>
<tr>
<td>BYTE asnLen;</td>
<td>ASN data length</td>
</tr>
</tbody>
</table>

[Stack API] > [SNMP] > [Internal Members] > [DATA_TYPE_INFO Structure]
DATA_TYPE_TABLE_SIZE Macro

```
#define DATA_TYPE_TABLE_SIZE (sizeof(dataTypeTable)/sizeof(dataTypeTable))
```

Stack API > SNMP > Internal Members > DATA_TYPE_TABLE_SIZE Macro
**dataTypeTable Variable**

| C-ROM | DATA_TYPE_INFO | dataTypeTable[] = { { ASN_INT, 1 }, { ASN_INT, 2 }},

**Description**

ASN format datatype for snmp v1 and v2c

---

Stack API > SNMP > Internal Members > dataTypeTable Variable

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
FindOIDsInRequest Function

C

```c
static BYTE FindOIDsInRequest(
    WORD pdulen
);
```

Description

This routine is used to find the number of OIDs requested in the received snmp pdu.

Preconditions

ProcessVariables() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdulen</td>
<td>Length of snmp pdu request received.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varCount</td>
<td>Number of OIDs found in a pdu request.</td>
</tr>
</tbody>
</table>

Remarks

None.
GET_BULK_REQUEST Macro

```c
#define GET_BULK_REQUEST (0xa5)
```

Description

This is macro GET_BULK_REQUEST.
GET_NEXT_REQUEST Macro

C

#define GET_NEXT_REQUEST (0xa1)

Description

This is macro GET_NEXT_REQUEST.
GET_REQUEST Macro

C

#define GET_REQUEST (0xa0)
GET_RESPONSE Macro

C

#define GET_RESPONSE (0xa2)

Description

This is macro GET_RESPONSE.
hMPFS Variable

```c
MPFS_HANDLE hMPFS;
```

**Description**

MPFS file handler

Stack API > SNMP > Internal Members > hMPFS Variable
INDEX_INFO Union

```c
typedef union {
    struct {
        unsigned int bIsOID : 1;
    } Flags;
    BYTE Val;
} INDEX_INFO;
```

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int bIsOID : 1;</td>
<td>value is OID/index int flag</td>
</tr>
<tr>
<td>BYTE Val;</td>
<td>value is OID/index byte flag</td>
</tr>
</tbody>
</table>

Stack API > SNMP > Internal Members > INDEX_INFO Union
IS_AGENT_PDU Macro

C

```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.
IS_ASN_INT Macro

```
#define IS_ASN_INT(a) (a==ASN_INT)
```

Description

This is macro IS_ASN_INT.
IS_ASN_NULL Macro

C

#define IS_ASN_NULL(a) (a==ASN_NULL)

Description

This is macro IS_ASN_NULL.
IS_GET_NEXT_REQUEST Macro

```c
#define IS_GET_NEXT_REQUEST(a) (a==GET_NEXT_REQUEST)
```

Description

This is macro IS_GET_NEXT_REQUEST.
**IS_GET_REQUEST Macro**

```c
#define IS_GET_REQUEST(a) (a==GET_REQUEST)
```

**Description**

This is macro IS_GET_REQUEST.
IS_GET_RESPONSE Macro

C

#define IS_GET_RESPONSE(a) (a==GET_RESPONSE)

Description

This is macro IS_GET_RESPONSE.
IS_OCTET_STRING Macro

C

#define IS_OCTET_STRING(a) (a==OCTET_STRING)

Description

This is macro IS_OCTET_STRING.

Stack API > SNMP > Internal Members > IS_OCTET_STRING Macro
IS_OID Macro

```c
#define IS_OID(a) (a==ASN_OID)
```

Description

This is macro IS_OID.
GetDataTypeInfo Function

```c
BOOL GetDataTypeInfo(
    DATA_TYPE dataType,
    DATA_TYPE_INFO *info
);
```

Description

This routine validates the SNMP GEt dtat type and collects ASN data type value to the info.

Preconditions

ProcessHeader() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataType</td>
<td>SNMP pdu data type info.</td>
</tr>
<tr>
<td>info</td>
<td>matching data type info w.r.t dataTypeTable</td>
</tr>
</tbody>
</table>

Returns

TRUE - success FALSE - failure

Remarks

None.
IS_SET_REQUEST Macro

```c
#define IS_SET_REQUEST(a) (a==SET_REQUEST)
```

Description

This is macro IS_SET_REQUEST.
### IS_STRUCTURE Macro

```
# define IS_STRUCTURE(a) (a == STRUCTURE)
```

**Stack API > SNMP > Internal Members > IS_STRUCTURE Macro**

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
IS_TRAP Macro

C

#define IS_TRAP(a) (a==TRAP)

Description

This is macro IS_TRAP.
IsASNNull Function

C

```c
static BOOL IsASNNull();
```

Description

For Get, Get_NEXT, Get_Bulk snmp request, the var bind the value data type should be `ASN_NULL` and value field must be NULL and . This routine verifies the data type and value fields in the received requests. The SET request, the value data type can not be `ASN_NULL`, otherwise the snmp request is not processed.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If value type is <code>ASN_NULL</code> and value is NULL.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If data type and value is other than <code>ASN_NULL</code> and NULL resp.</td>
</tr>
</tbody>
</table>

Remarks

None.
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

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int bIsDistantSibling : 1;</td>
<td>Object have distant sibling node</td>
</tr>
<tr>
<td>unsigned int bIsConstant : 1;</td>
<td>Object is constant</td>
</tr>
<tr>
<td>unsigned int bIsSequence : 1;</td>
<td>Object is sequence</td>
</tr>
<tr>
<td>unsigned int bIsSibling : 1;</td>
<td>Sibling node flag</td>
</tr>
<tr>
<td>unsigned int bIsParent : 1;</td>
<td>Node is parent flag</td>
</tr>
<tr>
<td>unsigned int bIsEditable : 1;</td>
<td>Node is editable flag</td>
</tr>
<tr>
<td>unsigned int bIsAgentID : 1;</td>
<td>Node have agent id flag</td>
</tr>
<tr>
<td>unsigned int bIsIDPresent : 1;</td>
<td>Id present flag</td>
</tr>
<tr>
<td>BYTE Val;</td>
<td>MIB Obj info as byte value</td>
</tr>
</tbody>
</table>
OCTET_STRING Macro

```c
#define OCTET_STRING (0x04u)
```

Description

This is macro OCTET_STRING.
## OID_INFO Structure

```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

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD hNode;</td>
<td>Node location in the mib</td>
</tr>
<tr>
<td>BYTE oid;</td>
<td>Object Id</td>
</tr>
<tr>
<td>MIB_INFO nodeInfo;</td>
<td>Node info</td>
</tr>
<tr>
<td>DATA_TYPE dataType;</td>
<td>Data type</td>
</tr>
<tr>
<td>SNMP_ID id;</td>
<td>Snmp Id</td>
</tr>
<tr>
<td>WORD_VAL dataLen;</td>
<td>Data length</td>
</tr>
<tr>
<td>DWORD hData;</td>
<td>Data</td>
</tr>
<tr>
<td>DWORD hSibling;</td>
<td>Sibling info</td>
</tr>
<tr>
<td>DWORD hChild;</td>
<td>Child info</td>
</tr>
<tr>
<td>BYTE index;</td>
<td>Index of object</td>
</tr>
<tr>
<td>BYTE indexLen;</td>
<td>Index length</td>
</tr>
</tbody>
</table>
PDU_INFO Structure

C

typedef struct {
    DWORD_VAL requestID;
    BYTE nonRepeators;
    BYTE maxRepetitions;
    BYTE pduType;
    BYTE errorStatus;
    BYTE erroIndex;
    BYTE snmpVersion;
    WORD pduLength;
} PDU_INFO;

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD_VAL requestID;</td>
<td>Snmp request id</td>
</tr>
<tr>
<td>BYTE nonRepeators;</td>
<td>non repeaters in the request</td>
</tr>
<tr>
<td>BYTE maxRepetitions;</td>
<td>max repeaters in the request</td>
</tr>
<tr>
<td>BYTE pduType;</td>
<td>Snmp pdu type</td>
</tr>
<tr>
<td>BYTE errorStatus;</td>
<td>Pdu error status</td>
</tr>
<tr>
<td>BYTE erroIndex;</td>
<td>Pdu error Index</td>
</tr>
<tr>
<td>BYTE snmpVersion;</td>
<td>Snmp version</td>
</tr>
<tr>
<td>WORD pduLength;</td>
<td>Pdu length</td>
</tr>
</tbody>
</table>

Stack API > SNMP > Internal Members > PDU_INFO Structure
reqVarErrStatus Structure

C

```c
typedef struct {
    WORD noSuchObjectErr;
    WORD noSuchNameErr;
    WORD noSuchInstanceErr;
    WORD endOfMibViewErr;
} reqVarErrStatus;
```

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD noSuchObjectErr;</td>
<td>Var list no such obj errors flags</td>
</tr>
<tr>
<td>WORD noSuchNameErr;</td>
<td>Var list no such name error</td>
</tr>
<tr>
<td>WORD noSuchInstanceErr;</td>
<td>Var list no such instance error</td>
</tr>
<tr>
<td>WORD endOfMibViewErr;</td>
<td>Var list end of mib view error</td>
</tr>
</tbody>
</table>
SET_REQUEST Macro

```c
#define SET_REQUEST (0xa3)
```

Description

This is macro SET_REQUEST.

Stack API > SNMP > Internal Members > SET_REQUEST Macro
SetErrorStatus Function

```c
void SetErrorStatus(
    WORD errorStatusOffset,
    WORD errorIndexOffset,
    SNMP_ERR_STATUS errorStatus,
    BYTE errorIndex
);
```

**Description**

This routine processes the received snmp Get request pdu for the variable binding in the request and also creates the response pdu.

**Preconditions**

ProcessVariables() is called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorStatusOffset</td>
<td>Offset to update error status in Response Tx pdu</td>
</tr>
<tr>
<td>errorIndexOffset</td>
<td>Offset to update error index</td>
</tr>
<tr>
<td>errorStatus</td>
<td>Snmp process error to be updated in response.</td>
</tr>
<tr>
<td>errorIndex</td>
<td>Index of the request varbind in the var bind list for which error status is to be updated.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**Remarks**

None.
SNMP_AGENT_PORT Macro

```c
#define SNMP_AGENT_PORT (161)
```
SNMP_BIB_FILE_NAME Macro

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.
SNMP_COUNTER32 Macro

```c
#define SNMP_COUNTER32 (0x41)
```

Description

This is macro SNMP_COUNTER32.
# SNMP_ERR_STATUS Enumeration

**C**

```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_ResourceUnavailable,
    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

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP_NO_ERR = 0</td>
<td>Snmp no error</td>
</tr>
<tr>
<td>SNMP_TOO_BIG</td>
<td>Value too big error</td>
</tr>
<tr>
<td>SNMP_NO_SUCH_NAME</td>
<td>No such name in MIB error</td>
</tr>
<tr>
<td>SNMP_BAD_VALUE</td>
<td>Not assignable value for the var error</td>
</tr>
<tr>
<td>SNMP_READ_ONLY</td>
<td>Read only variable, write not allowed err</td>
</tr>
<tr>
<td>SNMP_GEN_ERR</td>
<td>Snmp gen error</td>
</tr>
<tr>
<td>SNMP_NO_ACCESS</td>
<td>Access to modify or read not granted err</td>
</tr>
<tr>
<td>SNMP_WARNING</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>SNMP_WRONG_TYPE</td>
<td>Variable data type wrong error</td>
</tr>
<tr>
<td>SNMP_WRONG_LENGTH</td>
<td>Wrong data length error</td>
</tr>
<tr>
<td>SNMP_WRONG_ENCODING</td>
<td>Wrong encoding error</td>
</tr>
<tr>
<td>SNMP_WRONG_VALUE</td>
<td>Wrong value for the var type</td>
</tr>
<tr>
<td>SNMP_NO_CREATION</td>
<td>No creating error</td>
</tr>
<tr>
<td>SNMP_INCONSISTENT_VAL</td>
<td>Inconsistent value error</td>
</tr>
<tr>
<td>SNMP_RESOURCE_UNAVAILABLE</td>
<td>Resource unavailability error</td>
</tr>
<tr>
<td>SNMP_COMMIT_FAILED</td>
<td>Modification update failed error</td>
</tr>
<tr>
<td>SNMP_UNDO_FAILED</td>
<td>Modification undo failed</td>
</tr>
<tr>
<td>SNMP_AUTH_ERROR</td>
<td>Authorization failed error</td>
</tr>
<tr>
<td>SNMP_NOT_WRITABLE</td>
<td>Variable read only</td>
</tr>
<tr>
<td>SNMP_INCONSISTENT_NAME</td>
<td>Inconsistent name</td>
</tr>
<tr>
<td>SNMP_NO_SUCH_OBJ = 128</td>
<td>No such object error</td>
</tr>
<tr>
<td>SNMP_NO_SUCH_INSTANCE = 129</td>
<td>No such instance error</td>
</tr>
<tr>
<td>SNMP_END_OF_MIB_VIEW = 130</td>
<td>Reached to end of mib error</td>
</tr>
</tbody>
</table>
### SNMP_GAUGE32 Macro

```c
#define SNMP_GAUGE32 (0x42)
```

#### Description

This is macro SNMP_GAUGE32.
SNMP_IP_ADDR Macro

C

#define SNMP_IP_ADDR (0x40)

Description

This is macro SNMP_IP_ADDR.
SNMP_NMS_PORT Macro

C

#define SNMP_NMS_PORT (162)

Description

This is macro SNMP_NMS_PORT.
## SNMP_NOTIFY_INFO Structure

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char community[NOTIFY_COMMUNITY_LEN];</td>
<td>Community name array</td>
</tr>
<tr>
<td>BYTE communityLen;</td>
<td>Community name length</td>
</tr>
<tr>
<td>SNMP_ID agentIDVar;</td>
<td>Agent id for trap identification</td>
</tr>
<tr>
<td>BYTE notificationCode;</td>
<td>Trap notification code</td>
</tr>
<tr>
<td>UDP_SOCKET socket;</td>
<td>Udp socket number</td>
</tr>
<tr>
<td>DWORD_VAL timestamp;</td>
<td>Time stamp for trap</td>
</tr>
<tr>
<td>SNMP_ID trapIDVar;</td>
<td>SNMPV2 specific trap</td>
</tr>
</tbody>
</table>

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
SNMP_NSAP_ADDR Macro

```
#define SNMP_NSAP_ADDR (0x45)
```

Description

This is macro SNMP_NSAP_ADDR.
IsValidLength Function

C

BYTE IsValidLength(
    WORD * len
);

Description

Checks current packet and returns total length value as well as actual length bytes. We do not support any length byte count of more than 2 i.e. total length value must not be more than 16-bit.

 Preconditions

None

 Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>Pointer to memory where actual length is stored.</td>
</tr>
</tbody>
</table>

 Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lengthBytes</td>
<td>Total length bytes are 0x80 itself plus tempData.</td>
</tr>
</tbody>
</table>

 Remarks

None.
SNMP_OPAQUE Macro

C

```
#define SNMP_OPAQUE (0x44)
```

Description

This is macro SNMP_OPAQUE.
SNMP_STATUS Union

C

typedef union {
  struct {
    unsigned int bIsFileOpen : 1;
  } Flags;
  BYTE Val;
} SNMP_STATUS;

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int bIsFileOpen : 1;</td>
<td>MIB file access int flag</td>
</tr>
<tr>
<td>BYTE Val;</td>
<td>MIB file access byte flag</td>
</tr>
</tbody>
</table>

Stack API > SNMP > Internal Members > SNMP_STATUS Union
SNMP_TIME_TICKS Macro

C

#define SNMP_TIME_TICKS (0x43)

Description

This is macro SNMP_TIME_TICKS.
SNMP_V1 Macro

```
#define SNMP_V1 (0)
```

Stack API > SNMP > Internal Members > SNMP_V1 Macro

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
SNMP_V2C Macro

C
#define SNMP_V2C (1)

Description

This is macro SNMP_V2C.

Stack API > SNMP > Internal Members > SNMP_V2C Macro
SNMPAgentSocket Variable

```c
UDP_SOCKET SNMPAgentSocket = INVALID_UDP_SOCKET;
```

Description

Snmp udp socket

[Stack API > SNMP > Internal Members > SNMPAgentSocket Variable](#)
SNMPNotifyInfo Variable

```c
SNMP_NOTIFY_INFO SNMPNotifyInfo;
```

Description

notify info for trap
snmpReqVarErrStatus Variable

```c
reqVarErrStatus snmpReqVarErrStatus;
```

**Description**

vars from req list processing err status

Stack API > SNMP > Internal Members > snmpReqVarErrStatus Variable
SNMPRxOffset Variable

C

WORD SNMPRxOffset = 0;

Description

Snmp udp buffer rx offset

Stack API > SNMP > Internal Members > SNMPRxOffset Variable
SNMPStatus Variable

```c
SNMP_STATUS SNMPStatus;
```

**Description**

MIB file access status

[Stack API > SNMP > Internal Members > SNMPStatus Variable](#)
SNMPTxOffset Variable

```c
WORD SNMPTxOffset;
```

Description

`Snmp udp buffer tx offset`

Stack API > SNMP > Internal Members > SNMPTxOffset Variable
#define STRUCTURE (0x30u)
TRAP Macro

```
#define TRAP (0xa4)
```

Description

This is macro TRAP.
trapInfo Variable

```c
TRAP_INFO trapInfo = { TRAP_TABLE_SIZE };
```

Description

Initialize trap table with no entries.
GetNextLeaf Function

```c
BOOL GetNextLeaf(
    OID_INFO* rec
);
```

**Description**

This routine searches for the next leaf node from the current node. The input to this function is the node from where next leaf node is to be located. The next leaf node will be a sibling else distant sibling or leaf node of next branch, if any present. The input parameter var pointer will be updated with the newly found leaf node OID info.

**Preconditions**

(ProcessGetBulkVar() else ProcessGetNextVar() is called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec</td>
<td>Pointer to SNMP MIB variable object information</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If next leaf node is found.</td>
</tr>
<tr>
<td>FALSE</td>
<td>There is no next leaf node.</td>
</tr>
</tbody>
</table>

**Remarks**

None.
GetOIDStringByAddr Function

```c
BOOL GetOIDStringByAddr(
    OID_INFO* rec,
    BYTE* oidString,
    BYTE* len
);
```

Description

This routine is called when a OID string is required to be searched from MPFS using node address.

Preconditions

None.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec</td>
<td>Pointer to SNMP MIB variable object information</td>
</tr>
<tr>
<td>oidString</td>
<td>Pointer to store the string of OID searched</td>
</tr>
<tr>
<td>len</td>
<td>Oid length</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If oid string is found.</td>
</tr>
<tr>
<td>FLASE</td>
<td>Otherwise.</td>
</tr>
</tbody>
</table>

Remarks

None.
GetOIDStringByID Function

C

```c
BOOL GetOIDStringByID(
    SNMP_ID id,
    OID_INFO* info,
    BYTE* oidString,
    BYTE* len
);
```

Description

This routine is called when a OID string is required to be searched from MPFS using agent id. The string is saved with agent. TRAP pdu is send with this OID corresponding to the SNMP_ID used by the agent application to send the pdu.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>System ID to use identify this agent.</td>
</tr>
<tr>
<td>info</td>
<td>Pointer to SNMP MIB variable object information</td>
</tr>
<tr>
<td>oidString</td>
<td>Pointer to store the string of OID searched</td>
</tr>
<tr>
<td>len</td>
<td>Oid length</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If oid string is found for the variable id in MPFS.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Otherwise.</td>
</tr>
</tbody>
</table>

Remarks
This function is used only when TRAP is enabled.
IsValidCommunity Function

```c
static BOOL IsValidCommunity(
    char* community,
    BYTE* len
);
```

Description

This routine populates and validates the community datatype, community name and length from the received snmp request pdu. Community name is used for accessing public and private members of the mib.

Preconditions

ProcessHeader() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>community</td>
<td>Pointer to memory where community string will be stored.</td>
</tr>
<tr>
<td>len</td>
<td>Pointer to memory where community length gets stored.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If valid community received.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If community is not valid.</td>
</tr>
</tbody>
</table>

Remarks

None.
IsValidInt Function

C

BOOL IsValidInt(
    DWORD* val
);

Description

This routine populates and validates the received variable for the data type as "ASN_INT" and the data length for max 4 bytes.

Preconditions

ProcessHeader() or ProcessGetSetHeader() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Pointer to memory where int var value will be stored.</td>
</tr>
<tr>
<td>ReturnValue</td>
<td>TRUE - If valid integer type and value is received.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Other than integer data type and value received.</td>
</tr>
</tbody>
</table>

Remarks

None.
IsValidOID Function

C

```
static BOOL IsValidOID(
    BYTE* oid,
    BYTE* len
);
```

Description

In this routine, OID data type "ASN_OID" is verified in the received pdu. If the data type is matched, then only var bind is processed. OID length and OID is populated. The max OID length can be 15.

Preconditions

ProcessVariabels() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oid</td>
<td>Pointer to memory to store the received OID string</td>
</tr>
<tr>
<td>len</td>
<td>Pointer to memory to store OID length</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If value type is ASN_OID and oid length not more than 15.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Otherwise.</td>
</tr>
</tbody>
</table>

Remarks

None.
IsValidPDU Function

C

```c
static BOOL IsValidPDU(SNMP_ACTION* pdu);
```

Description

This routine populates and verifies for the received snmp request pdu type.

Preconditions

ProcessHeader() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Pointer to memory where received snmp request type is stored.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If this snmp request can be processed by the agent.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If the request can not be processed.</td>
</tr>
</tbody>
</table>

Remarks

None.
IsValidStructure Function

C

BYTE IsValidStructure(
    WORD* dataLen
);

Description

This routine is used to verify whether the received varbind is of type STRUCTURE and to find out the variable binding structure length.

Preconditions

ProcessHeader() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datalen</td>
<td>Pointer to memory to store OID structure length.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>headrbytes</td>
<td>Variable binding length.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If variable data structure is not type STRUCTURE.</td>
</tr>
</tbody>
</table>

Remarks

None.
OIDLookup Function

C

BYTE OIDLookup(  
    PDU_INFO* pduDbPtr,  
    BYTE* oid,  
    BYTE oidLen,  
    OID_INFO* rec
);

Description

The MIB database is stored with the agent in binary mib format. This is the binary mib format: , , ...>]}, ChildNode variable bind name is a dotted string of oid. Every oid is a node in the MIB tree and have varied information. This routine on reception of the snmp request, will search for every oid in the var name. This routine will return information whether the requested var name is part of the MIB tree data structre of this agent or not.

Preconditions

Valid snmp request with valid OID format is received.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pduDbPtr</td>
<td>Pointer to received snmp pdu elements information</td>
</tr>
<tr>
<td>oid</td>
<td>Pointer to the string of OID to be searched</td>
</tr>
<tr>
<td>oidLen</td>
<td>Oid length</td>
</tr>
<tr>
<td>rec</td>
<td>Pointer to SNMP MIB variable object information</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If the complete OID string is found in the mib</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>FALSE</td>
<td>If complete OID do not match. Also different errors returned are SNMP_END_OF_MIB_VIEW SNMP_NO_SUCH_NAME SNMP_NO_SUCH_OBJ SNMP_NO_SUCH_INSTANCE</td>
</tr>
</tbody>
</table>

Remarks

This routine works for the MPFS2 mib storage format. It uses the MPFS2 APIs to read, search and collect information from the mib database.

Stack API > SNMP > Internal Members > OIDLookup Function
ProcessGetSetHeader Function

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.

Preconditions

ProcessHeader() is called and returns pdu type and do not returns SNMP_ACTION_UNKNOWN

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pduDbPtr</td>
<td>Pointer to received pdu information database.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If the received request header is validated and passed.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If rxed request header is not valid.</td>
</tr>
</tbody>
</table>

Remarks

The request pdu will be processed only if this routine returns TRUE
ProcessHeader Function

```c
static SNMP_ACTION ProcessHeader(
    PDU_INFO* pduDbPtr,
    char* community,
    BYTE* len
);
```

**Description**

Collects PDU_INFO (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.

**Preconditions**

UDPIsGetReady(SNMPAgentSocket) is called in SNMPTask(), it check if there is any packet on SNMP Agent socket, should return TRUE.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pduDbPtr</td>
<td>Pointer to received pdu information database</td>
</tr>
<tr>
<td>community</td>
<td>Pointer to var storing, community string in rxed pdu</td>
</tr>
<tr>
<td>len</td>
<td>Pointer to var storing, community string length rxed in pdu</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP_ACTION</td>
<td>Snmp request pdu type.</td>
</tr>
</tbody>
</table>
Remarks

The received pdu will be processed only if this routine returns the pdu type else the pdu is discarded as not Snmp pdu.
ProcessSetVar Function

C

BYTE ProcessSetVar(
    PDU_INFO* pduDbPtr,
    OID_INFO* rec,
    SNMP_ERR_STATUS* errorStatus
);

Description

This routine processes the received snmp set request pdu for the variable binding in the request and also creates the response pdu.

Preconditions

ProcessVariables() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pduDbPtr</td>
<td>Received pdu information database pointer</td>
</tr>
<tr>
<td>rec</td>
<td>Pointer to SNMP MIB variable object information</td>
</tr>
<tr>
<td>errorStatus</td>
<td>Pointer to update error status info</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copiedBytes</td>
<td>Number of bytes copied by this routine to the snmp pdu tx buffer.</td>
</tr>
</tbody>
</table>

Remarks

None.
ProcessVariables Function

```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.

Preconditions

The received udp packet is varified as SNMP request. ProcessHeader() and ProcessGetSetHeader() returns but FALSE.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pduDbPtr</td>
<td>Pointer to received pdu information database</td>
</tr>
<tr>
<td>community</td>
<td>Pointer to var, storing community string in rxed pdu</td>
</tr>
<tr>
<td>len</td>
<td>Pointer to var, storing community string length rxed in pdu</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If the snmp request processing is successful.</td>
</tr>
<tr>
<td></td>
<td>If the processing failed else the processing is not</td>
</tr>
</tbody>
</table>
FALSE completed.

Remarks

None

Stack API > SNMP > Internal Members > ProcessVariables Function

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
ReadMIBRecord Function

```c
static void ReadMIBRecord(
    DWORD h,
    OID_INFO* rec
);
```

**Description**

This routine is called when a OID string is required to be searched from MPFS using node address.

**Preconditions**

GetOIDStringByID() or GetOIDStringByAddr() is called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>Node address whose oid is to be read.</td>
</tr>
<tr>
<td>rec</td>
<td>Pointer to store SNMP MIB variable object information</td>
</tr>
</tbody>
</table>

**Returns**

None.

**Remarks**

None.

Stack API > SNMP > Internal Members > ReadMIBRecord Function
**SNMPCheckIfPvtMibObjRequested Function**

```c
static BOOL SNMPCheckIfPvtMibObjRequested(
    BYTE* OIDValuePtr
);
```

**Description**

This routine is used to find whether requested object belongs to the private object group of the mib of agent. If yes, then that mib object can be accessed only with private community (supported in SNMPv2c).

**Preconditions**

*ProcessVariables()* is called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OIDValuePtr</td>
<td>Pointer to memory stored with received OID.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If the requested object is of private branch of the mib.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If the requested object is publically accessible.</td>
</tr>
</tbody>
</table>

**Remarks**

None.
SNMP Stack Members

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.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMPInit</td>
<td>Initialize SNMP module internals.</td>
</tr>
<tr>
<td>SNMPTask</td>
<td>Polls for every snmp pdu received.</td>
</tr>
</tbody>
</table>

### Module

SNMP

Stack API > SNMP > Stack Members
SNMPInit Function

C
void SNMPInit();

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.

Preconditions

At least one UDP socket must be available. UDPInit() is already called.

Returns

None

Remarks

This function is called only once during lifetime of the application. One UDP socket will be used.
SNMPTask Function

C

BOOL SNMPTask();

Description

Handle incoming SNMP requests as well as any outgoing SNMP responses and timeout conditions.

Preconditions

SNMPInit() is already called.

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If SNMP module has finished with a state</td>
</tr>
<tr>
<td>FALSE</td>
<td>If a state has not been finished.</td>
</tr>
</tbody>
</table>

Remarks

None
## Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IsSNMPv3ValidStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>_Snmpv3IsValidAuthStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>_Snmpv3IsValidInt</td>
<td>Verifies variable datatype as INT and retrieves its value.</td>
</tr>
<tr>
<td>FindOIDsFromSnmpV3Request</td>
<td>Finds number of varbinds in the varbind list received in a SNMPv3 pdu.</td>
</tr>
<tr>
<td>getSnmpV2GenTrapOid</td>
<td>Resolves generic trap code to generic trap OID.</td>
</tr>
<tr>
<td>IsSnmpV3ASNNull</td>
<td>Verifies the value type as ASN_NULL.</td>
</tr>
<tr>
<td>IsSnmpv3ValidOID</td>
<td>Populates OID type, length and oid string from the received pdu.</td>
</tr>
<tr>
<td>IsSNMPv3ValidStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>ProcessGetBulkVar</td>
<td>This routine process the SNMPv2c Get Bulk Request.</td>
</tr>
<tr>
<td>ProcessGetNextVar</td>
<td>Retrieves next node from the MIB database.</td>
</tr>
<tr>
<td>ProcessGetVar</td>
<td>Processes snmp Get request pdu.</td>
</tr>
<tr>
<td>ProcessSnmpv3MsgData</td>
<td>This routine processes the snmpv3 request and parallely creates the response pdu.</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SNMPGetExactIndex</td>
<td>To search for exact index node in case of a Sequence variable.</td>
</tr>
<tr>
<td>SNMPGetTrapTime</td>
<td>Returns trap resolve get time.</td>
</tr>
<tr>
<td>SNMPIdRecrdValidation</td>
<td>Used to Restrict the access dynamic and non dynamic OID string for A particular SNMP Version.</td>
</tr>
<tr>
<td>SNMPIsValidSetLen</td>
<td>Validates the set variable data length to data type.</td>
</tr>
<tr>
<td>Snmpv3AESDecryptRxedScopedPdu</td>
<td>Incoming SNMPv3 scoped PDU decryption using AES decryption protocol.</td>
</tr>
<tr>
<td>Snmpv3AESEncryptResponseScopedPdu</td>
<td>outGoing SNMPv3 scoped PDU Encryption using AES encryption protocol.</td>
</tr>
<tr>
<td>Snmpv3AuthenticateRxedPduForDataIntegrity</td>
<td>Authenticate an incoming SNMPV3 USM PDU using MD5 or SHA</td>
</tr>
<tr>
<td>Snmpv3AuthenticateTxPduForDataIntegrity</td>
<td>Authenticate to an outgoing SNMPV3 USM PDU using MD5 or SHA</td>
</tr>
<tr>
<td>Snmpv3AuthKeyZeroing2HmacBufLen64</td>
<td>Pad zero to the authentication key localized buffer.</td>
</tr>
<tr>
<td>Snmpv3BufferPut</td>
<td>Copies BYTE data to dynamically allocated memory buffer.</td>
</tr>
<tr>
<td>Snmpv3CmprTrapSecNameAndSecLvlWithUSMDb</td>
<td>Routine to find the index of the user name in the user data base table.</td>
</tr>
<tr>
<td>Snmpv3ComputeHMACIpadOpadForAuthLoclzedKey</td>
<td>Compute HMAC inner and outer pad for authorization localized key.</td>
</tr>
<tr>
<td>Snmpv3ComputeHmacMD5Digest</td>
<td>Compute HMAC - MD5</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Snmpv3ComputeHmacShaDigest</td>
<td>Compute HMAC - SHA authentication code</td>
</tr>
<tr>
<td>Snmpv3ComputeMd5HmacCode</td>
<td>Compute HMAC - MD5 authentication code</td>
</tr>
<tr>
<td>Snmpv3ComputeShaHmacCode</td>
<td>Compute HMAC - SHA authentication code</td>
</tr>
<tr>
<td>Snmpv3FormulateEngineID</td>
<td>Formulates the snmpEngineID for the SNMPV3 engine.</td>
</tr>
<tr>
<td>Snmpv3FreeDynAllocMem</td>
<td>Allocated dynamic memory freeing is done by this routine.</td>
</tr>
<tr>
<td>Snmpv3GetAuthEngineTime</td>
<td>Updates the snmp engine time variable 'snmpEngineTime' for the SNMPV3 engine.</td>
</tr>
<tr>
<td>Snmpv3GetBufferData</td>
<td>Reads BYTE data from dynamically allocated memory buffer.</td>
</tr>
<tr>
<td>Snmpv3GetSecurityLevel</td>
<td>Get Security level from authentication and Privacy type.</td>
</tr>
<tr>
<td>Snmpv3GetTrapSecurityLevel</td>
<td>Routine to find the report, auth and privacy flags settings in the TRAP.</td>
</tr>
<tr>
<td>Snmpv3Init</td>
<td>SNMPv3 initialization.</td>
</tr>
<tr>
<td>Snmpv3InitializeUserDatabase</td>
<td>Initialize default SNMPv3 global user database.</td>
</tr>
<tr>
<td>Snmpv3IsValidAuthStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>Snmpv3IsValidInt</td>
<td>Verifies variable datatype as INT and retrieves its value.</td>
</tr>
<tr>
<td></td>
<td>This routine collects or</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>snmpv3MsgProcessingModelProcessPDU</td>
<td>populates the message processing model information from the received SNMPv3 request PDU or to the response PDU respectively.</td>
</tr>
<tr>
<td>snmpv3Notify</td>
<td>Creates and Sends SNMPv3 TRAP pdu.</td>
</tr>
<tr>
<td>snmpv3Pswd2LocalizedAuthKeyMD5Hashing</td>
<td>Convert MD5 Auth password to the localized Key using SNMPEngineID.</td>
</tr>
<tr>
<td>snmpv3Pswd2LocalizedAuthKeySHAHashing</td>
<td>Convert SHA Auth password to the localized Key using SNMPEngineID.</td>
</tr>
<tr>
<td>snmpv3ScopedPduProcessing</td>
<td>This routine collects the scoped pdu header information from the received SNMPv3 request PDU or populates to the response PDU respectively.</td>
</tr>
<tr>
<td>snmpv3SetErrorStatus</td>
<td>Set snmpv3 error status in the response pdu.</td>
</tr>
<tr>
<td>snmpv3TrapScopedPdu</td>
<td>TRAP PDU scoped pdu header construction.</td>
</tr>
<tr>
<td>snmpv3UserSecurityModelProcessPDU</td>
<td>This routine collects or populates the security model parameters information from the received SNMPv3 request PDU or to the response PDU respectively.</td>
</tr>
<tr>
<td>snmpv3UsmAesEncryptDecryptInitVector</td>
<td>AES Encryption and decryption init vector (RFC 3826)</td>
</tr>
<tr>
<td>snmpv3UsmOutMsgAuthenticationParam</td>
<td>Both MD5 and SHA1 is used for the outgoing message authentication.</td>
</tr>
<tr>
<td>Module</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>SNMP</strong></td>
<td>Convert Auth and Priv password to the localized Key using SNMPEngineID.</td>
</tr>
<tr>
<td><strong>Snmpv3ValidateEngineId</strong></td>
<td>Validate engine ID.</td>
</tr>
<tr>
<td><strong>Snmpv3ValidateSecNameAndSecLvl</strong></td>
<td>Validate security name with Security level.</td>
</tr>
<tr>
<td><strong>Snmpv3ValidateSecName</strong></td>
<td>Validate SNMPV3 user name or security name.</td>
</tr>
</tbody>
</table>
BOOL _IsSNMPv3ValidStructure(
    UINT8* wholeMsgPtr,
    WORD* pos,
    WORD* dataLen
);

Description

This routine populates and validates the received variable for the data type as "STRUCTURE" and the data length for max 4 bytes.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wholeMsgPtr</td>
<td>Pointer to memory where int var value is be stored.</td>
</tr>
<tr>
<td>pos</td>
<td>position in the memory buffer where data type to be verified is stored</td>
</tr>
<tr>
<td>val</td>
<td>Pointer to memory where int var value will be stored.</td>
</tr>
<tr>
<td>ReturnValues</td>
<td>TRUE - If valid integer type and value is received.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Other than integer data type and value received.</td>
</tr>
</tbody>
</table>

Remarks

None.
__Snmpv3IsValidAuthStructure Function

C

BYTE __Snmpv3IsValidAuthStructure(
    WORD* dataLen
);

Description

This routine is used to verify whether the received varbind is of type STRUCTURE and to find out the variable binding structure length. This routine only refers to the incoming snmpv3 request dynamically allocated memory buffer 'gSnmpV3InPduWholeMsgBuf'.

Preconditions

ProcessHeader() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataLen</td>
<td>Pointer to memory to store OID structure length.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>headrbytes</td>
<td>Variable binding length.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If variable data structure is not type STRUCTURE.</td>
</tr>
</tbody>
</table>

Remarks

None.
_Snmpv3IsValidInt Function

C

```c
BOOL _Snmpv3IsValidInt(
    UINT8 * wholeMsgPtr,
    WORD* pos,
    DWORD* val
);
```

Description

This routine populates and validates the received variable for the data type as "ASN_INT" and the data length for max 4 bytes.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wholeMsgPtr</td>
<td>Pointer to memory where int var value is be stored.</td>
</tr>
<tr>
<td>pos</td>
<td>position in the memory buffer where data type to be verified is stored</td>
</tr>
<tr>
<td>val</td>
<td>Pointer to memory where int var value will be stored.</td>
</tr>
<tr>
<td>ReturnValues</td>
<td>TRUE - If valid integer type and value is received.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Other than integer data type and value received</td>
</tr>
</tbody>
</table>

Remarks

None.
FindOIDsFromSnmpV3Request Function

C

static BYTE FindOIDsFromSnmpV3Request(
    WORD pdulen
);

Description

This routine is used to find the number of OIDs requested in the received snmp pdu.

Preconditions

ProcessVariables() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdulen</td>
<td>Length of snmp pdu request received.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varCount</td>
<td>Number of OIDs found in a pdu request.</td>
</tr>
</tbody>
</table>

Remarks

None.

Stack API > SNMP > Functions > FindOIDsFromSnmpV3Request Function
getSnmpV2GenTrapOid Function

```
C
BYTE * getSnmpV2GenTrapOid(
    BYTE generic_trap_code,
    BYTE * len
);
```

**Description**

This function resolves given generic trap code to generic trap OID.

**Preconditions**

`SNMPNotifyPrepare()` is already called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>generic_trap_code</code></td>
<td><code>GENERIC_TRAP_NOTIFICATION_TYPE</code></td>
</tr>
<tr>
<td><code>len</code></td>
<td>generic trap OID length</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BYTE *</code></td>
<td>TRAP OID</td>
</tr>
</tbody>
</table>

**Remarks**

This would fail if `generic_trap_code` is not coming under `GENERIC_TRAP_NOTIFICATION_TYPE`
IsSnmpV3ASNNull Function

C

static BOOL IsSnmpV3ASNNull();

Description

For Get, Get_Next, Get_Bulk snmp request, the var bind the value data type should be ASN_NULL and value field must be NULL and . This routine verifies the data type and value fields in the received requests. The SET request, the value data type can not be ASN_NULL, otherwise the snmp request is not processed. This routine only refers to the incoming snmpv3 request dynamically allocated memory buffer 'gSNMPv3ScopedPduRequestBuf'.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If value type is ASN_NULL and value is NULL.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If data type and value is other than ASN_NULL and NULL resp.</td>
</tr>
</tbody>
</table>

Remarks

None.

Stack API > SNMP > Functions > IsSnmpV3ASNNull Function
IsSnmpv3ValidOID Function

```c
static BOOL IsSnmpv3ValidOID(
    BYTE* oid,
    BYTE* len
);
```

**Description**

In this routine, OID data type "ASN_OID" is verified in the received pdu. If the data type is matched, then only var bind is processed. OID length and OID is populated. The max OID length can be 15.

**Preconditions**

ProcessVariabels() is called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oid</td>
<td>Pointer to memory to store the received OID string</td>
</tr>
<tr>
<td>len</td>
<td>Pointer to memory to store OID length</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If value type is ASN_OID and oid length not more than 15.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Otherwise.</td>
</tr>
</tbody>
</table>

**Remarks**

None.
IsSNMPv3ValidStructure Function

```c
BOOL IsSNMPv3ValidStructure(
   WORD* dataLen
);
```

Description

This routine is used to verify whether the received varbind is of type `STRUCTURE` and to find out the variable binding structure length. This routine only refers to the incoming snmpv3 request dynamically allocated memory buffer `gSNMPv3ScopedPduRequestBuf`.

Preconditions

`ProcessHeader()` is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datalen</td>
<td>Pointer to memory to store OID structure length.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If valid Structure data type and value is received.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If variable data structure is not type <code>STRUCTURE</code>.</td>
</tr>
</tbody>
</table>

Remarks

None.
ProcessGetBulkVar Function

```
C

BYTE ProcessGetBulkVar(
    OID_INFO* rec,
    BYTE* oidValuePtr,
    BYTE* oidLenPtr,
    BYTE* successor,
    PDU_INFO* pduDbPtr
);
```

### Description

`ProcessVariables()` processes the received snmp request pdu for each of the variable binding in the variable binding list to produce a response pdu. Depending on the number of the Max_repetitions for every variable in the list for which Bulk information is expected, `ProcessGetBulkVar()` is executed. It searches for the next lexicographically ordered successor for of the OID received in the request. For each of the iterations upto max-repetitions, the next leaf node is searched in the MIB to that of the leaf node found in the last iteration, for the corresponding variable binding.

### Preconditions

`ProcessVariables()` is called.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec</td>
<td>Pointer to SNMP MIB variable object information OID</td>
</tr>
<tr>
<td>oidValuePtr</td>
<td>Pointer to new node OID found in MIB</td>
</tr>
<tr>
<td>oidLenPtr</td>
<td>Oid length</td>
</tr>
<tr>
<td>successor</td>
<td>'I'th lexicographic successor to be found value</td>
</tr>
</tbody>
</table>
## Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>If no lexicographic successor found</td>
</tr>
<tr>
<td>temp.v[0]</td>
<td>Total number of bytes copied to response packet</td>
</tr>
</tbody>
</table>

## Remarks

None.

Stack API > SNMP > Functions > ProcessGetBulkVar Function

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
ProcessGetNextVar Function

```
BYTE ProcessGetNextVar(
    OID_INFO* rec,
    PDU_INFO* pduDbPtr
);
```

**Description**

This routine reads into the MIB stored with the agent in MPFS2 format. It will search for the first lexicographic successor of the variable binding's name in the incoming GetNextRequest-PDU. If found, the corresponding variable binding's name and value fields in the Response pdu are set to the name and value of the located variable. If the lexicographic successor is not found, the value filed is set to "endofMibView" and name field is retained as in request.

**Preconditions**

`ProcessVariables` is called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec</td>
<td>Pointer to SNMP MIB object information for which next node to be found</td>
</tr>
<tr>
<td>pduDbPtr</td>
<td>Pointer to received pdu information database</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Total number of bytes copied to response packet if sucessful.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If End of MIB is reached or processing is failure.</td>
</tr>
</tbody>
</table>
Remarks

None.

Stack API > SNMP > Functions > ProcessGetNextVar Function
ProcessGetVar Function

C

BYTE ProcessGetVar(
  OID_INFO* rec,
  BOOL bAsOID,
  PDU_INFO* pduDbPtr
);

Description

This routine processes the received snmp Get request pdu for the variable binding in the request and also creates the response pdu.

Preconditions

ProcessVariables() is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec</td>
<td>Pointer to SNMP MIB variable object information</td>
</tr>
<tr>
<td>bAsOID</td>
<td>Oid flag.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varLen</td>
<td>Number of bytes put in response tx pdu</td>
</tr>
<tr>
<td>FALSE</td>
<td>If any of the elements of the request pdu validation fails.</td>
</tr>
</tbody>
</table>

Remarks

None.

Stack API > SNMP > Functions > ProcessGetVar Function
ProcessSnmpv3MsgData Function

```c
BOOL ProcessSnmpv3MsgData(
PDU_INFO* pduDbPtr
);
```

### Description

Once the received pdu is validated as Snmpv3 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.

### Preconditions

The received udp packet is verified as valid SNMPv3 request.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pduDbPtr</td>
<td>Pointer to received pdu information database</td>
</tr>
</tbody>
</table>

### Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If the snmp request processing is successful.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If the processing failed else the processing is not completed.</td>
</tr>
</tbody>
</table>

### Remarks

None
SNMPGetExactIndex Function

C

```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.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>var</code></td>
<td>Variable id as per mib.h (input)</td>
</tr>
<tr>
<td><code>index</code></td>
<td>Index of variable (input)</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If the exact index value exists for given variable at given index.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Otherwise.</td>
</tr>
</tbody>
</table>

Remarks
Only sequence index needs to be handled in this function.
SNMPGetTrapTime Function

C

DWORD SNMPGetTrapTime();

Preconditions

SNMPNotifyPrepare() is already called.

Remarks

none
SNMPIdRecrdValidation Function

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.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>Variable rec whose record id need to be validated</td>
</tr>
<tr>
<td>oidValuePtr</td>
<td>OID Value</td>
</tr>
<tr>
<td>oidLen</td>
<td>oidValuePtr length</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>If a Var ID exists .</td>
</tr>
<tr>
<td>FALSE</td>
<td>Otherwise.</td>
</tr>
</tbody>
</table>
Remarks

None.

Stack API > SNMP > Functions > SNMPIdRecrdValidation Function
SNMPIsValidSetLen Function

```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.

Preconditions

_ProcessSetVar()_ is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>Variable id whose value is to be set</td>
</tr>
<tr>
<td>len</td>
<td>Length value that is to be validated.</td>
</tr>
<tr>
<td>index</td>
<td>instance of a OID</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>if given var can be set to given len</td>
</tr>
<tr>
<td>FALSE</td>
<td>if otherwise.</td>
</tr>
</tbody>
</table>

Remarks
This function will be called for only dynamic variables that are defined as ASCII_STRING and OCTET_STRING (i.e. data length greater than 4 bytes)
Snmpv3AESDecryptRxedScopedPdu Function

C

BYTE Snmpv3AESDecryptRxedScopedPdu();

Description

This routine decrypts SNMPV3 incoming PDU using AES protocol, but before this encrypted data length is verified. If the length of the encrypted OCTECT-STRING is not multiple of 8, then decryption will be halted. RFC - 3414. ( section 8)

Preconditions

SNMPv3Init() and ProcessVariabels() are called.

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMPV3_MSG_PRIV_FAIL</td>
<td>Failure</td>
</tr>
<tr>
<td>SNMPV3_MSG_PRIV_PASS</td>
<td>Success</td>
</tr>
</tbody>
</table>

Remarks

None
# Snmpv3AESEncryptResponseScopedPdu Function

```c
BYTE Snmpv3AESEncryptResponseScopedPdu(
    SNMPV3_RESPONSE_WHOLEMSG* plain_text
);
```

## Description

This routine encrypts SNMPV3 outgoing PDU using AES protocol to maintain the data confidentiality. The data is encrypted in Cipher Block Chaining mode. The length of the encrypted data should be multiple of 8 and it is not then then data is padded in the end if necessary. RFC - 3414. ( section 8)

## Preconditions

SNMPv3Init() and ProcessVariabels() are called.

## Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain_text</td>
<td>whole PDU message</td>
</tr>
</tbody>
</table>

## Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMPV3_MSG_PRIV_FAIL</td>
<td>Failure</td>
</tr>
<tr>
<td>SNMPV3_MSG_PRIV_PASS</td>
<td>Success</td>
</tr>
</tbody>
</table>

## Remarks

None
Snmpv3AuthenticateRxedPduForDataIntegrity

Function

C

```
BYTE Snmpv3AuthenticateRxedPduForDataIntegrity(
    SNMPV3_REQUEST_WHOLEMSG* rxDataPtr
);
```

Description

This routine authenticates SNMPV3 incoming report PDU message and also for different type of GET requests with both MD5 and SHA protocol. If the received PDU username is similar to "initial", then there should be report PDU. RFC - 3414.

Preconditions

SNMPv3Init() and ProcessVariabels() are called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rxDataPtr</td>
<td>incoming PDU</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMPV3_MSG_AUTH_PASS</td>
<td>Authentication success</td>
</tr>
<tr>
<td>SNMPV3_MSG_AUTH_FAIL</td>
<td>Authentication failure</td>
</tr>
</tbody>
</table>

Remarks

None
Snmpv3AuthenticateRxedPduForDataIntegrity Function
Snmpv3AuthenticateTxPduForDataIntegrity Function

C

BYTE Snmpv3AuthenticateTxPduForDataIntegrity(
   SNMPV3_RESPONSE_WHOLEMSG* txDataPtr
);

Description

This routine authenticates SNMPV3 outgoing report PDU message and also for GET Response PDU for whole message. RFC - 3414.

Preconditions

SNMPv3Init() and ProcessVariables() are called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>txDataPtr</td>
<td>outgoing PDU</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMPV3_MSG_AUTH_PASS</td>
<td>Authentication success</td>
</tr>
<tr>
<td>SNMPV3_MSG_AUTH_FAIL</td>
<td>Authentication failure</td>
</tr>
</tbody>
</table>

Remarks

None
Snmpv3AuthKeyZeroing2HmacBufLen64 Function

C

```c
void Snmpv3AuthKeyZeroing2HmacBufLen64(
    UINT8* authKey,
    UINT8 authKeyLen,
    UINT8 hashType
);
```

Description

this routine will pad the (64-authKeyLen) number of zeros to the end of auth key localized buffer.

Preconditions

SNMPv3Init() and ProcessVariables() are called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authkey</td>
<td>authentication key buffer</td>
</tr>
<tr>
<td>authKeylen</td>
<td>authentication key length</td>
</tr>
<tr>
<td>hashType</td>
<td>authentication type</td>
</tr>
</tbody>
</table>

Remarks

None
Snmpv3BufferPut Function

C

```c
BOOL Snmpv3BufferPut(
    BYTE val,
    SNMPV3MSGDATA * putbuf
);
```

Description

The SNMPv3 stack implementation uses dynamically allocated memory buffer for processing of request and response packets. This routine copies the BYTE data to the allocated buffer and updates the offset length counter.

Preconditions

The SNMPv3 stack has successfully allocated dynamic memory buffer from the Heap

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>BYTE value to be written to the buffer</td>
</tr>
<tr>
<td>putbuf</td>
<td>pointer to the dynamically allocated buffer to which the 'val' to be written</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>if successfully write to the buffer</td>
</tr>
<tr>
<td>FALSE</td>
<td>failure in writing to the buffer</td>
</tr>
</tbody>
</table>

Remarks
This routine is used by the SNMPv3 stack. If required to be used by the application code, valid pointers should be passed to this routine.
Snmpv3CmprTrapSecNameAndSecLvlWithUSMDb Function

C

BOOL Snmpv3CmprTrapSecNameAndSecLvlWithUSMDb(
    BYTE targetIndex,
    BYTE userTrapSecLen,
    UINT8 * userTrapSecurityName,
    STD_BASED_SNMPV3_SECURITY_LEVEL securityLevel
);

Description

There are two different data base tables defined with SNMPv3 stack, like 'snmpV3UserDataBase' and 'gSnmpv3TrapConfigData'. This routine is used to validate the trap user security level setting with SET request.

Preconditions

SET operation would be allowed if the USM security conditions and user security name in the request is matched to one of the user security name stored in the usm user database.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userTrapSecLen</td>
<td>user sec name length in the SET request</td>
</tr>
<tr>
<td>userTrapSecurityName</td>
<td>pointer to user sec name in the SET request</td>
</tr>
<tr>
<td>securityLevel</td>
<td>trap security level to be SET on the agent</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>if the trap target user sec level setting is successful</td>
</tr>
</tbody>
</table>
If the SET failed due to non matching of the security parameters

Remarks

None.
Snmpv3ComputeHMACIpadOpadForAuthLoclzedKey Function

```c
void Snmpv3ComputeHMACIpadOpadForAuthLoclzedKey(
    UINT8 userDBIndex
);
```

**Description**

This routine computes HMAC inner and outer pad strings for authorization localized key. RFC - 2104.

**Preconditions**

SNMPv3Init() and ProcessVariabels() are called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userDBIndex</td>
<td>password storage poniter</td>
</tr>
</tbody>
</table>

**Remarks**

None
Snmpv3ComputeHmacMD5Digest Function

C

```c
UINT8* Snmpv3ComputeHmacMD5Digest(
    UINT8 * inData,
    UINT32 dataLen,
    UINT8* userExtendedLclzdKeyIpad,
    UINT8* userExtendedLclzdKeyOpad
);
```

Description

This routine supports data origin authentication and data integrity MD5 authentication. Both iPAD and OPAD is used to calculate the authenticate digest string. RFC - 3414 ( section 6)

Preconditions

SNMPv3Init() and ProcessVariabels() are called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>digestptr</td>
<td>output string</td>
</tr>
<tr>
<td>indata</td>
<td>input data</td>
</tr>
<tr>
<td>dataLen</td>
<td>input data length</td>
</tr>
<tr>
<td>userExtendedLclzdKeyIpad</td>
<td>IPAD</td>
</tr>
<tr>
<td>userExtendedLclzdKeyOpad</td>
<td>OPAD</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8 *</td>
<td>HMAC MD5 digest string</td>
</tr>
</tbody>
</table>
Remarks

None

Stack API > SNMP > Functions > Snmpv3ComputeHmacMD5Digest Function
**Snmpv3ComputeHmacShaDigest Function**

```c
UINT8* Snmpv3ComputeHmacShaDigest(
    UINT8 * inData,
    UINT32 dataLen,
    UINT8* userExtendedLclzdKeyIpad,
    UINT8* userExtendedLclzdKeyOpad
);
```

**Description**

This routine supports data origin authentication and data integrity SHA authentication. Both iPAD and OPAD is used to calculate the authenticate digest string. RFC - 3414 (section 6)

**Preconditions**

SNMPv3Init() and ProcessVariabels() are called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>digestptr</td>
<td>output string</td>
</tr>
<tr>
<td>indata</td>
<td>input data</td>
</tr>
<tr>
<td>dataLen</td>
<td>input data length</td>
</tr>
<tr>
<td>userExtendedLclzdKeyIpad</td>
<td>IPAD</td>
</tr>
<tr>
<td>userExtendedLclzdKeyOpad</td>
<td>OPAD</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8 *</td>
<td>HMAC SHA digest string</td>
</tr>
</tbody>
</table>
Remarks

None

Stack API > SNMP > Functions > Snmpv3ComputeHmacShaDigest Function
Snmpv3ComputeMd5HmacCode Function

```c
void Snmpv3ComputeMd5HmacCode(
    UINT8  xx_bits,
    UINT8* digestptr,
    UINT8 * indata,
    UINT32 dataLen,
    UINT8* userExtendedLclzdKeyIpad,
    UINT8* userExtendedLclzdKeyOpad
);
```

Description

This routine supports data origin authentication and data integrity MD5 authentication. Both iPAD and OPAD is used to calculate the authenticate digest string. RFC - 3414 (section 6)

Preconditions

SNMPv3Init() and ProcessVariabels() are called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx_bits</td>
<td>96 bit</td>
</tr>
<tr>
<td>digestptr</td>
<td>output string</td>
</tr>
<tr>
<td>indata</td>
<td>input data</td>
</tr>
<tr>
<td>dataLen</td>
<td>input data length</td>
</tr>
<tr>
<td>userExtendedLclzdKeyIpad</td>
<td>IPAD</td>
</tr>
<tr>
<td>userExtendedLclzdKeyOpad</td>
<td>OPAD</td>
</tr>
</tbody>
</table>

Remarks
Stack API > SNMP > Functions > Snmpv3ComputeMd5HmacCode Function
Snmpv3ComputeShaHmacCode Function

C

```c
void Snmpv3ComputeShaHmacCode(
    UINT8 xx_bits,
    UINT8* digestptr,
    UINT8 * indata,
    UINT32 dataLen,
    UINT8* userExtendedLclzdKeyIpad,
    UINT8* userExtendedLclzdKeyOpad
);
```

Description

This routine supports data origin authentication and data integrity SHA authentication. Both IPAD and OPAD is used to calculate the authencate digest string. RFC - 3414 ( section 6)

Preconditions

SNMPv3Init() and ProcessVariabels() are called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx_bits</td>
<td>96 bit</td>
</tr>
<tr>
<td>digestptr</td>
<td>output string</td>
</tr>
<tr>
<td>indata</td>
<td>input data</td>
</tr>
<tr>
<td>dataLen</td>
<td>input data length</td>
</tr>
<tr>
<td>userExtendedLclzdKeyIpad</td>
<td>IPAD</td>
</tr>
<tr>
<td>userExtendedLclzdKeyOpad</td>
<td>OPAD</td>
</tr>
</tbody>
</table>

Remarks
**Snmpv3FormulateEngineID Function**

```c
void Snmpv3FormulateEngineID(
    UINT8 fifthOctectIdentifier
);
```

**Description**

Formulates the `snmpEngineID` depending on value of 'fifthOctectIdentifier'. as MAC_ADDR_ENGN_ID using the application MAC address. 'fifthOctectIdentifier' default set to MAC_ADDR_ENGN_ID as the following octets used for the `snmpEngineID` are of mac address.

User can set this octet of their choice to formulate new `snmpEngineID`.

```c
fifthOctectIdentifier=IPV4_ADDR_ENGN_ID;
```

If `fifthOctectIdentifier=ADMIN_ASSIGNED_TEXT;` or `fifthOctectIdentifier=ADMIN_ASSIGNED_OCTETS;` then the following octets should be provided by the administrator through some custom application interface mechanism. API parameter 'fifthOctectIdentifier' has to be updated in the interface API before passing through `Snmpv3FormulateEngineID()`.

**Preconditions**

`InitAppConfig();` is called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fifthOctectIdentifier</td>
<td>Value of the 5th octet in the <code>snmpEngineID</code> which indicates how the rest (6th and following octets) are formatted.</td>
</tr>
</tbody>
</table>

**Remarks**
Authentication and encryption keys are generated using corresponding passwords and \texttt{snmpEngineID}. If the \texttt{snmpEngineID} is newly configured, then the auth and privacy keys would also change. Hence while using this API to change the \texttt{snmpEngineID} dynamically, care should be taken to update the new localized keys at the agent as well as at the manager.
Snmpv3FreeDynAllocMem Function

C

```c
void Snmpv3FreeDynAllocMem();
```

Description

On the successful completion of the processing of the SNMPv3 request, or the failure in the processing due to improper PDU formats, the allocated dynamic memory is required to be freed. This routine calls the `free()`, to deallocate memory.

Preconditions

The dynamic memory buffer is allocated.

Remarks

The SNMPv3 stack does uses the dynamic memory extensively for different processing needs, hence incoming and outgoing pdu memory buffers are created. This routine checks for the memory is being allocated before it attempts for the deallocation.
Snmpv3GetAuthEngineTime Function

C

```c
void Snmpv3GetAuthEngineTime();
```

Description

'SnmpEngineTime' is used for Timeliness checking for Message level security. Snmp engine keep updating the 'snmpEngineTime' variable for checking the time window for the request and responses/inform etc. This routine also updates snmpEngineBoots in scenarios of internal timer reset or 'snmpEngineTime' cntr overflowed the (2^31 -1) value specified in RFC3411.

Preconditions

SNMPInt(); is called.

Remarks

This routine is called every time the rx/tx PDU processing is handled by the SNMP agent. Updates the 'snmpEngineTime' and requires fregunet access to internal timer registers.

Stack API > SNMP > Functions > Snmpv3GetAuthEngineTime Function
Snmpv3GetBufferData Function

```c
BYTE Snmpv3GetBufferData(
    SNMPV3MSGDATA getbuf,
    UINT16 pos
);
```

Description

The SNMPv3 stack implementation uses dynamically allocated memory buffer for processing of request and response packets. This routine reads the BYTE data from the allocated buffer at the positions (offset) provided.

Preconditions

The SNMPv3 stack has sucessfully allocated dynamic memory buffer from the Heap

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getbuf</td>
<td>Structure from where to read the data byte.</td>
</tr>
<tr>
<td>pos</td>
<td>position in the buffer from which the data to be read</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>1 byte value read</td>
</tr>
</tbody>
</table>

Remarks

The read position offset is required to be provided every time the routine is called. This API do not increment the buffer read offset
automatically, everytime it is called.
### Snmpv3GetSecurityLevel Function

#### Description

This routine uses authentication a dn privacy type to find out the exact security enum type.

#### Preconditions

SNMPv3Init() and ProcessVariables() are called.

#### Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_REPORT_PRIVACY_AND_AUTH_PROVIDED</td>
<td>Only Privacy and <a href="#">Authentication</a></td>
</tr>
<tr>
<td>NO_REPORT_NO_PRIVACY_BUT_AUTH_PROVIDED</td>
<td>Only Auth, no privacy and no report</td>
</tr>
<tr>
<td>NO_REPORT_NO_PRIVACY_NO_AUTH</td>
<td>No report, no Privacy and no <a href="#">Authentication</a></td>
</tr>
</tbody>
</table>

#### Remarks

None
Snmpv3GetTrapSecurityLevel Function

C

```
BYTE Snmpv3GetTrapSecurityLevel(
    STD_BASED_SNMPV3_SECURITY_LEVEL securityLevel
);
```

Description

This routine to find the report, auth and privacy flags setting for the trap to be generated. The message flags octet's least significant three bits: Reportable, PrivFlag, AuthFlag forms different security level combinations.

Preconditions

None

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_REPORT_NO_PRIVACY_NO_AUTH</td>
<td>No authentication, no encryption</td>
</tr>
<tr>
<td>NO_REPORT_NO_PRIVACY_BUT_AUTH_PROVIDED</td>
<td>authentication but no encryption</td>
</tr>
<tr>
<td>NO_REPORT_PRIVACY_AND_AUTH_PROVIDED</td>
<td>authentication and encryption</td>
</tr>
<tr>
<td>INVALID_MSG</td>
<td>if security level doesn't match any of the above</td>
</tr>
</tbody>
</table>

Remarks

None.

Stack API > SNMP > Functions > Snmpv3GetTrapSecurityLevel Function
Snmpv3Init Function

C

void Snmpv3Init();

Description

This routine will initialize SNMPv3 global data base, engine ID and USM

Preconditions

At least one UDP socket must be available. UDPInit() is already called.

Remarks

None
Snmpv3InitializeUserDataBase Function

```c
void Snmpv3InitializeUserDataBase();
```

**Description**

There are three default username, authentication, authenticaton password and Privacy name and privacy password initialized with SNMPv3 global database.

**Preconditions**

SNMPv3Init() is called.

**Remarks**

None
Snmpv3IsValidAuthStructure Function

C

```c
BYTE Snmpv3IsValidAuthStructure(
    WORD* dataLen
);
```

Description

This routine is used to verify whether the received varbind is of type `STRUCTURE` and to find out the variable binding structure length.

Preconditions

`ProcessHeader()` is called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataLen</td>
<td>Pointer to memory to store OID structure length.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>headrbytes</td>
<td>Variable binding length.</td>
</tr>
<tr>
<td>FALSE</td>
<td>If variable data structure is not type <code>STRUCTURE</code>.</td>
</tr>
</tbody>
</table>

Remarks

None.
Snmpv3IsValidInt Function

C

```c
BOOL Snmpv3IsValidInt(
    DWORD* val
);
```

Description

This routine populates and validates the received variable for the data type as "ASN_INT" and the data length for max 4 bytes. This routine only refers to the incoming snmpv3 request dynamically allocated memory buffer 'gSNMPv3ScopedPduRequestBuf'.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Pointer to memory where int var value will be stored.</td>
</tr>
</tbody>
</table>

ReturnValues

| TRUE   | If valid integer type and value is received. |
| FALSE  | Other than integer data type and value received. |

Remarks

None.

Stack API > SNMP > Functions > Snmpv3IsValidInt Function
Snmpv3MsgProcessingModelProcessPDU

Function

```
C

SNMP_ERR_STATUS Snmpv3MsgProcessingModelProcessPDU(
    BYTE inOutPdu
);
```

Description

The received SNMPv3 PDU or the transmit PDU header has message processing data bytes information. This routine retrieves the message processing model information from the stored pdu or write the appropriate msg proc info to the response msg buffer.

Preconditions

Valid SNMPv3 request msg is received.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inOutPdu</td>
<td>indicates whether the incoming PDU is to be read for msg proc values to be retrieved or the response PDU is to be populated with these values</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP_NO_CREATION</td>
<td>Failure due to improper msg processing information format in the received PDU or failure in constructing the response PDU.</td>
</tr>
<tr>
<td>SNMP_NO_ERR</td>
<td>The message processing information retrieval or response PDU formation is successful</td>
</tr>
</tbody>
</table>
Remarks

The message processing model parameters like 'msgID', 'msgMaxSize', 'msgFlags' and 'msgSecurityModel' decides the SNMPv3 engine processing modalities regarding request or response PDU.
Snmpv3Notify Function

C

```c
BOOL Snmpv3Notify(
    SNMP_ID var,
    SNMP_VAL val,
    SNMP_INDEX index,
    UINT8 targetIndex);
```

Description

This function creates SNMPv3 trap PDU and sends it to previously specified remoteHost.

Preconditions

TRAP event is triggered.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>SNMP var ID that is to be used in notification</td>
</tr>
<tr>
<td>val</td>
<td>Value of var. Only value of BYTE, WORD or DWORD can be sent.</td>
</tr>
<tr>
<td>index</td>
<td>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. targetIndex -index of the <code>gSnmpv3TrapConfigData</code> table's security user name for which the TRAP PDU message header to constructed.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>if SNMP notification was successful sent. This does not guarantee that remoteHost received it.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Notification sent failed.</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>This would fail under following conditions</td>
<td>1) Given SNMP_BIB_FILE does not exist in MPFS 2) Given var does not exist. 3) Previously given agentID does not exist</td>
</tr>
<tr>
<td>4) Data type of given var is unknown</td>
<td>only possible if MPFS itself was corrupted. SNMPV3_MSG_MESSAGE - encryption of the trap msg failed</td>
</tr>
<tr>
<td>SNMPV3_MSG_AUTH_FAIL</td>
<td>HAMC of the trap msg failed</td>
</tr>
</tbody>
</table>

**Remarks**

None

Stack API > SNMP > Functions > Snmpv3Notify Function
Snmpv3Pswd2LocalizedAuthKeyMD5Hashing Function

C

```c
void Snmpv3Pswd2LocalizedAuthKeyMD5Hashing(
    UINT8* pswdToLocalized,
    UINT8 pswdLen
);
```

Description

This routine converts HMAC-MD5 authentication password key to localized key using snmpSngineID(RFC-3414).

Preconditions

SNMPv3Init() and ProcessVariabels() are called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pswdToLocalized</td>
<td>password storage poniter</td>
</tr>
<tr>
<td>pswdLen</td>
<td>password length.</td>
</tr>
</tbody>
</table>

Remarks

None

Stack API > SNMP > Functions > Snmpv3Pswd2LocalizedAuthKeyMD5Hashing Function
Snmpv3Pswd2LocalizedAuthKeySHAHashing Function

C

```c
void Snmpv3Pswd2LocalizedAuthKeySHAHashing(
    UINT8* pswdToLocalized,
    UINT8 pswdLen
);
```

**Description**

This routine converts HMAC-SHA authentication password key to localized key using snmpSngineID(RFC-3414).

**Preconditions**

SNMPv3Init() and ProcessVariabels() are called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pswdToLocalized</td>
<td>password storage poniter</td>
</tr>
<tr>
<td>pswdLen</td>
<td>password length.</td>
</tr>
</tbody>
</table>

**Remarks**

None
void Snmpv3ReportPdu(
    SNMPV3MSGDATA * dynScopedBufPtr
);

Description

The SNMPv3 PDU exchange starts with the agent sending a report pdu on reception of any Get_Request PDU for SNMPv3 request. This routine forms the report pdu for response to the requesting entity.

Preconditions

ProcessVariables() is called and a valid SNMPv3 request is received.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynScopedBufPtr</td>
<td>pointer to the response buffer memory where the 'report' response to be saved for transmission.</td>
</tr>
</tbody>
</table>

Remarks

None.
Snmpv3ScopedPduProcessing Function

C

```c
SNMP_ERR_STATUS Snmpv3ScopedPduProcessing(
    BYTE inOutPdu
);
```

Description

The received SNMPv3 PDU or the transmit PDU header has scoped pdu parameters like 'contextEngineID' 'context name' etc. This routine retrieves these parameters information from the stored incoming pdu or write the appropriate dynamically allocated memory for the transmit response PDU.

Preconditions

Valid SNMPv3 request msg is received.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inOutPdu</td>
<td>indicates whether the incoming PDU is to be read for scoped pdu parameters to be retrieved or the response PDU to be populated with these values</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP_NO_CREATION</td>
<td>Failure due to improper scoped pdu information format in the PDU or failure in constructing the response PDU.</td>
</tr>
<tr>
<td>SNMP_NO_ERR</td>
<td>The scoped parameters retrieval or response PDU formation is successful</td>
</tr>
</tbody>
</table>

Remarks
The scoped pDu parameters msg data : - [==

Stack API > SNMP > Functions > Snmpv3ScopedPduProcessing Function
Snmpv3SetErrorResponse Status Function

```c
void Snmpv3SetErrorResponse(
    WORD errorStatusOffset,
    WORD errorIndexOffset,
    SNMP_ERR_STATUS errorStatus,
    BYTE errorIndex,
    SNMPV3MSGDATA * dynScopedPduPutBuf
);
```

**Description**

This routine processes the received snmp Get request pdu for the variable binding in the request and also creates the response pdu.

**Preconditions**

`ProcessVariables()` is called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorStatusOffset</td>
<td>Offset to update error status in Response Tx pdu</td>
</tr>
<tr>
<td>errorIndexOffset</td>
<td>Offset to update error index</td>
</tr>
<tr>
<td>errorStatus</td>
<td>Snmp process error to be updated in response.</td>
</tr>
<tr>
<td>errorIndex</td>
<td>Index of the request varbind in the var bind list for which error status is to be updated.</td>
</tr>
<tr>
<td>dynScopedPduPutBuf</td>
<td>dynamic snmpv3 scoped pdu buffer</td>
</tr>
</tbody>
</table>

**Returns**

None.

**Remarks**
None.
**Snmpv3TrapScopedpdu Function**

```c
UINT8 Snmpv3TrapScopedpdu(
    SNMP_ID var,
    SNMP_VAL val,
    SNMP_INDEX index,
    UINT8 targetIndex
);
```

**Description**

This routine forms the trap scoped pdu header for the SNMPv3 trap PDU to be originated from this agent. Scoped pdu comprises of msg data : - [== For ASCII STR trap VAL(argument) contains the pointer address of the string variable.

**Preconditions**

TRAP event is triggered.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>var id of the variable whose value to be sent in the trap pdu</td>
</tr>
<tr>
<td>val</td>
<td>value of the variable</td>
</tr>
<tr>
<td>index</td>
<td>index of the variable in the multiple variable bind scenario targetIndex -index of the 'gSnmpv3TrapConfigData' table's security user name for which the TRAP PDU message header to constructed.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The trap scoped pdu header generation is successful.</td>
</tr>
</tbody>
</table>
TRUE | FALSE - The trap scoped pdu header generation failed.

Remarks

None.

Stack API > SNMP > Functions > Snmpv3TrapScopedpdu Function
Snmpv3UserSecurityModelProcessPDU Function

```
int Snmpv3UserSecurityModelProcessPDU(
    BYTE inOutPdu
);
```

Description

The received SNMPv3 PDU or the transmit PDU header has message security data bytes information. This routine retrieves the message security parameters information from the stored incoming pdu or write the appropriate security model info to the response msg buffer.

Preconditions

Valid SNMPv3 request msg is received.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inOutPdu</td>
<td>indicates whether the incoming PDU is to be read for user security model to be retrieved or the response PDU to be populated with these values</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP_NO_CREATION</td>
<td>Failure due to improper security model processing information format in the received PDU or failure in constructing the response PDU.</td>
</tr>
<tr>
<td>SNMP_NO_ERR</td>
<td>The user security model retrieval or response PDU formation is successful</td>
</tr>
</tbody>
</table>
Remarks

The user security parameter constitute the vital information for the message authentication and privacy of the message. The user security model parameters header structure:

```
MsgAuthEngnID+MsgAuthEngnBoots+MsgAuthEngnTime
+MsgUserName+MsgAuthParam+MsgPrivParam
```

Stack API > SNMP > Functions > Snmpv3UserSecurityModelProcessPDU Function
Snmpv3UsmAesEncryptDecryptInitVector Function

C

```c
void Snmpv3UsmAesEncryptDecryptInitVector(
    BYTE inOutPdu
);
```

Description

The IV is concatenated as: the 32-bit `snmpEngineBoots` is converted to the first 4 Octects and the `snmpEngineTime` converted to subsequent four bytes.

Preconditions

SNMPv3Init() and ProcessVariabels() are called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inOutPdu</td>
<td>SNMP request PDU</td>
</tr>
</tbody>
</table>

Remarks

None
Snmpv3UsmOutMsgAuthenticationParam
Function

C

```c
void Snmpv3UsmOutMsgAuthenticationParam(
    UINT8 hashType
);
```

Description

This routine prepares out message with HMAC-MD5 or HMAC-SHA1 authentication protocol. (RFC- 3414 - section 6)

Preconditions

SNMPv3Init() and ProcessVariabels() are called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hashType</td>
<td>authentication protocol type</td>
</tr>
</tbody>
</table>

Remarks

None
Snmpv3USMOutMsgPrivParam Function

```c
void Snmpv3USMOutMsgPrivParam();
```

**Description**

SNMPEngineTime is used to encrypt the outgoing message with a random value.

**Preconditions**

SNMPv3Init() and ProcessVariables() are called.

**Remarks**

None
Snmpv3UsmSnmpEngnAuthPrivPswdLocalization Function

C

void Snmpv3UsmSnmpEngnAuthPrivPswdLocalization(
    UINT8 userDBIndex
);

Description

This routine converts MD5 or SHA1 and AES privacy password key to localized key using snmpSngineID(RFC- 3414 - section 6).

Preconditions

SNMPv3Init() and ProcessVariabels() are called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userDBIndex</td>
<td>authentication protocol type</td>
</tr>
</tbody>
</table>

Remarks

None
Snmpv3ValidateEngineId Function

C

BOOL Snmpv3ValidateEngineId();

Description

This routine validates Engine ID.

Preconditions

SNMPv3Init() and ProcessVariabels() are called.

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>Failure</td>
</tr>
<tr>
<td>TRUE</td>
<td>Success</td>
</tr>
</tbody>
</table>

Remarks

None
Snmpv3ValidateSecNameAndSecLvl Function

C

BOOL Snmpv3ValidateSecNameAndSecLvl();

Description

This routine validates security name and security level with SNMP global data base for an incoming PDU.

Preconditions

SNMPv3Init() and ProcessVariables() are called.

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>success</td>
</tr>
<tr>
<td>FALSE</td>
<td>failure</td>
</tr>
</tbody>
</table>

Remarks

None

Stack API > SNMP > Functions > Snmpv3ValidateSecNameAndSecLvl Function
Snmpv3ValidateSecurityName Function

```c
BOOL Snmpv3ValidateSecurityName();
```

**Description**

This routine validates user name.

**Preconditions**

SNMPv3Init() and ProcessVariabels() are called.

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>Failure</td>
</tr>
<tr>
<td>TRUE</td>
<td>Success</td>
</tr>
</tbody>
</table>

**Remarks**

None
# Structs, Records, Enums

## Module

**SNMP**

### Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AccessCtrlSubSysIsAccessAllowed</strong></td>
<td>Applications are the typical clients of the service(s) of the Access Control Subsystem. The following primitive is provided by the Access Control Subsystem to check if access is allowed: statusInformation = -- success or errorIndication</td>
</tr>
<tr>
<td><strong>dispatcherProcessPdu</strong></td>
<td>Process Incoming Request or Notification PDU Dispatcher provides the following primitive to pass an incoming snmp pdu to an application.</td>
</tr>
<tr>
<td><strong>dispatcherStatusInfo</strong></td>
<td>Generate Outgoing Request or Notification statusInformation = -- sendPduHandle if success -- errorIndication if failure</td>
</tr>
<tr>
<td><strong>dispatcherReturnResponsePdu</strong></td>
<td>Generate Outgoing Response The PDU Dispatcher provides the following primitive for an application to return an SNMP Response PDU to the PDU Dispatcher: result = SUCCESS or FAILURE</td>
</tr>
<tr>
<td><strong>MsgProcModPrepareDataElements</strong></td>
<td>Prepare Data Elements from an Incoming SNMP Message The Message Processing Subsystem provides this service primitive for preparing the abstract data elements from an incoming SNMP message: result = -- SUCCESS or errorIndication</td>
</tr>
<tr>
<td><strong>Prepare Outgoing SNMP Request or</strong></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>MsgProcModPrepareOutgoingMessage</strong></td>
<td>Notification Message: The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Request or Notification Message.</td>
</tr>
<tr>
<td><strong>MsgProcModPrepareResponseMessage</strong></td>
<td>Prepare an Outgoing SNMP Response Message: The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Response Message: result = -- SUCCESS or FAILURE.</td>
</tr>
<tr>
<td><strong>processResponsePdu</strong></td>
<td>Process Incoming Response PDU: The PDU Dispatcher provides the following primitive to pass an incoming SNMP Response PDU to an application.</td>
</tr>
<tr>
<td><strong>registerContextEngineID</strong></td>
<td>success or errorIndication</td>
</tr>
<tr>
<td><strong>SecuritySysGenerateRequestMsg</strong></td>
<td>This is record SecuritySysGenerateRequestMsg.</td>
</tr>
<tr>
<td><strong>SecuritySysGenerateResponseMsg</strong></td>
<td>Generate a Response Message: The Security Subsystem provides the following primitive to generate a Response message:</td>
</tr>
<tr>
<td><strong>StateRelease</strong></td>
<td>Release State Reference Information: All Subsystems which pass stateReference information also provide a primitive to release the memory that holds the referenced state information.</td>
</tr>
<tr>
<td><strong>unregisterContextEngineID</strong></td>
<td>This is record unregisterContextEngineID.</td>
</tr>
</tbody>
</table>
## AccessCtrlSubSysIsAccessAllowed Structure

### C
```
struct AccessCtrlSubSysIsAccessAllowed {
    BYTE securityModel;
    BYTE* securityName;
    BYTE securityLevel;
    BYTE viewType;
    BYTE* contextName;
    BYTE* variableName;
};
```

### Description
Applications are the typical clients of the service(s) of the Access Control Subsystem. The following primitive is provided by the Access Control Subsystem to check if access is allowed:

```
statusInformation = -- success or errorIndication
```

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE securityModel;</td>
<td>Security Model in use</td>
</tr>
<tr>
<td>BYTE* securityName;</td>
<td>principal who wants to access</td>
</tr>
<tr>
<td>BYTE securityLevel;</td>
<td>Level of Security</td>
</tr>
<tr>
<td>BYTE viewType;</td>
<td>read, write, or notify view</td>
</tr>
<tr>
<td>BYTE* contextName;</td>
<td>context containing variableName</td>
</tr>
<tr>
<td>BYTE* variableName;</td>
<td>OID for the managed object</td>
</tr>
</tbody>
</table>

[Stack API] > [SNMP] > [Structs, Records, Enums] > [AccessCtrlSubSysIsAccessAllowed Structure]

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
[Contents] | [Index] | [Home]
dispatcherProcessPdu Structure

```c
struct dispatcherProcessPdu {
    BYTE messageProcessingModel;
    BYTE securityModel;
    BYTE* securityName;
    BYTE securityLevel;
    BYTE* contextEngineID;
    BYTE* contextName;
    BYTE pduVersion;
    BYTE* PDU;
    DWORD_VAL maxSizeResponseScopedPDU;
    UINT32 stateReference;
};
```

Description

Process Incoming Request or Notification PDU

Dispatcher provides the following primitive to pass an incoming snmp pdu to an application.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE messageProcessingModel;</td>
<td>typically, SNMP version</td>
</tr>
<tr>
<td>BYTE securityModel;</td>
<td>Security Model in use</td>
</tr>
<tr>
<td>BYTE* securityName;</td>
<td>on behalf of this principal</td>
</tr>
<tr>
<td>BYTE securityLevel;</td>
<td>Level of Security</td>
</tr>
<tr>
<td>BYTE* contextEngineID;</td>
<td>data from/at this SNMP entity</td>
</tr>
<tr>
<td>BYTE* contextName;</td>
<td>data from/in this context</td>
</tr>
<tr>
<td>BYTE pduVersion;</td>
<td>the version of the PDU</td>
</tr>
<tr>
<td>BYTE* PDU;</td>
<td>SNMP Protocol Data Unit</td>
</tr>
<tr>
<td>DWORD_VAL</td>
<td></td>
</tr>
<tr>
<td>maxiSizeResponseScopedPDU;</td>
<td></td>
</tr>
<tr>
<td>maximum size of the Response PDU</td>
<td></td>
</tr>
<tr>
<td>UINT32 stateReference;</td>
<td></td>
</tr>
<tr>
<td>reference to state information needed when sending a response</td>
<td></td>
</tr>
</tbody>
</table>

Stack API > SNMP > Structs, Records, Enums > dispatcherProcessPdu Structure
**dispatcherStatusInfo Structure**

```c
struct dispatcherStatusInfo {
    BYTE transportDomain;
    UINT32 transportAddress;
    BYTE messageProcessingModel;
    BYTE securityModel;
    BYTE* securityName;
    BYTE securityLevel;
    BYTE* contextEngineID;
    BYTE* contextName;
    BYTE pduVersion;
    BYTE* PDU;
    BOOL expectResponse;
};
```

**Description**

Generate Outgoing Request or Notification

```plaintext
statusInformation = -- sendPduHandle if success -- errorIndication if failure
```

**Members**

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE transportDomain;</td>
<td>transport domain to be used</td>
</tr>
<tr>
<td>UINT32 transportAddress;</td>
<td>transport address to be used</td>
</tr>
<tr>
<td>BYTE messageProcessingModel;</td>
<td>typically, SNMP version</td>
</tr>
<tr>
<td>BYTE securityModel;</td>
<td>Security Model to use</td>
</tr>
<tr>
<td>BYTE* securityName;</td>
<td>on behalf of this principal</td>
</tr>
<tr>
<td>BYTE securityLevel;</td>
<td>Level of Security requested</td>
</tr>
<tr>
<td>BYTE* contextEngineID;</td>
<td>data from/at this entity</td>
</tr>
<tr>
<td>BYTE* contextName;</td>
<td>data from/in this context</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>BYTE pduVersion;</td>
<td>the version of the PDU</td>
</tr>
<tr>
<td>BYTE* PDU;</td>
<td>SNMP Protocol Data Unit</td>
</tr>
<tr>
<td>BOOL expectResponse;</td>
<td>TRUE or FALSE</td>
</tr>
</tbody>
</table>
dispatherReturnResponsePdu Structure

C

```c
struct dispatherReturnResponsePdu {
    BYTE messageProcessingModel;
    BYTE securityModel;
    BYTE* securityName;
    BYTE securityLevel;
    BYTE* contextEngineID;
    BYTE* contextName;
    BYTE pduVersion;
    BYTE* PDU;
    UINT32 maxSizeResponseScopedPDU;
    UINT32 stateReference;
    statusInformation statInfo;
};
```

Description

Generate Outgoing Response

The PDU Dispatcher provides the following primitive for an application to return an SNMP Response PDU to the PDU Dispatcher:

```c
result = SUCCESS or FAILURE
```

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE messageProcessingModel;</td>
<td>typically, SNMP version</td>
</tr>
<tr>
<td>BYTE securityModel;</td>
<td>Security Model in use</td>
</tr>
<tr>
<td>BYTE* securityName;</td>
<td>on behalf of this principal</td>
</tr>
<tr>
<td>BYTE securityLevel;</td>
<td>same as on incoming request</td>
</tr>
<tr>
<td>BYTE* contextEngineID;</td>
<td>data from/at this SNMP entity</td>
</tr>
<tr>
<td>BYTE* contextName;</td>
<td>data from/in this context</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BYTE pduVersion;</td>
<td>the version of the PDU</td>
</tr>
<tr>
<td>BYTE* PDU;</td>
<td>SNMP Protocol Data Unit</td>
</tr>
<tr>
<td>UINT32 maxSizeResponseScopedPDU;</td>
<td>maximum size sender can accept</td>
</tr>
<tr>
<td>UINT32 stateReference;</td>
<td>reference to state information as presented with the request</td>
</tr>
<tr>
<td>statusInformation statInfo;</td>
<td>success or errorIndication, error counter OID/value if error</td>
</tr>
</tbody>
</table>
MsgProcModPrepareDataElements Structure

C

```c
struct MsgProcModPrepareDataElements {
    BYTE transportDomain;
    UINT32 transportAddress;
    BYTE* wholeMsg;
    UINT32 wholeMsgLength;
    BYTE messageProcessingModel;
    BYTE securityModel;
    BYTE* securityName;
    BYTE securityLevel;
    BYTE* contextEngineID;
    BYTE* contextName;
    BYTE pduVersion;
    BYTE* PDU;
    BYTE pduType;
    UINT32 maxSizeResponseScopedPDU;
    statusInformation statInfo;
    UINT32 stateReference;
};
```

Description

Prepare Data Elements from an Incoming SNMP Message

The Message Processing Subsystem provides this service primitive for preparing the abstract data elements from an incoming SNMP message: result = -- SUCCESS or errorIndication

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE transportDomain;</td>
<td>origin transport domain</td>
</tr>
<tr>
<td>UINT32 transportAddress;</td>
<td>origin transport address</td>
</tr>
<tr>
<td>BYTE* wholeMsg;</td>
<td>as received from the network</td>
</tr>
<tr>
<td>UINT32 wholeMsgLength;</td>
<td>as received from the network</td>
</tr>
<tr>
<td>BYTE</td>
<td>typically, SNMP version</td>
</tr>
<tr>
<td>Message Processing Model Structure</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>messageProcessingModel;</td>
<td></td>
</tr>
<tr>
<td>BYTE securityModel;</td>
<td>Security Model to use</td>
</tr>
<tr>
<td>BYTE* securityName;</td>
<td>on behalf of this principal</td>
</tr>
<tr>
<td>BYTE securityLevel;</td>
<td>Level of Security requested</td>
</tr>
<tr>
<td>BYTE* contextEngineID;</td>
<td>data from/at this entity</td>
</tr>
<tr>
<td>BYTE* contextName;</td>
<td>data from/in this context</td>
</tr>
<tr>
<td>BYTE pduVersion;</td>
<td>the version of the PDU</td>
</tr>
<tr>
<td>BYTE* PDU;</td>
<td>SNMP Protocol Data Unit</td>
</tr>
<tr>
<td>BYTE pduType;</td>
<td>SNMP PDU type OUT sendPduHandle; // handle for matched request</td>
</tr>
<tr>
<td>UINT32 maxSizeResponseScopedPDU;</td>
<td>maximum size sender can accept</td>
</tr>
<tr>
<td>statusInformation statInfo;</td>
<td>success or errorIndication error counter OID/value if error</td>
</tr>
<tr>
<td>UINT32 stateReference;</td>
<td>reference to state information to be used for possible Response</td>
</tr>
</tbody>
</table>

Stack API > SNMP > Structs, Records, Enums > MsgProcModPrepareDataElements Structure

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
### MsgProcModPrepareOutgoingMessage Structure

```c
struct MsgProcModPrepareOutgoingMessage {
    BYTE transportDomain;
    UINT32 transportAddress;
    BYTE messageProcessingModel;
    BYTE securityModel;
    BYTE* securityName;
    BYTE securityLevel;
    BYTE* contextEngineID;
    BYTE* contextName;
    BYTE pduVersion;
    BYTE* PDU;
    BOOL expectResponse;
    BYTE destTransportDomain;
    UINT32 destTransportAddress;
    BYTE* outgoingMessage;
    UINT32 outgoingMessageLength;
};
```

### Description

Prepare Outgoing SNMP Request or Notification Message

The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Request or Notification Message

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE transportDomain;</td>
<td>transport domain to be used</td>
</tr>
<tr>
<td>UINT32 transportAddress;</td>
<td>transport address to be used</td>
</tr>
<tr>
<td>BYTE messageProcessingModel;</td>
<td>typically, SNMP version</td>
</tr>
<tr>
<td>BYTE securityModel;</td>
<td>Security Model to use</td>
</tr>
<tr>
<td>BYTE* securityName;</td>
<td>on behalf of this principal</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>BYTE securityLevel;</td>
<td>Level of Security requested</td>
</tr>
<tr>
<td>BYTE* contextEngineID;</td>
<td>data from/at this entity</td>
</tr>
<tr>
<td>BYTE* contextName;</td>
<td>data from/in this context</td>
</tr>
<tr>
<td>BYTE pduVersion;</td>
<td>the version of the PDU</td>
</tr>
<tr>
<td>BYTE* PDU;</td>
<td>SNMP Protocol Data Unit</td>
</tr>
<tr>
<td>BOOL expectResponse;</td>
<td>TRUE or FALSE IN sendPduHandle; //the handle for matching incoming responses</td>
</tr>
<tr>
<td>BYTE destTransportDomain;</td>
<td>destination transport domain</td>
</tr>
<tr>
<td>UINT32 destTransportAddress;</td>
<td>destination transport address</td>
</tr>
<tr>
<td>BYTE* outgoingMessage;</td>
<td>the message to send</td>
</tr>
<tr>
<td>UINT32 outgoingMessageLength;</td>
<td>its length</td>
</tr>
</tbody>
</table>
**MsgProcModPrepareResponseMessage Structure**

```c
struct MsgProcModPrepareResponseMessage {
    BYTE messageProcessingModel;
    BYTE securityModel;
    BYTE* securityName;
    BYTE securityLevel;
    BYTE* contextEngineID;
    BYTE* contextName;
    BYTE pduVersion;
    BYTE* PDU;
    UINT32 maxSizeResponseScopedPDU;
    UINT32 stateReference;
    statusInformation statInfo;
    BYTE destTransportDomain;
    UINT32 destTransportAddress;
    BYTE* outgoingMessage;
    UINT32 outgoingMessageLength;
};
```

**Description**

Prepare an Outgoing SNMP Response Message

The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Response Message: result = -- SUCCESS or FAILURE

**Members**

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE messageProcessingModel;</td>
<td>typically, SNMP version</td>
</tr>
<tr>
<td>BYTE securityModel;</td>
<td>same as on incoming request</td>
</tr>
<tr>
<td>BYTE* securityName;</td>
<td>same as on incoming request</td>
</tr>
<tr>
<td>BYTE securityLevel;</td>
<td>same as on incoming request</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>BYTE* contextEngineID;</td>
<td>data from/at this SNMP entity</td>
</tr>
<tr>
<td>BYTE* contextName;</td>
<td>data from/in this context</td>
</tr>
<tr>
<td>BYTE pduVersion;</td>
<td>the version of the PDU</td>
</tr>
<tr>
<td>BYTE* PDU;</td>
<td>SNMP Protocol Data Unit</td>
</tr>
<tr>
<td>UINT32 maxSizeResponseScopedPDU;</td>
<td>maximum size able to accept</td>
</tr>
<tr>
<td>UINT32 stateReference;</td>
<td>reference to state information as presented with the request</td>
</tr>
<tr>
<td>statusInformation statInfo;</td>
<td>success or errorIndication, error counter OID/value if error</td>
</tr>
<tr>
<td>BYTE destTransportDomain;</td>
<td>destination transport domain</td>
</tr>
<tr>
<td>UINT32 destTransportAddress;</td>
<td>destination transport address</td>
</tr>
<tr>
<td>BYTE* outgoingMessage;</td>
<td>the message to send</td>
</tr>
<tr>
<td>UINT32 outgoingMessageLength;</td>
<td>its length</td>
</tr>
</tbody>
</table>
processResponsePdu Structure

C

```c
struct processResponsePdu {
    BYTE messageProcessingModel;
    BYTE securityModel;
    BYTE* securityName;
    BYTE securityLevel;
    BYTE* contextEngineID;
    BYTE* contextName;
    BYTE pduVersion;
    BYTE* PDU;
    statusInformation statInfo;
};
```

Description

Process Incoming Response PDU

The PDU Dispatcher provides the following primitive to pass an incoming SNMP Response PDU to an application:

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE messageProcessingModel;</td>
<td>typically, SNMP version</td>
</tr>
<tr>
<td>BYTE securityModel;</td>
<td>Security Model in use</td>
</tr>
<tr>
<td>BYTE* securityName;</td>
<td>on behalf of this principal</td>
</tr>
<tr>
<td>BYTE securityLevel;</td>
<td>Level of Security</td>
</tr>
<tr>
<td>BYTE* contextEngineID;</td>
<td>data from/at this SNMP entity</td>
</tr>
<tr>
<td>BYTE* contextName;</td>
<td>data from/in this context</td>
</tr>
<tr>
<td>BYTE pduVersion;</td>
<td>the version of the PDU</td>
</tr>
<tr>
<td>BYTE* PDU;</td>
<td>SNMP Protocol Data Unit</td>
</tr>
</tbody>
</table>
statusInformation statInfo;

success or errorIndication IN sendPduHandle; //handle from sendPdu
SecuritySysGenerateRequestMsg Structure

C

```c
struct SecuritySysGenerateRequestMsg {
    BYTE messageProcessingModel;
    BYTE* globalData;
    UINT32 maxMessageSize;
    BYTE securityModel;
    BYTE* securityEngineID;
    BYTE* securityName;
    BYTE securityLevel;
    BYTE* scopedPDU;
    BYTE* wholeMsg;
    UINT32 wholeMsgLength;
};
```

Description

This is record SecuritySysGenerateRequestMsg.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE messageProcessingModel;</td>
<td>typically, SNMP version</td>
</tr>
<tr>
<td>BYTE* globalData;</td>
<td>message header, admin data</td>
</tr>
<tr>
<td>UINT32 maxMessageSize;</td>
<td>of the sending SNMP entity</td>
</tr>
<tr>
<td>BYTE securityModel;</td>
<td>for the outgoing message</td>
</tr>
<tr>
<td>BYTE* securityEngineID;</td>
<td>authoritative SNMP entity</td>
</tr>
<tr>
<td>BYTE* securityName;</td>
<td>on behalf of this principal</td>
</tr>
<tr>
<td>BYTE securityLevel;</td>
<td>Level of Security requested</td>
</tr>
<tr>
<td>BYTE* scopedPDU;</td>
<td>message (plaintext) payload OUT securityParameters; //filled in by Security Module</td>
</tr>
<tr>
<td>BYTE* wholeMsg;</td>
<td>complete generated message</td>
</tr>
</tbody>
</table>
UINT32 wholeMsgLength;

length of the generated message
SecuritySysGenerateResponseMsg Structure

C

```c
struct SecuritySysGenerateResponseMsg {
    BYTE messageProcessingModel;
    BYTE* globalData;
    UINT32 maxMessageSize;
    BYTE securityModel;
    BYTE* securityEngineID;
    BYTE* securityName;
    BYTE securityLevel;
    BYTE* scopedPDU;
    BYTE* wholeMsg;
    UINT32 wholeMsgLength;
};
```

Description

Generate a Response Message

The Security Subsystem provides the following primitive to generate a Response message:

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE messageProcessingModel;</td>
<td>typically, SNMP version</td>
</tr>
<tr>
<td>BYTE* globalData;</td>
<td>message header, admin data</td>
</tr>
<tr>
<td>UINT32 maxMessageSize;</td>
<td>of the sending SNMP entity</td>
</tr>
<tr>
<td>BYTE securityModel;</td>
<td>for the outgoing message</td>
</tr>
<tr>
<td>BYTE* securityEngineID;</td>
<td>authoritative SNMP entity</td>
</tr>
<tr>
<td>BYTE* securityName;</td>
<td>on behalf of this principal</td>
</tr>
<tr>
<td>BYTE securityLevel;</td>
<td>for the outgoing message</td>
</tr>
<tr>
<td>BYTE* scopedPDU;</td>
<td>message (plaintext) payload IN securityStateReference;</td>
</tr>
</tbody>
</table>
BYTE* scopedPDU;
//reference to security state information from original request OUT securityParameters; //filled in by Security Module

BYTE* wholeMsg;
complete generated message

UINT32 wholeMsgLength;
length of the generated message

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
StateRelease Structure

C

```c
struct StateRelease {
    UINT32 stateReference;
};
```

Description

Release State Reference Information

All Subsystems which pass stateReference information also provide a primitive to release the memory that holds the referenced state information.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT32 stateReference;</td>
<td>handle of reference to be released</td>
</tr>
</tbody>
</table>

Stack API > SNMP > Structs, Records, Enums > StateRelease Structure
unregisterContextEngineID Structure

C

```c
struct unregisterContextEngineID {
    BYTE* contextEngineID;
    BYTE pduType;
};
```

Description

This is record unregisterContextEngineID.

Members

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Members</td>
<td>Description</td>
</tr>
<tr>
<td>BYTE* contextEngineID;</td>
<td>give up responsibility for this one</td>
</tr>
<tr>
<td>BYTE pduType;</td>
<td>the pduType(s) to be unregistered</td>
</tr>
</tbody>
</table>

Stack API > SNMP > Structs, Records, Enums > unregisterContextEngineID Structure
## Types

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INOUT_SNMP_PDU</td>
<td>This is type INOUT_SNMP_PDU.</td>
</tr>
<tr>
<td>REPORT_FLAG_AND_SECURITY_LEVEL_FLAGS</td>
<td>This is type REPORT_FLAG_AND_SECURITY_LEVEL_FLAGS.</td>
</tr>
<tr>
<td>SNMP_ENGNID_OCTET_IDENTIFIER_VAL</td>
<td>The fifth octet indicates how the rest (6th and following octets) are formatted. Refer to RFC3411 section 5 Page 41.</td>
</tr>
<tr>
<td>SNMPV3_HMAC_HASH_TYPE</td>
<td>Type of hash being calculated.</td>
</tr>
<tr>
<td>SNMPV3_MSG_AUTH_SEC_PARAM_RESULT</td>
<td>This is type SNMPV3_MSG_AUTH_SEC_PARAM_RESULT.</td>
</tr>
<tr>
<td>SNMPV3_MSG_PRIV_SEC_PARAM_RESULT</td>
<td>This is type SNMPV3_MSG_PRIV_SEC_PARAM_RESULT.</td>
</tr>
<tr>
<td>SNMPV3_PRIV_PROT_TYPE</td>
<td>This is type SNMPV3_PRIV_PROT_TYPE.</td>
</tr>
<tr>
<td>STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL</td>
<td>This is type STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL.</td>
</tr>
<tr>
<td>STD_BASED_SNMP_SECURITY_MODEL</td>
<td>Snmp Message Processing Model</td>
</tr>
<tr>
<td>STD_BASED_SNMPV3_SECURITY_LEVEL</td>
<td>This is type STD_BASED_SNMPV3_SECURITY_LEVEL.</td>
</tr>
<tr>
<td>USM_SECURITY_LEVEL</td>
<td>This is type USM_SECURITY_LEVEL.</td>
</tr>
</tbody>
</table>

### Module

**SNMP**

### Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This is type</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>SNMPNONMIBRECDINFO</td>
<td>This is type SNMPNONMIBRECDINFO.</td>
</tr>
<tr>
<td>SNMPV3REQUEST_WHOLEMSG</td>
<td>This is type SNMPV3_REQUEST_WHOLEMSG.</td>
</tr>
<tr>
<td>SNMPV3RESPONSE_WHOLEMSG</td>
<td>This is type SNMPV3_RESPONSE_WHOLEMSG.</td>
</tr>
<tr>
<td>snmpV3EngnUserDataBase</td>
<td>This is type snmpV3EngnUserDataBase.</td>
</tr>
<tr>
<td>SNMPV3MSGDATA</td>
<td>SNMPv3</td>
</tr>
<tr>
<td>snmpV3TrapConfigDataBase</td>
<td>snmpv3 target configuration with respect to trap.</td>
</tr>
<tr>
<td>statusInformation</td>
<td>success or errorIndication</td>
</tr>
</tbody>
</table>

Stack API > SNMP > Types

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
INOUT_SNMP_PDU Enumeration

C

```c
typedef enum {
    SNMP_RESPONSE_PDU = 0x01,
    SNMP_REQUEST_PDU = 0x02
} INOUT_SNMP_PDU;
```

Description

This is type INOUT_SNMP_PDU.
REPORT_FLAG_AND_SECURITY_LEVEL_FLAGS

Description

This is type REPORT_FLAG_AND_SECURITY_LEVEL_FLAGS.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_REPORT_NO_PRIVACY_NO_AUTH = 0x00</td>
<td>00000000b</td>
</tr>
<tr>
<td>NO_REPORT_NO_PRIVACY_BUT_AUTH_PROVIDED = 0x01</td>
<td>00000001b</td>
</tr>
<tr>
<td>NO_REPORT_PRIVACY_PROVIDED_BUT_NO_AUTH = 0x02</td>
<td>00000010b Priv without Auth is not allowed</td>
</tr>
<tr>
<td>NO_REPORT_PRIVACY_AND_AUTH_PROVIDED = 0x03</td>
<td>00000011b</td>
</tr>
<tr>
<td>REPORT2REQ_NO_PRIVACY_NO_AUTH = 0x04</td>
<td>00000100b</td>
</tr>
<tr>
<td>REPORT2REQ_NO_PRIVACY_BUT_AUTH_PROVIDED = 0x05</td>
<td>00000101b</td>
</tr>
<tr>
<td>REPORT2REQ_PRIVACY_PROVIDED_BUT_NO_AUTH = 0x06</td>
<td>00000110b Priv without Auth is not allowed</td>
</tr>
<tr>
<td>REPORT2REQ_PRIVACY_AND_AUTH_PROVIDED = 0x07</td>
<td>00000111b</td>
</tr>
</tbody>
</table>
REPORT2REQ_PRIVACY_AND_AUTH_PROVIDED = 0x07

Stack API > SNMP > Types > REPORT_FLAG_AND_SECURITY_LEVEL_FLAGS Enumeration
SecuritySysProcessIncomingMsg Structure

```c
typedef struct {
    UINT32 maxMessageSize;
    UINT32 wholeMsgLength;
    BYTE* wholeMsg;
    BYTE* securityEngineID;
    BYTE* securityName;
    BYTE* scopedPDU;
    UINT32 maxSizeResponseScopedPDU;
    BYTE messageProcessingModel;
    BYTE securityModel;
    BYTE securityLevel;
    UINT8 securityEngineIDLen;
    BYTE securityNameLength;
} SecuritySysProcessIncomingMsg;
```

Description

This is type SecuritySysProcessIncomingMsg.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT32 maxMessageSize;</td>
<td>of the sending SNMP entity IN securityParameters; //for the received message</td>
</tr>
<tr>
<td>UINT32 wholeMsgLength;</td>
<td>length as received on the wire</td>
</tr>
<tr>
<td>BYTE* wholeMsg;</td>
<td>as received on the wire</td>
</tr>
<tr>
<td>BYTE* securityEngineID;</td>
<td>authoritative SNMP entity</td>
</tr>
<tr>
<td>BYTE* securityName;</td>
<td>identification of the principal</td>
</tr>
<tr>
<td>BYTE* scopedPDU;</td>
<td>message (plaintext) payload OUT</td>
</tr>
<tr>
<td>UINT32 maxSizeResponseScopedPDU;</td>
<td>securityStateReference; //reference to security state</td>
</tr>
<tr>
<td>BYTE</td>
<td>maximum size sender can handle</td>
</tr>
<tr>
<td>messageProcessingModel;</td>
<td>typically, SNMP version</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>BYTE securityModel;</td>
<td>for the received message</td>
</tr>
<tr>
<td>BYTE securityLevel;</td>
<td>Level of Security</td>
</tr>
<tr>
<td>UINT8 securityEngineIDLen;</td>
<td>authoritative SNMP entity</td>
</tr>
</tbody>
</table>
### SNMP_ENGNID_OCTET_IDENTIFIER_VAL Enumeration

```c
typedef enum {
    RESERVED = 0x0,
    IPV4_ADDR_ENGN_ID = 0x01,
    IPV6_ADDR_ENGN_ID = 0x02,
    MAC_ADDR_ENGN_ID = 0x03,
    ADMIN_ASSIGNED_TEXT = 0x04,
    ADMIN_ASSIGNED_OCTETS = 0x05,
    RESERVED_UNUSED = 0x06,
    ENTERPRISE_DEFINED = 128
} SNMP_ENGNID_OCTET_IDENTIFIER_VAL;
```

### Description

The fifth octet indicates how the rest (6th and following octets) are formatted. Refer to RFC3411 section5 Page# 41

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPV4_ADDR_ENGN_ID = 0x01</td>
<td>4octets</td>
</tr>
<tr>
<td>IPV6_ADDR_ENGN_ID = 0x02</td>
<td>16 octets</td>
</tr>
<tr>
<td>MAC_ADDR_ENGN_ID = 0x03</td>
<td>6 octets</td>
</tr>
<tr>
<td>RESERVED_UNUSED = 0x06</td>
<td>6 to 127 are reserved and unused</td>
</tr>
<tr>
<td>ENTERPRISE_DEFINED = 128</td>
<td>128 to 255 as defined by the enterprise maximum remaining length</td>
</tr>
</tbody>
</table>
SNMPNONMIBRECDINFO Structure

C

typedef struct {
    UINT8 oidstr[16];
    UINT8 version;
} SNMPNONMIBRECDINFO;

Description

This is type SNMPNONMIBRECDINFO.
SNMPV3_HMAC_HASH_TYPE Enumeration

C

typedef enum {
    SNMPV3_HAMC_MD5 = 0u,
    SNMPV3_HMAC_SHA1,
    SNMPV3_NO_HMAC_AUTH
} SNMPV3_HMAC_HASH_TYPE;

Description

Type of hash being calculated

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMPV3_HAMC_MD5 = 0u</td>
<td>MD5 is being calculated</td>
</tr>
<tr>
<td>SNMPV3_HMAC_SHA1</td>
<td>SHA-1 is being calculated</td>
</tr>
</tbody>
</table>

Stack API  >  SNMP  >  Types  >  SNMPV3_HMAC_HASH_TYPE Enumeration
SNMPV3_MSG_AUTH_SEC_PARAM_RESULT

Enumeration

c
typedef enum {
    SNMPV3_MSG_AUTH_FAIL = 0x00,
    SNMPV3_MSG_AUTH_PASS = 0x01
} SNMPV3_MSG_AUTH_SEC_PARAM_RESULT;

Description

This is type SNMPV3_MSG_AUTH_SEC_PARAM_RESULT.
SNMPV3_MSG_PRIV_SEC_PARAM_RESULT Enumeration

```
C
typedef enum {
    SNMPV3_MSG_PRIV_FAIL = 0x00,
    SNMPV3_MSG_PRIV_PASS = 0x01
} SNMPV3_MSG_PRIV_SEC_PARAM_RESULT;
```

Description

This is type SNMPV3_MSG_PRIV_SEC_PARAM_RESULT.
SNMPV3_PRIV PROT_TYPE Enumeration

C

typedef enum {
    SNMPV3_DES_PRIV = 0x0,
    SNMPV3_AES_PRIV,
    SNMPV3_NO_PRIV
} SNMPV3_PRIV PROT TYPE;

Description

This is type SNMPV3_PRIV PROT TYPE.
typedef struct {
    UINT8* wholeMsgHead;
    UINT8* snmpMsgHead;
    WORD_VAL wholeMsgLen;
    WORD_VAL snmpMsgLen;
    WORD msgAuthParamOffsetInWholeMsg;
    WORD scopedPduOffset;
    BYTE scopedPduAuthStructVal;
    WORD scopedPduStructLen;
} SNMPV3_REQUEST_WHOLEMSG;

Description

This is type SNMPV3_REQUEST_WHOLEMSG.
### SNMPV3_RESPONSE_WHOLEMSG Structure

C

```c
typedef struct {
    UINT8* wholeMsgHead;
    UINT8* snmpMsgHead;
    WORD wholeMsgLen;
    WORD snmpMsgLen;
    UINT8* msgAuthParamOffsetOutWholeMsg;
    UINT8* scopedPduOffset;
    WORD scopedPduStructLen;
    BYTE scopedPduAuthStructVal;
} SNMPV3_RESPONSE_WHOLEMSG;
```

### Description

This is type SNMPV3_RESPONSE_WHOLEMSG.
snmpV3EngnUserDataBase Structure

```c
typedef struct {
    UINT8 userName[SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE];
    UINT8 userAuthPswd[SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE];
    UINT8 userPrivPswd[SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE];
    UINT8 userAuthPswdLoclizdKey[SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE];
    UINT8 userPrivPswdLoclizdKey[SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE];
    UINT8 userDBIndex;
    UINT8 userHashType;
    UINT8 userNameLength;
    UINT8 userAuthPswdLen;
    UINT8 userPrivPswdLen;
    UINT8 userPrivType;
    UINT8 userAuthLocalKeyHmacIp[64];
    UINT8 userAuthLocalKeyHmacOp[64];
} snmpV3EngnUserDataBase;
```

Description

This is type snmpV3EngnUserDataBase.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8 userAuthPswd[SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE];</td>
<td>RFC specifies not to save password with the managed nodes instead store pswd ipad and opad values.</td>
</tr>
<tr>
<td>UINT8 userAuthLocalKeyHmacIp[64];</td>
<td>UINT8 userPrivType;</td>
</tr>
</tbody>
</table>
SNMPV3MSGDATA Structure

C

typedef struct {
    UINT8 * head;
    WORD length;
    WORD maxlength;
    WORD msgAuthParamOffset;
} SNMPV3MSGDATA;

Description

SNMPv3
### Description

snmpv3 target configuration with respect to trap.
**statusInformation Structure**

```c
typedef struct registerContextEngineID {
    BYTE* contextEngineID;
    BYTE pduType;
} statusInformation;
```

**Description**

success or errorIndication

**Members**

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE* contextEngineID;</td>
<td>take responsibility for this one</td>
</tr>
<tr>
<td>BYTE pduType;</td>
<td>the pduType(s) to be registered</td>
</tr>
</tbody>
</table>

[Stack API] > [SNMP] > [Types] > [statusInformation Structure]
C

typedef enum {
    SNMPV1_MSG_PROCESSING_MODEL = 0X00,
    SNMPV2C_MSG_PROCESSING_MODEL = 0X01,
    SNMPV2U_V2_MSG_PROCESSING_MODEL = 0X02,
    SNMPV3_MSG_PROCESSING_MODEL = 0X03
} STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL;

Description

This is type
STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL.
STD_BASED_SNMP_SECURITY_MODEL

Enumeration

C

typedef enum {
    ANY_SECURITY_MODEL = 0x00,
    SNMPV1_SECURITY_MODEL = 0x01,
    SNMPV2C_SECURITY_MODEL = 0x02,
    SNMPV3_USM_SECURITY_MODEL = 0x03
} STD_BASED_SNMP_SECURITY_MODEL;

Description

Snmp Message Processing Model

Stack API > SNMP > Types > STD_BASED_SNMP_SECURITY_MODEL Enumeration
STD_BASED_SNMPV3_SECURITY_LEVEL

Enumeration

c
typedef enum {
    NO_AUTH_NO_PRIV = 1,
    AUTH_NO_PRIV,
    AUTH_PRIV
} STD_BASED_SNMPV3_SECURITY_LEVEL;

Description

This is type STD_BASED_SNMPV3_SECURITY_LEVEL.
USM_SECURITY_LEVEL Enumeration

```
C

typedef enum {
    noAuthProtocol = 0x1,
    hmacMD5Auth,
    hmacSHAAuth,
    noPrivProtocol,
    desPrivProtocol = 0x5,
    aesPrivProtocol = 0x6
} USM_SECURITY_LEVEL;
```

Description

This is type USM_SECURITY_LEVEL.
### Variables

#### Module

**SNMP**

#### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authKey_iPad</td>
<td>This is variable authKey_iPad.</td>
</tr>
<tr>
<td>authKey_oPad</td>
<td>This is variable authKey_oPad.</td>
</tr>
<tr>
<td>authoritativeSnmpEngineBoots</td>
<td>The number of times that the authoritative SNMP engine has (re-)initialized itself since its snmpEngineID was last configured.</td>
</tr>
<tr>
<td>authoritativeSnmpEngineTime</td>
<td>The number of seconds since the value of the authoritativeSnmpEngineBoots object last changed</td>
</tr>
<tr>
<td>cipher_text</td>
<td>This is variable cipher_text.</td>
</tr>
<tr>
<td>deciphered_text</td>
<td>This is variable deciphered_text.</td>
</tr>
<tr>
<td>getZeroInstance</td>
<td>This variable is used for gext next request for zero instance</td>
</tr>
<tr>
<td>gSnmpV3InPduWholeMsgBuf</td>
<td>Dynamic memory stub and PDU details for Incoming stored PDU</td>
</tr>
<tr>
<td>gSnmpV3OUTPduWholeMsgBuf</td>
<td>Dynamic memory stub details and constructed outgoing stored PDU details</td>
</tr>
<tr>
<td>gSNMPv3PduHeaderBuf</td>
<td>Response PDU construction offset details</td>
</tr>
<tr>
<td>gSNMPv3ScopedPduDataPos</td>
<td>Offset to read scoped PDU data bytes for processing from dynamic memory stub</td>
</tr>
<tr>
<td>gSNMPv3ScopedPduRequestBuf</td>
<td>Stored request scoped pdu details</td>
</tr>
<tr>
<td>gSNMPv3ScopedPduResponseBuf</td>
<td>Processed response scoped pdu details</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>gSnmpV3TrapConfigData</td>
<td>SNMPv3 global configuration database to be used for trap notification</td>
</tr>
<tr>
<td>gSNMPv3TrapMsgHeaderBuf</td>
<td>TRAP message PDU header construction offset details</td>
</tr>
<tr>
<td>gSnmpV3TrapOUTPduWholeMsgBuf</td>
<td>Dynamic memory stub details and constructed trap PDU details</td>
</tr>
<tr>
<td>gSNMPv3TrapScopedPduResponseBuf</td>
<td>TRAP scoped PDU construction offset details</td>
</tr>
<tr>
<td>gSNMPV3TrapSecurityLevel</td>
<td>This is variable gSNMPV3TrapSecurityLevel.</td>
</tr>
<tr>
<td>gSnmpv3UserDBIndex</td>
<td>Index to the particular reference configured in User security model data base snmpV3UserDataBase.</td>
</tr>
<tr>
<td>gUsmStatsEngineID</td>
<td>Global variable to find out how many times SNMPv3 engine id has been validated</td>
</tr>
<tr>
<td>hmacAuthKeyBuf</td>
<td>This is variable hmacAuthKeyBuf.</td>
</tr>
<tr>
<td>HmacMd5Digest</td>
<td>This is variable HmacMd5Digest.</td>
</tr>
<tr>
<td>HmacSHADigest</td>
<td>This is variable HmacSHADigest.</td>
</tr>
<tr>
<td>incomingPdu</td>
<td>Incoming PDU details</td>
</tr>
<tr>
<td>incomingSnmpPDUmsgID</td>
<td>Retrieved Incoming Msg ID value from PDU</td>
</tr>
<tr>
<td>ivEncrptKeyOut</td>
<td>This is variable ivEncrptKeyOut.</td>
</tr>
<tr>
<td>md5LocalizedAuthKey</td>
<td>This is variable md5LocalizedAuthKey.</td>
</tr>
<tr>
<td>msgSecrtyParamLenOffset</td>
<td>This is variable msgSecrtyParamLenOffset.</td>
</tr>
<tr>
<td>securityPrimitivesOfIncomingPdu</td>
<td>Incoming PDU Security primitive details.</td>
</tr>
<tr>
<td>session_key</td>
<td>This is variable session_key.</td>
</tr>
<tr>
<td>sha1LocalizedAuthKey</td>
<td>This is variable sha1LocalizedAuthKey.</td>
</tr>
<tr>
<td>snmpEngineBoots</td>
<td>The number of times that the SNMP engine has (re-)initialized itself since</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>snmpEngineID</strong></td>
<td>Reserving 32 bytes for the <code>snmpEngineID</code> as the octet string length can vary from 5 to 32 bytes.</td>
</tr>
<tr>
<td><strong>snmpEngineMaxMessageSize</strong></td>
<td>The maximum message size the SNMP engine can handle.</td>
</tr>
<tr>
<td><strong>snmpEngineMsgProcessModel</strong></td>
<td>Type of Message processing model used. Value Maximum range (2^31-1), RFC3411</td>
</tr>
<tr>
<td><strong>snmpEngineSecurityModel</strong></td>
<td>Type of security model used. Value Maximum range (2^31-1), RFC3411</td>
</tr>
<tr>
<td><strong>snmpEngineTime</strong></td>
<td>The number of seconds since the value of the <code>snmpEngineBoots</code> object last changed.</td>
</tr>
<tr>
<td><strong>snmpEngineTimeOffset</strong></td>
<td>Stores the time value in seconds since SNMP Engine reset.</td>
</tr>
<tr>
<td><strong>snmpEngnIDLength</strong></td>
<td>Engine ID length of the SNMP Engine.</td>
</tr>
<tr>
<td><strong>snmpInMsgAuthParamLen</strong></td>
<td>Incoming SNMPv3 msg authentication parameters string is 12 bytes long.</td>
</tr>
<tr>
<td><strong>snmpInMsgAuthParamStrng</strong></td>
<td>Reserving 12 bytes for the incoming SNMPv3 msg authentication parameters.</td>
</tr>
<tr>
<td><strong>snmpInMsgPrivParamLen</strong></td>
<td>Incoming SNMPv3 msg privacy parameters string is 8 bytes long.</td>
</tr>
<tr>
<td><strong>snmpInMsgPrvParamStrng</strong></td>
<td>Reserving 8 bytes for the incoming SNMPv3 msg privacy parameters.</td>
</tr>
<tr>
<td><strong>snmpMsgBufSeekPos</strong></td>
<td>Offset to read PDU data bytes for processing from dynamic memory stub.</td>
</tr>
<tr>
<td><strong>snmpOutMsgAuthParamLen</strong></td>
<td>Outgoing SNMPv3 msg authentication parameters string is 12 bytes long.</td>
</tr>
<tr>
<td><strong>snmpOutMsgAuthParamStrng</strong></td>
<td>Reserving 12 bytes for the outgoing SNMPv3 msg authentication parameters.</td>
</tr>
<tr>
<td><strong>snmpOutMsgPrivParamLen</strong></td>
<td>Outgoing SNMPv3 msg privacy parameters string is 8 bytes long.</td>
</tr>
<tr>
<td><strong>snmpOutMsgPrvParamStrng</strong></td>
<td>Reserving 8 bytes for the outgoing SNMPv3 msg privacy parameters.</td>
</tr>
<tr>
<td><strong>snmpResponseSecurityFlag</strong></td>
<td>Type of Security for outgoing message in response to the incoming message.</td>
</tr>
<tr>
<td><strong>snmpSecurityLevel</strong></td>
<td>Type of security. noAuthNoPriv(0), AuthNoPriv(1), AuthPriv(3)</td>
</tr>
<tr>
<td><strong>snmpTrapTimer</strong></td>
<td>This is variable snmpTrapTimer.</td>
</tr>
<tr>
<td><strong>snmpV3AesDecryptInitVector</strong></td>
<td>128 Bit</td>
</tr>
<tr>
<td><strong>snmpV3AesEncryptInitVector</strong></td>
<td>128 Bit</td>
</tr>
<tr>
<td><strong>snmpV3UserDataBase</strong></td>
<td>This is variable snmpV3UserDataBase.</td>
</tr>
</tbody>
</table>
authKey_iPad Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8 authKey_iPad[64];</td>
</tr>
</tbody>
</table>

Description

This is variable authKey_iPad.
authKey_oPad Variable

C
UINT8 authKey_oPad[64];

Description

This is variable authKey_oPad.
authoritativeSnmpEngineBoots Variable

C

DWORD_VAL authoritativeSnmpEngineBoots;

Description

The number of times that the authoritative SNMP engine has (re-)initialized itself since its snmpEngineID was last configured.

Stack API > SNMP > Variables > authoritativeSnmpEngineBoots Variable
authoritativeSnmpEngineTime Variable

C

DWORD_VAL authoritativeSnmpEngineTime;

Description

The number of seconds since the value of the authoritativeSnmpEngineBoots object last changed
cipher_text Variable

```
C
UINT8 cipher_text[16];
```

Description

This is variable cipher_text.
deciphered_text Variable

| C |
|UINT8 deciphered_text[16];|

Description

This is variable deciphered_text.
### Description

This variable is used for `gext` next request for zero instance

```c
BOOL getZeroInstance = FALSE;
```
gSnmpV3InPduWholeMsgBuf Variable

Description

Dynamic memory stub and PDU details for Incoming stored PDU

Stack API > SNMP > Variables > gSnmpV3InPduWholeMsgBuf Variable
gSnmpV3OUTPduWholeMsgBuf Variable

C

SNMPV3 RESPONSE WHOLEMSG gSnmpV3OUTPduWholeMsgBuf = {NULL, NULL, {}, {}}

Description

Dynamic memory stub details and constructed outgoing stored PDU details
gSNMPv3PduHeaderBuf Variable

C

SNMPV3MSGDATA gSNMPv3PduHeaderBuf = {NULL, 0, 0, 0};

Description

Response PDU construction offset details

Stack API > SNMP > Variables > gSNMPv3PduHeaderBuf Variable
gSNMPv3ScopedPduDataPos Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT16  gSNMPv3ScopedPduDataPos = 0;</td>
</tr>
</tbody>
</table>

### Description

Offset to read scoped PDU data bytes for processing from dynamic memory stub
gSNMPv3ScopedPduRequestBuf Variable

Description

Stored request scoped pdu details
gSNMPv3ScopedPduResponseBuf Variable

```c
SNMPV3MSGDATA gSNMPv3ScopedPduResponseBuf = {NULL, 0, 0, 0};
```

Description

Processed response scoped pdu details
gSnmpv3TrapConfigData Variable

Description

SNMPv3 global configuration database to be used for trap notification
gSNMPv3TrapMsgHeaderBuf Variable

```
C
SNMPV3MSGDATA gSNMPv3TrapMsgHeaderBuf;
```

Description

TRAP message PDU header construction offset details
gSnmpV3TrapOUTPduWholeMsgBuf Variable

```
SNMPV3_RESPONSE_WHOLEMSG gSnmpV3TrapOUTPduWholeMsgBuf = {NULL,NULL,{0},{0},0,0,0,0};
```

Description

Dynamic memory stub details and constructed trap PDU details
gSNMPv3TrapScopedPduResponseBuf Variable

C

SNMPV3MSGDATA gSNMPv3TrapScopedPduResponseBuf = {NULL, 0, 0, 0};

Description

TRAP scoped PDU construction offset details

Stack API > SNMP > Variables > gSNMPv3TrapScopedPduResponseBuf Variable
gSNMPV3TrapSecurityLevel Variable

```
C

UINT8 gSNMPV3TrapSecurityLevel = NO_REPORT_NO_PRIVACY_NO_AUTH;
```

Description

This is variable gSNMPV3TrapSecurityLevel.
gSnmpv3UserDBIndex Variable

C

WORD gSnmpv3UserDBIndex;

Description

Index to the particular reference configured in User security model database **snmpV3UserDataBase**.
gUsmStatsEngineID Variable

C

WORD_VAL  gUsmStatsEngineID = {0};

Description

Global variable to find out how many times SNMPv3 engine id has been validated

Stack API > SNMP > Variables > gUsmStatsEngineID Variable
hmacAuthKeyBuf Variable

```
C

UINT8 hmacAuthKeyBuf[64];
```

Description

This is variable hmacAuthKeyBuf.
HmacMd5Digest Variable

C

UINT8 HmacMd5Digest[16];

Description

This is variable HmacMd5Digest.
HmacSHADigest Variable

C

UINT8 HmacSHADigest[20];

Description

This is variable HmacSHADigest.
Incoming PDU Variable

Description

Incoming PDU details
incomingSnmpPDUsMsgID Variable

C

DWORD_VAL incomingSnmpPDUsMsgID;

Description

Retrieved Incoming Msg ID value from PDU
### ivEncrptKeyOut Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8 ivEncrptKeyOut[16];</td>
</tr>
</tbody>
</table>

#### Description

This is variable ivEncrptKeyOut.
md5LocalizedAuthKey Variable

C

UINT8 md5LocalizedAuthKey[16];

Description

This is variable md5LocalizedAuthKey.

Stack API > SNMP > Variables > md5LocalizedAuthKey Variable
msgSecrtyParamLenOffset Variable

| C
| WORD msgSecrtyParamLenOffset;

Description

This is variable msgSecrtyParamLenOffset.
securityPrimitivesOfIncomingPdu Variable

Description

Incoming PDU Security primitive details.

Stack API > SNMP > Variables > securityPrimitivesOfIncomingPdu Variable
session_key Variable

C
AES_SESSION_KEY_128_BIT session_key;

Description

This is variable session_key.
sha1LocalizedAuthKey Variable

C

```c
UINT8 sha1LocalizedAuthKey[20];
```

Description

This is variable sha1LocalizedAuthKey.
snmpEngineBoots Variable

C

UINT32 snmpEngineBoots = 0;

Description

The number of times that the SNMP engine has (re-)initialized itself since snmpEngineID was last configured.

Stack API > SNMP > Variables > snmpEngineBoots Variable
### snmpEngineID Variable

```c
UINT8 snmpEngineID[32+1];
```

**Description**

Reserving 32 bytes for the snmpEngineID as the octet string length can vary from 5 to 32
snmpEngineMaxMessageSize Variable

C

DWORD _VAL  snmpEngineMaxMessageSize;

Description

The maximum message size the SNMP engine can handle.
snmpEngineMsgProcessModel Variable

C

UINT32 snmpEngineMsgProcessModel = 0;

Description

Type of Message processing model used. Value Maximum range (2^31-1), RFC3411
**snmpEngineSecurityModel** Variable

```c
UINT32 snmpEngineSecurityModel = 0;
```

**Description**

Type of security model used. Value Maximum range \(2^{31-1}\), RFC3411.

[Stack API] > [SNMP] > [Variables] > [snmpEngineSecurityModel Variable]
snmpEngineTime Variable

C

DWORD_VAL  snmpEngineTime;

Description

The number of seconds since the value of the snmpEngineBoots object last changed
**snmpEngineTimeOffset Variable**

```c
DWORD snmpEngineTimeOffset = 0;
```

**Description**

Stores the time value in seconds since SNMP Engine reset
snmpEngnIDLength Variable

C

UINT8 snmpEngnIDLength = 0;

Description

Engine ID length of the SNMP Engine

Stack API > SNMP > Variables > snmpEngnIDLength Variable
snmpInMsgAuthParamLen Variable

```
C

UINT8 snmpInMsgAuthParamLen;
```

Description

Incoming SNMPv3 msg authentication parameters string is 12 bytes long.
### snmpInMsgAuthParamStrng Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8 snmpInMsgAuthParamStrng[];</td>
</tr>
</tbody>
</table>

**Description**

Reserving 12 bytes for the incoming SNMPv3 msg authentication parameters.

[Stack API > SNMP > Variables > snmpInMsgAuthParamStrng Variable](#)
snmpInMsgPrivParamLen Variable

C

UINT8 snmpInMsgPrivParamLen = 8;

Description

Incoming SNMPv3 msg privacy parameters string is 8 bytes long.
snmpInMsgPrvParamStrng Variable

C

UINT8 snmpInMsgPrvParamStrng[];

Description

Reserving 8 bytes for the incoming SNMPv3 msg privacy parameters.
C

UINT16 snmpMsgBufSeekPos = 0;

Description

Offset to read PDU data bytes for processing from dynamic memory stub
snmpOutMsgAuthParamLen Variable

C

UINT8 snmpOutMsgAuthParamLen;

Description

Outgoing SNMPv3 msg authentication parameters string is 12 bytes long.
snmpOutMsgAuthParamStrng Variable

C

UINT8 snmpOutMsgAuthParamStrng[];

Description

Reserving 12 bytes for the outgoing SNMPv3 msg authentication parameters.

Stack API > SNMP > Variables > snmpOutMsgAuthParamStrng Variable
snmpOutMsgPrivParamLen Variable

C

UINT8 snmpOutMsgPrivParamLen = 8;

Description

Outgoing SNMPv3 msg privacy parameters string is 8 bytes long.
**snmpOutMsgPrvParamStrng Variable**

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8 snmpOutMsgPrvParamStrng[];</td>
</tr>
</tbody>
</table>

**Description**

Reserving 8 bytes for the outgoing SNMPv3 msg privacy parameters.
snmpResponseSecurityFlag Variable

C

UINT8 snmpResponseSecurityFlag = 0;

Description

Type of Security for outgoing message in response to the incoming message.
**snmpSecurityLevel Variable**

```c
UINT8 snmpSecurityLevel = 0;
```

**Description**

Type of security. noAuthNoPriv(0), AuthNoPriv(1), AuthPriv(3)
snmpTrapTimer Variable

C

DWORD snmpTrapTimer = 0;

Description

This is variable snmpTrapTimer.

Stack API > SNMP > Variables > snmpTrapTimer Variable
snmpV3AesDecryptInitVector Variable

C

UINT8 snmpV3AesDecryptInitVector[16+1];

Description

128 Bit
snmpV3AesEncryptInitVector Variable

C

UINT8 snmpV3AesEncryptInitVector[16+1];

Description

128 Bit

Stack API > SNMP > Variables > snmpV3AesEncryptInitVector Variable
snmpV3UserDataBase Variable

This is variable snmpV3UserDataBase.
## Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH_LOCALIZED_PASSWORD_KEY_LEN</td>
<td>SNMPv3 Authentication length size</td>
</tr>
<tr>
<td>INVALID_INDEX</td>
<td>This is macro INVALID_INDEX.</td>
</tr>
<tr>
<td>IS_SNMPV3_AUTH_STRUCTURE</td>
<td>This is macro IS_SNMPV3_AUTH_STRUCTURE.</td>
</tr>
<tr>
<td>MSG_AUTHORITATIVE_HEADER_LEN</td>
<td>Length of SNMPv3 authoritative msg header:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- engine boot time (4 bytes) + engine time (4 bytes) + security name</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- authentication parameters (snmpOutMsgAuthParamLen parameters (snmpOutMsgAuthParamLen) +</td>
</tr>
<tr>
<td></td>
<td>- security primitives of incoming PDU (snmpOutMsgAuthParamLen)</td>
</tr>
<tr>
<td>MSGGLOBAL_HEADER_LEN</td>
<td>Length of the SNMPv3 msg header (x):</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- MSGID: size (type (1 byte) + length of value (1 byte) + 4 bytes value)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- msg_MAXSIZE: type (1 byte) + length of value (1 byte) + 4 bytes value</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- msg_flag: type (1 byte) + length of value (1 byte) + 1 byte value</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- security_model: type (1 byte) + length of value (1 byte) + 1 byte value</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>#define</td>
</tr>
</tbody>
</table>

### AUTH_LOCALIZED_PASSWORD_KEY_LEN

The **AUTH_LOCALIZED_PASSWORD_KEY_LEN** macro is used to specify the authentication length size in SNMPv3. It represents the length of the localized password key.

### INVALID_INDEX

The **INVALID_INDEX** macro is a placeholder for invalid index values. It indicates that a given index is not valid.

### IS_SNMPV3_AUTH_STRUCTURE

The **IS_SNMPV3_AUTH_STRUCTURE** macro is used to check if the structure is for SNMPv3 authentication. It helps in determining if the structure is relevant for SNMPv3 operations.

### MSG_AUTHORITATIVE_HEADER_LEN

The **MSG_AUTHORITATIVE_HEADER_LEN** macro defines the length of the authoritative message header in SNMPv3. This header contains important information such as the engine boot time, engine time, security name, authentication parameters, and security primitives.

### MSGGLOBAL_HEADER_LEN

The **MSGGLOBAL_HEADER_LEN** macro specifies the length of the SNMPv3 message header, which includes various fields such as MSGID, msg_MAXSIZE, msg_flag, and security_model. This macro is crucial for correctly defining the size of the message header in SNMPv3.
<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIV_LOCALIZED_PASSWORD_KEY_LEN</td>
<td>PRIV_LOCALIZED_PASSWORD_KEY_LEN</td>
</tr>
<tr>
<td>REPORT_RESPONSE</td>
<td>REPORT_RESPONSE</td>
</tr>
<tr>
<td>SNMPENGINE_MAX_MSG_SIZE</td>
<td>SNMPENGINE_MAX_MSG_SIZE</td>
</tr>
<tr>
<td>SNMP_MAX_MSG_SIZE</td>
<td>SNMP_MAX_MSG_SIZE</td>
</tr>
<tr>
<td>SNMP_MAX_OID_LEN_MEM_USE</td>
<td>SNMP_MAX_OID_LEN_MEM_USE</td>
</tr>
<tr>
<td>SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE</td>
<td>SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE</td>
</tr>
<tr>
<td>SNMP_V3</td>
<td>SNMP_V3</td>
</tr>
<tr>
<td>SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE</td>
<td>SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE</td>
</tr>
<tr>
<td>SNMPV3_H</td>
<td>SNMPV3_H</td>
</tr>
<tr>
<td>SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE</td>
<td>SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE</td>
</tr>
<tr>
<td>SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE</td>
<td>SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE</td>
</tr>
<tr>
<td>SNMPV3_USM_MAX_USER</td>
<td>SNMPV3_USM_MAX_USER</td>
</tr>
<tr>
<td>SNMPV3MSG_AUTHENTICATION_FAIL</td>
<td>SNMPV3MSG_AUTHENTICATION_FAIL</td>
</tr>
<tr>
<td>SNMPV3MSG_AUTHENTICATION_SUCCESS</td>
<td>SNMPV3MSG_AUTHENTICATION_SUCCESS</td>
</tr>
<tr>
<td>USER_SECURITY_NAME_LEN</td>
<td>USER_SECURITY_NAME_LEN</td>
</tr>
</tbody>
</table>
Module

SNMP

Stack API > SNMP > Macros

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
AUTH_LOCALIZED_PASSWORD_KEY_LEN

Macro

```c
#define AUTH_LOCALIZED_PASSWORD_KEY_LEN (20)
```

Description

SNMPv3 Authentication Localized passwed key length size

Stack API > SNMP > Macros > AUTH_LOCALIZED_PASSWORD_KEY_LEN Macro
INVALID_INDEX Macro

```c
#define INVALID_INDEX 0xFF
```

Description

This is macro INVALID_INDEX.

Stack API > SNMP > Macros > INVALID_INDEX Macro
IS_SNMPV3_AUTH_STRUCTURE Macro

```c
#define IS_SNMPV3_AUTH_STRUCTURE(a) (a==SNMPV3_ENCRYPTION)
```

Description

This is macro IS_SNMPV3_AUTH_STRUCTURE.
#define MSG_AUTHORITATIVE_HEADER_LEN (x) ( x=(2+2 \  
+1+1+snmpEngnIDLength \  
+1+1+4 \  
+1+1+4 \  
+1+1+securityPrimitivesOfIncoming \  
+1+1+snmpOutMsgAuthParamLen \  
+1+1+snmpOutMsgPrivParamLen) \  
)

Description

Length of SNMPv3 authoritative msg header length = Header length (2 + 2 bytes) + engineID (snmpEngnIDLength bytes)

- engine boot (4 bytes)+ engine time (4 bytes)
- +security name (securityPrimitivesOfIncomingPdu value)
- +authentication parameters (snmpOutMsgAuthParamLen value)
- +privacy parameters (snmpOutMsgAuthParamLen value)
MSGGLOBAL_HEADER_LEN Macro

```c
#define MSGGLOBAL_HEADER_LEN(x) ( x= (2 \n    +1+1+4 \n    +1+1+4 \n    +1+1+1 \n    +1+1+1)\n    )
```

Description

Length of the SNMPv3 msg header(x) = Header length (2 bytes)

- MSGID size (type(1 byte) + length of value(1 byte)+4 bytes value)
- msgMAXSIZE(type + length of value +4 bytes value)
- msg flag(type + length of value +1 byte value)
- security model type(type + length of value +1 byte value)
# PRIV_LOCALIZED_PASSWORD_KEY_LEN Macro

```c
#define PRIV_LOCALIZED_PASSWORD_KEY_LEN (20)
```

**Description**

#define PRIV_LOCALIZED_PASSWORD_KEY_LEN 16 SNMPv3 Privacy Password key length size
REPORT_RESPONSE Macro

```
#define REPORT_RESPONSE (0xa8)
```

Description

This is macro REPORT_RESPONSE.

Stack API > SNMP > Macros > REPORT_RESPONSE Macro
SNMP_ENGINE_MAX_MSG_SIZE Macro

```
#define SNMP_ENGINE_MAX_MSG_SIZE 1024
```

Description

SNMP_ENGINE_MAX_MSG_SIZE is determined as the minimum of the max msg size values supported among all of the transports available to and supported by the engine.
SNMP_MAX_MSG_SIZE Macro

```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.
SNMP_MAX_OID_LEN_MEM_USE Macro

C
#define SNMP_MAX_OID_LEN_MEM_USE (18)

Description

This macro will be used to avoid SNMP OID memory buffer corruption.
Macro

```c
#define SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE (8)
```

Description

This macro will be used to avoid SNMP OID memory buffer corruption
SNMP_V3 Macro

C
#define SNMP_V3 (3)

Description

This is macro SNMP_V3.
SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE Macro

C
#define SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE (AUTH_LOCALIZED

Description

SNMPv3 authentication localized Key length for memory validation

Stack API > SNMP > Macros > SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE Macro
SNMPV3_H Macro

C

#define SNMPV3_H

Description

This is macro SNMPV3_H.
SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE

Macro

C

#define SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE (PRIV_LOCALIZED

Description

SNMPv3 privacy key length size for memory validation

Stack API > SNMP > Macros > SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE Macro
SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE Macro

```c
#define SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE (USER_SECURITY_NAME_LEN+1)
```

Description

User security name length for memory validation

Stack API > SNMP > Macros > SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE Macro
### SNMPV3_USM_MAX_USER Macro

```
#define SNMPV3_USM_MAX_USER 3 //User Security Model should have at least 1 user. Default is 3. User should change as per the requirement.
```

#### Description

User Security Model should have at least 1 user. Default is 3. User should change as per the requirement.
SNMPV3MSG_AUTHENTICATION_FAIL Macro

```c
#define SNMPV3MSG_AUTHENTICATION_FAIL 0
```

**Description**

This is macro SNMPV3MSG_AUTHENTICATION_FAIL.
SNMPV3MSG_AUTHENTICATION_SUCCESS Macro

```
#define SNMPV3MSG_AUTHENTICATION_SUCCESS 1
```

Description

This is macro SNMPV3MSG_AUTHENTICATION_SUCCESS.
USER_SECURITY_NAME_LEN Macro

```c
#define USER_SECURITY_NAME_LEN (16)
```

Description

SNMPv3 User Security Name length
### Files

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMPv3.c</td>
<td>- Simple Network Management Protocol (SNMP) Version 3 Agent</td>
</tr>
<tr>
<td></td>
<td>- Module for Microchip TCP/IP Stack</td>
</tr>
<tr>
<td></td>
<td>- Provides SNMPv3 API for doing stuff</td>
</tr>
<tr>
<td></td>
<td>- Reference: RFCs 3410, 3411, 3412, 3413, 3414</td>
</tr>
<tr>
<td>SNMPv3.h</td>
<td>- Simple Network Management Protocol (SNMP) Version 3 Agent</td>
</tr>
<tr>
<td></td>
<td>- Module for Microchip TCP/IP Stack</td>
</tr>
<tr>
<td></td>
<td>- Provides SNMPv3 API for doing stuff</td>
</tr>
<tr>
<td></td>
<td>- Reference: RFCs 3410, 3411, 3412, 3413, 3414</td>
</tr>
<tr>
<td>SNMPv3USM.c</td>
<td>- Simple Network Management Protocol (SNMP) Version 3 Agent</td>
</tr>
<tr>
<td></td>
<td>- Module for Microchip TCP/IP Stack</td>
</tr>
<tr>
<td></td>
<td>- Provides SNMPv3 API for doing stuff</td>
</tr>
</tbody>
</table>
- Reference: RFCs 3410, 3411, 3412, 3413, 3414

- Simple Network Management Protocol (SNMP) Version 1 Agent
- Simple Network Management Protocol (SNMP) Version 2 community based Agent
- Module for Microchip TCP/IP Stack
- Provides SNMP API for doing stuff

- Reference: RFC 1157 (for SNMP V1)
- RFC 3416 (for SNMPv2C)

- SNMPDefs for Microchip TCP/IP Stack

Module

SNMP

Stack API > SNMP > Files

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
SNMPv3.c

- Simple Network Management Protocol (SNMP) Version 3 Agent

- Module for Microchip TCP/IP Stack
- Provides SNMPv3 API for doing stuff

- Reference: RFCs 3410, 3411, 3412, 3413, 3414

*******************************************************************

FileName: SNMPv3.c
Dependencies: TCP/IP stack
Processor: PIC32
Compiler: Microchip C32

Software License Agreement

Copyright (C) 2012 Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy, and distribute:
(i) the Software when embedded on a Microchip microcontroller or digital signal controller product ("Device") which is integrated into Licensee's product; or
(ii) ONLY the Software driver source files ENC28J60.c, ENC28J60.h,
    ENCX24J600.c and ENCX24J600.h ported to a non-Microchip
device
- used in conjunction with a Microchip ethernet controller for
- the sole purpose of interfacing with the ethernet controller.

* You should refer to the license agreement accompanying this
  Software for additional information regarding your rights and
  obligations.

* THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
  WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
  LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
  PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
  MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
  CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
  PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
  BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
  THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
  SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
  (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
## Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snmpv3IsValidAuthStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>FindOIDsFromSnmpV3Request</td>
<td>Finds number of varbinds in the varbind list received in a SNMPv3 pdu.</td>
</tr>
<tr>
<td>IsSnmpV3ASNNull</td>
<td>Verifies the value type as ASN_NULL.</td>
</tr>
<tr>
<td>IsSnmpV3ValidOID</td>
<td>Populates OID type, length and oid string from the received pdu.</td>
</tr>
<tr>
<td>IsSNMPv3ValidStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>Snmpv3FreeDynAllocMem</td>
<td>Allocated dynamic memory freeing is done by this routine.</td>
</tr>
<tr>
<td>Snmpv3IsValidAuthStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>Snmpv3IsValidInt</td>
<td>Verifies variable datatype as INT and retrieves its value.</td>
</tr>
<tr>
<td>Snmpv3SetErrorStatus</td>
<td>Set snmpv3 error status in the response pdu.</td>
</tr>
</tbody>
</table>

## Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_INDEX</td>
<td>This is macro INVALID_INDEX.</td>
</tr>
<tr>
<td>MSG_AUTHORITATIVE_HEADER_LEN</td>
<td>Length of SNMPv3 authorative msg header length = Header length ( 2 + 2 bytes) + engineID ( snmpEngnIDLength bytes)</td>
</tr>
<tr>
<td></td>
<td>• engine boot( 4 bytes)+ engine time(4 bytes)</td>
</tr>
<tr>
<td></td>
<td>+security name (securityPrimitivesOfIncomingPdu value)</td>
</tr>
<tr>
<td></td>
<td>+authentication parameters</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gSnmpV3InPduWholeMsgBuf</code></td>
<td>Dynamic memory stub and PDU details for Incoming stored PDU</td>
</tr>
<tr>
<td><code>gSnmpV3OUTPduWholeMsgBuf</code></td>
<td>Dynamic memory stub details and constructed outgoing stored PDU details</td>
</tr>
<tr>
<td><code>gSNMPv3PduHeaderBuf</code></td>
<td>Response PDU construction offset details</td>
</tr>
<tr>
<td><code>gSNMPv3ScopedPduDataPos</code></td>
<td>Offset to read scoped PDU data bytes for processing from dynamic memory stub</td>
</tr>
<tr>
<td><code>gSNMPv3ScopedPduRequestBuf</code></td>
<td>Stored request scoped pdu details</td>
</tr>
</tbody>
</table>

- `MSGGLOBAL_HEADER_LEN`: Length of the SNMPv3 msg header(x) = Header length (2 bytes)
  - MSGID size (type(1 byte) + length of value(1 byte)+4 bytes value)
  - `msgMAXSIZE(type + length of value +4 bytes value)`
  - `msg flag(type + length of value +1 byte value)`
  - `security model type(type + length of value +1 byte value)`

- `SNMP_ENGINE_MAX_MSG_SIZE`: SNMP_ENGINE_MAX_MSG_SIZE is determined as the minimum of the max msg size values supported among all of the transports available to and supported by the engine.
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gSNMPv3ScopedPduResponseBuf</td>
<td>Processed response scoped pdu details</td>
</tr>
<tr>
<td>gSnmpv3TrapConfigData</td>
<td>SNMPv3 global configuration database to be used for trap notification</td>
</tr>
<tr>
<td>gSnmpV3TrapOUTPduWholeMsgBuf</td>
<td>Dynamic memory stub details and constructed trap PDU details</td>
</tr>
<tr>
<td>gSNMPv3TrapScopedPduResponseBuf</td>
<td>TRAP scoped PDU construction offset details</td>
</tr>
<tr>
<td>gSNMPV3TrapSecurityLevel</td>
<td>This is variable gSNMPV3TrapSecurityLevel.</td>
</tr>
<tr>
<td>gUsmStatsEngineID</td>
<td>Global variable to find out how many times SNMPv3 engine id has been validated</td>
</tr>
<tr>
<td>incomingPdu</td>
<td>Incoming PDU details</td>
</tr>
<tr>
<td>incomingSnmpPDUmsgID</td>
<td>Retrieved Incoming Msg ID value from PDU</td>
</tr>
<tr>
<td>msgSecrtyParamLenOffset</td>
<td>This is variable msgSecrtyParamLenOffset.</td>
</tr>
<tr>
<td>securityPrimitivesOfIncomingPdu</td>
<td>Incoming PDU Security primitive details.</td>
</tr>
<tr>
<td>snmpEngineBoots</td>
<td>The number of times that the SNMP engine has (re-)initialized itself since snmpEngineID was last configured.</td>
</tr>
<tr>
<td>snmpEngineID</td>
<td>Reserving 32 bytes for the snmpEngineID as the octet string length can vary form 5 to 32</td>
</tr>
<tr>
<td>snmpEngineMaxMessageSize</td>
<td>The maximum message size the SNMP engine can handle.</td>
</tr>
<tr>
<td>snmpEngineMsgProcessModel</td>
<td>Type of Message processing model used. Value Maximum range (2^31-1), RFC3411</td>
</tr>
<tr>
<td>snmpEngineSecurityModel</td>
<td>Type of security model used. Value Maximum range (2^31-1), RFC3411</td>
</tr>
<tr>
<td>snmpEngineTime</td>
<td>The number of seconds since the value of the snmpEngineBoots object last changed</td>
</tr>
<tr>
<td>snmpEngineTimeOffset</td>
<td>Stores the time value in seconds since SNMP Engine reset</td>
</tr>
<tr>
<td><strong>snmpEngnIDLength</strong></td>
<td>Engine ID length of the SNMP Engine</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td><strong>snmpInMsgPrivParamLen</strong></td>
<td>Incoming SNMPv3 msg privacy parameters string is 8 bytes long.</td>
</tr>
<tr>
<td><strong>snmpMsgBufSeekPos</strong></td>
<td>Offset to read PDU data bytes for processing from dynamic memory stub</td>
</tr>
<tr>
<td><strong>snmpOutMsgPrivParamLen</strong></td>
<td>Outgoing SNMPv3 msg privacy parameters string is 8 bytes long.</td>
</tr>
<tr>
<td><strong>snmpResponseSecurityFlag</strong></td>
<td>Type of Security for outgoing message in response to the incoming message.</td>
</tr>
<tr>
<td><strong>snmpSecurityLevel</strong></td>
<td>Type of security. noAuthNoPriv(0), AuthNoPriv(1), AuthPriv(3)</td>
</tr>
<tr>
<td><strong>SNMPTxOffset</strong></td>
<td>Snmp udp buffer tx offset</td>
</tr>
</tbody>
</table>
SNMPv3.h

- Simple Network Management Protocol (SNMP) Version 3 Agent

- Module for Microchip TCP/IP Stack
- Provides SNMPv3 API for doing stuff

- Reference: RFCs 3410, 3411, 3412, 3413, 3414

*******************************************************************

- FileName: SNMPv3.h
- Dependencies: TCP/IP stack
- Processor: PIC32
- Compiler: Microchip C32

Software License Agreement

- Copyright (C) 2012 Microchip Technology Inc. All rights reserved.

- Microchip licenses to you the right to use, modify, copy, and distribute:
  (i) the Software when embedded on a Microchip microcontroller or digital signal controller product ("Device") which is integrated into Licensee's product; or
  (ii) ONLY the Software driver source files ENC28J60.c, ENC28J60.h, ENCX24J600.c and ENCX24J600.h ported to a non-Microchip
device

- used in conjunction with a Microchip ethernet controller for
- the sole purpose of interfacing with the ethernet controller.

* You should refer to the license agreement accompanying this
  Software for additional information regarding your rights and
  obligations.

* THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
  WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
  LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
  PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
  MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
  CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
  PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
  BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
  THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
  SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
  (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORT_FLAG_AND_SECURITY_LEVEL_FLAGS</td>
<td>This is type REPORT_FLAG_AND_SECURITY_LEVEL_FLAGS.</td>
</tr>
<tr>
<td>SNMP_ENGNID_OCTET_IDENTIFIER_VAL</td>
<td>The fifth octet indicates how the rest (6th and following octets) are formatted. Refer to RFC3411 section5 Page# 41</td>
</tr>
<tr>
<td>SNMPV3_HMAC_HASH_TYPE</td>
<td>Type of hash being calculated.</td>
</tr>
<tr>
<td>SNMPV3_MSG_AUTH_SEC_PARAM_RESULT</td>
<td>This is type SNMPV3_MSG_AUTH_SEC_PARAM_RESULT.</td>
</tr>
<tr>
<td>SNMPV3_MSG_PRIV_SEC_PARAM_RESULT</td>
<td>This is type SNMPV3_MSG_PRIV_SEC_PARAM_RESULT.</td>
</tr>
<tr>
<td>SNMPV3_PRIV_PROT_TYPE</td>
<td>This is type SNMPV3_PRIV_PROT_TYPE.</td>
</tr>
<tr>
<td>STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL</td>
<td>This is type STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL.</td>
</tr>
<tr>
<td>STD_BASED_SNMP_SECURITY_MODEL</td>
<td>Snmp Message Processing Model</td>
</tr>
<tr>
<td>STD_BASED_SNMPV3_SECURITY_LEVEL</td>
<td>This is type STD_BASED_SNMPV3_SECURITY_LEVEL.</td>
</tr>
<tr>
<td>USM_SECURITY_LEVEL</td>
<td>This is type USM_SECURITY_LEVEL.</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IsSNMPv3ValidStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>_Snmpv3IsValidInt</td>
<td>Verifies variable datatype as INT and retrieves its value.</td>
</tr>
<tr>
<td>Snmpv3AESEncryptResponseScopedPdu</td>
<td>outGoing SNMPv3 scoped PDU Encryption using AES encryption protocol.</td>
</tr>
<tr>
<td>Snmpv3AuthenticateRxedPduForDataIntegrity</td>
<td>Authenticate an incoming SNMPV3 USM PDU using MD5 or SHA</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Snmpv3AuthenticateTxPduForDataIntegrity</td>
<td>Authenticate to an outgoing SNMPv3 USM PDU using MD5 or SHA.</td>
</tr>
<tr>
<td>Snmpv3AuthKeyZeroing2HmacBufLen64</td>
<td>Pad zero to the authentication key localized buffer.</td>
</tr>
<tr>
<td>Snmpv3CmprTrapSecNameAndSecLvIWithUSMDb</td>
<td>Routine to find the index of the user name in the user data base table.</td>
</tr>
<tr>
<td>Snmpv3ComputeHMACIpadOpadForAuthLoclzedKey</td>
<td>Compute HMAC inner and outer pad for authorization localized key.</td>
</tr>
<tr>
<td>Snmpv3ComputeHmacMD5Digest</td>
<td>Compute HMAC - MD5 authentication code.</td>
</tr>
<tr>
<td>Snmpv3ComputeHmacShaDigest</td>
<td>Compute HMAC - SHA authentication code.</td>
</tr>
<tr>
<td>Snmpv3ComputeMd5HmacCode</td>
<td>Compute HMAC - MD5 authentication code.</td>
</tr>
<tr>
<td>Snmpv3ComputeShaHmacCode</td>
<td>Compute HMAC - SHA authentication code.</td>
</tr>
<tr>
<td>Snmpv3GetSecurityLevel</td>
<td>Get Security level from authentication and Privacy type.</td>
</tr>
<tr>
<td>Snmpv3GetTrapSecurityLevel</td>
<td>Routine to find the report, auth and privacy flags settings in the TRAP.</td>
</tr>
<tr>
<td>Snmpv3Init</td>
<td>SNMPv3 initialization.</td>
</tr>
<tr>
<td>Snmpv3InitializeUserDatabase</td>
<td>Initialize default SNMPv3 global user database.</td>
</tr>
<tr>
<td>Snmpv3Pswd2LocalizedAuthKeyMD5Hashing</td>
<td>Convert MD5 Auth password to the localized Key using SNMPEngineID.</td>
</tr>
<tr>
<td>Snmpv3Pswd2LocalizedAuthKeySHAHashing</td>
<td>Convert SHA Auth password to the localized Key using SNMPEngineID.</td>
</tr>
</tbody>
</table>
**Snmpv3USMOutMsgPrivParam**
SNMP USM out message uses Privacy protocol (RFC 3826)

**Snmpv3UsmSnmpEngnAuthPrivPswdLocalization**
Convert Auth and Priv password to the localized Key using SNMPEngineID.

## Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AUTH_LOCALIZED_PASSWORD_KEY_LEN</strong></td>
<td>SNMPv3 Authentication length size</td>
</tr>
<tr>
<td><strong>PRIV_LOCALIZED_PASSWORD_KEY_LEN</strong></td>
<td>#define PRIV_LOCALIZED_PASSWORD_KEY_LEN</td>
</tr>
<tr>
<td><strong>SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE</strong></td>
<td>SNMPv3 authentication localized Key length for memory validation</td>
</tr>
<tr>
<td><strong>SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE</strong></td>
<td>SNMPv3 privacy key length size for memory validation</td>
</tr>
<tr>
<td><strong>SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE</strong></td>
<td>User security name length size for memory validation</td>
</tr>
<tr>
<td><strong>SNMPV3_USM_MAX_USER</strong></td>
<td>User Security Model should have at least 1 user. Default is 3. User should change as per requirement.</td>
</tr>
<tr>
<td><strong>SNMPV3MSG_AUTHENTICATION_FAIL</strong></td>
<td>This is macro SNMPV3MSG_AUTHENTICATION_FAIL</td>
</tr>
<tr>
<td><strong>SNMPV3MSG_AUTHENTICATION_SUCCESS</strong></td>
<td>This is macro SNMPV3MSG_AUTHENTICATION_SUCCESS</td>
</tr>
<tr>
<td><strong>USER_SECURITY_NAME_LEN</strong></td>
<td>SNMPv3 User Security Name length</td>
</tr>
</tbody>
</table>

## Structures
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AccessCtrlSubSysIsAccessAllowed</strong></td>
<td>Applications are the typical clients of the service(s) of the Access Control Subsystem. The following primitive is provided by the Access Control Subsystem to check if access is allowed: statusInformation = -- success or errorIndication</td>
</tr>
<tr>
<td><strong>dispatcherProcessPdu</strong></td>
<td>Process Incoming Request or Notification PDU Dispatcher provides the following primitive to pass an incoming snmp pdu to an application.</td>
</tr>
<tr>
<td><strong>dispatcherStatusInfo</strong></td>
<td>Generate Outgoing Request or Notification statusInformation = -- sendPduHandle if success -- errorIndication if failure</td>
</tr>
<tr>
<td><strong>dispatcherReturnResponsePdu</strong></td>
<td>Generate Outgoing Response The PDU Dispatcher provides the following primitive for an application to return an SNMP Response PDU to the PDU Dispatcher: result = SUCCESS or FAILURE</td>
</tr>
<tr>
<td><strong>MsgProcModPrepareDataElements</strong></td>
<td>Prepare Data Elements from an Incoming SNMP Message The Message Processing Subsystem provides this service primitive for preparing the abstract data elements from an incoming SNMP message: result = -- SUCCESS or errorIndication</td>
</tr>
<tr>
<td><strong>MsgProcModPrepareOutgoingMessage</strong></td>
<td>Prepare Outgoing SNMP Request or Notification Message The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Request or Notification Message</td>
</tr>
<tr>
<td><strong>MsgProcModPrepareResponseMessage</strong></td>
<td>Prepare an Outgoing SNMP Response Message The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Response Message: result = -- SUCCESS or FAILURE</td>
</tr>
<tr>
<td><strong>Process Incoming Response PDU</strong></td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>processResponsePdu</td>
<td>following primitive to pass an incoming SNMP Response PDU to an application:</td>
</tr>
<tr>
<td>registerContextEngineID</td>
<td>success or errorIndication</td>
</tr>
<tr>
<td>SecuritySysGenerateRequestMsg</td>
<td>This is record SecuritySysGenerateRequestMsg.</td>
</tr>
<tr>
<td>SecuritySysGenerateResponseMsg</td>
<td>Generate a Response Message The Security Subsystem provides the following primitive to generate a Response message:</td>
</tr>
<tr>
<td>StateRelease</td>
<td>Release State Reference Information All Subsystems which pass stateReference information also provide a primitive to release the memory that holds the referenced state information</td>
</tr>
<tr>
<td>unregisterContextEngineID</td>
<td>This is record unregisterContextEngineID.</td>
</tr>
<tr>
<td>SecuritySysProcessIncomingMsg</td>
<td>This is type SecuritySysProcessIncomingMsg.</td>
</tr>
<tr>
<td>SNMPV3_REQUEST_WHOLEMSG</td>
<td>This is type SNMPV3_REQUEST_WHOLEMSG.</td>
</tr>
<tr>
<td>SNMPV3_RESPONSE_WHOLEMSG</td>
<td>This is type SNMPV3_RESPONSE_WHOLEMSG.</td>
</tr>
<tr>
<td>snmpV3EngnUserDataBase</td>
<td>This is type snmpV3EngnUserDataBase.</td>
</tr>
<tr>
<td>snmpV3TrapConfigDataBase</td>
<td>snmpV3 target configuration with respect to trap.</td>
</tr>
<tr>
<td>statusInformation</td>
<td>success or errorIndication</td>
</tr>
</tbody>
</table>
SNMPv3USM.c

- Simple Network Management Protocol (SNMP) Version 3 Agent

- Module for Microchip TCP/IP Stack
- Provides SNMPv3 API for doing stuff

- Reference: RFCs 3410, 3411, 3412, 3413, 3414

*******************************************************************

FileName: SNMPv3USM.c
Dependencies: TCP/IP stack
Processor: PIC32
Compiler: Microchip C32

Software License Agreement

Copyright (C) 2012 Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy, and distribute:
(i) the Software when embedded on a Microchip microcontroller or
digital signal controller product ("Device") which is
integrated into Licensee's product; or
(ii) ONLY the Software driver source files ENC28J60.c,
ENC28J60.h,
ENCX24J600.c and ENCX24J600.h ported to a non-Microchip
device
- used in conjunction with a Microchip ethernet controller for
- the sole purpose of interfacing with the ethernet controller.

* You should refer to the license agreement accompanying this
  Software for additional information regarding your rights and
  obligations.

* THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
  WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
  LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
  PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
  MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
  CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
  PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
  BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
  THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
  SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
  (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authKey_iPad</td>
<td>This is variable authKey_iPad.</td>
</tr>
<tr>
<td>authKey_oPad</td>
<td>This is variable authKey_oPad.</td>
</tr>
<tr>
<td>authoritativeSnmpEngineBoots</td>
<td>The number of times that the authoritative SNMP engine has (re-)initialized itself since its snmpEngineID was last configured.</td>
</tr>
<tr>
<td>authoritativeSnmpEngineTime</td>
<td>The number of seconds since the value of the authoritativeSnmpEngineBoots object last changed</td>
</tr>
<tr>
<td>cipher_text</td>
<td>This is variable cipher_text.</td>
</tr>
<tr>
<td>deciphered_text</td>
<td>This is variable deciphered_text.</td>
</tr>
<tr>
<td>gSNMPv3TrapMsgHeaderBuf</td>
<td>TRAP message PDU header construction offset details</td>
</tr>
<tr>
<td>gSnmpv3UserDBIndex</td>
<td>Index to the particular reference configured in User security model data base snmpv3UserDataBase.</td>
</tr>
<tr>
<td>hmacAuthKeyBuf</td>
<td>This is variable hmacAuthKeyBuf.</td>
</tr>
<tr>
<td>HmacMd5Digest</td>
<td>This is variable HmacMd5Digest.</td>
</tr>
<tr>
<td>HmacSHADigest</td>
<td>This is variable HmacSHADigest.</td>
</tr>
<tr>
<td>ivEncrptKeyOut</td>
<td>This is variable ivEncrptKeyOut.</td>
</tr>
<tr>
<td>md5LocalizedAuthKey</td>
<td>This is variable md5LocalizedAuthKey.</td>
</tr>
<tr>
<td>session_key</td>
<td>This is variable session_key.</td>
</tr>
<tr>
<td>sha1LocalizedAuthKey</td>
<td>This is variable sha1LocalizedAuthKey.</td>
</tr>
<tr>
<td>snmpInMsgAuthParamLen</td>
<td>Incoming SNMPv3 msg authentication parameters string is 12 bytes long.</td>
</tr>
<tr>
<td>snmpInMsgAuthParamStrng</td>
<td>Reserving 12 bytes for the incoming SNMPv3 msg authentication parameters.</td>
</tr>
<tr>
<td>Identifier</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>snmpInMsgPrvParamStrng</td>
<td>Reserving 8 bytes for the incoming SNMPv3 msg privacy parameters.</td>
</tr>
<tr>
<td>snmpOutMsgAuthParamLen</td>
<td>Outgoing SNMPv3 msg authentication parameters string is 12 bytes long.</td>
</tr>
<tr>
<td>snmpOutMsgAuthParamStrng</td>
<td>Reserving 12 bytes for the outgoing SNMPv3 msg authentication parameters.</td>
</tr>
<tr>
<td>snmpOutMsgPrvParamStrng</td>
<td>Reserving 8 bytes for the outgoing SNMPv3 msg privacy parameters.</td>
</tr>
<tr>
<td>snmpV3AesDecryptInitVector</td>
<td>128 Bit</td>
</tr>
<tr>
<td>snmpV3AesEncryptInitVector</td>
<td>128 Bit</td>
</tr>
<tr>
<td>snmpV3UserDataBase</td>
<td>This is variable snmpV3UserDataBase.</td>
</tr>
</tbody>
</table>

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
SNMP.c

- Simple Network Management Protocol (SNMP) Version 1 Agent
- Simple Network Management Protocol (SNMP) Version 2 community based Agent
- Module for Microchip TCP/IP Stack
- -Provides SNMP API for doing stuff

* -Reference: RFC 1157 (for SNMP V1)
- RFC 3416 (for SNMPv2C)

*******************************************************************

- FileName: SNMP.c
- Dependencies: UDP, ARP
- Processor: PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32
- Compiler: Microchip C32 v1.05 or higher
- Microchip C30 v3.12 or higher
- Microchip C18 v3.30 or higher
- HI-TECH PICC-18 PRO 9.63PL2 or higher
- Company: Microchip Technology, Inc.

* - Software License Agreement

* - Copyright (C) 2002-2009 Microchip Technology Inc. All rights reserved.

* - Microchip licenses to you the right to use, modify, copy, and distribute:
• (i) the Software when embedded on a Microchip microcontroller or
digital signal controller product ("Device") which is
integrated into Licensee's product; or
• (ii) ONLY the Software driver source files ENC28J60.c,
ENC28J60.h,
ENCX24J600.c and ENCX24J600.h ported to a non-Microchip
device
used in conjunction with a Microchip ethernet controller for
the sole purpose of interfacing with the ethernet controller.

* You should refer to the license agreement accompanying this
Software for additional information regarding your rights and
obligations.

* THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS"
WITHOUT
WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED,
INCLUDING WITHOUT
LIMITATION, ANY WARRANTY OF MERCHANTABILITY,
FITNESS FOR A
PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN
NO EVENT SHALL
MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL,
INDIRECT OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
COST OF
PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR
SERVICES, ANY CLAIMS
BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY
DEFENSE
THEREOF), ANY CLAIMS FOR INDEMNITY OR
CONTRIBUTION, OR OTHER

- SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
- (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.

* 

- Author Date Comment

*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

- Nilesh Rajbharti 1/9/03 Original (Rev 1.0)
- Dan Cohen 12/11/03 Removed trap support by #define if not required to lower code space requirements
- Amit Shirbhate 09/24/08 SNMPv2c Support, comments and function headers added.
- Hrisikesh Sahu 04/15/10 SNMPv2 Trap Format Support.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindOIDsInRequest</td>
<td>Finds number of varbinds in the varbind list received in a pdu.</td>
</tr>
<tr>
<td>IsASNNull</td>
<td>Verifies the value type as ASN_NULL.</td>
</tr>
<tr>
<td>IsValidCommunity</td>
<td>Verifies for the community string datatype and the max community name and length, this agent can process.</td>
</tr>
<tr>
<td>IsValidOID</td>
<td>Populates OID type, length and oid string from the received pdu.</td>
</tr>
<tr>
<td>IsValidPDU</td>
<td>Verifies for the snmp request type.</td>
</tr>
<tr>
<td>ProcessGetSetHeader</td>
<td>Validates the received udp packet Get/Set request header.</td>
</tr>
<tr>
<td></td>
<td>Validates the received udp packet Snmp</td>
</tr>
</tbody>
</table>
ProcessHeader

header.

ProcessVariables

This routine processes the snmp request and pararellly creates the response pdu.

ReadMIBRecord

Get OID string from MPFS using the node address.

SNMPCheckIfPvtMibObjRequested

To find whether requested OID is only for private access.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appendZeroToOID</td>
<td>global flag to modify OID by appending zero</td>
</tr>
<tr>
<td>dataTypeTable</td>
<td>ASN format datatype for snmp v1 and v2c</td>
</tr>
<tr>
<td>getZeroInstance</td>
<td>This variable is used for gext next request for zero instance</td>
</tr>
<tr>
<td>hMPFS</td>
<td>MPFS file handler</td>
</tr>
<tr>
<td>SNMPAgentSocket</td>
<td>Snmp udp socket</td>
</tr>
<tr>
<td>SNMPNotifyInfo</td>
<td>notify info for trap</td>
</tr>
<tr>
<td>snmpReqVarErrStatus</td>
<td>vars from req list processing err status</td>
</tr>
<tr>
<td>SNMPRxOffset</td>
<td>Snmp udp buffer rx offset</td>
</tr>
<tr>
<td>SNMPStatus</td>
<td>MIB file access status</td>
</tr>
<tr>
<td>snmpTrapTimer</td>
<td>This is variable snmpTrapTimer.</td>
</tr>
</tbody>
</table>

Stack API > SNMP > Files > SNMP.c

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
SNMP.h

- SNMP Defs for Microchip TCP/IP Stack

* *

*******************************************************************

- FileName: SNMP.h
- Dependencies: StackTsk.h, UDP.h
- Processor: PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32
- Compiler: Microchip C32 v1.05 or higher
- Microchip C30 v3.12 or higher
- Microchip C18 v3.30 or higher
- HI-TECH PICC-18 PRO 9.63PL2 or higher
- Company: Microchip Technology, Inc.

* *

- Software License Agreement

* *

- Copyright (C) 2002-2009 Microchip Technology Inc. All rights reserved.

* *

- Microchip licenses to you the right to use, modify, copy, and distribute:
- (i) the Software when embedded on a Microchip microcontroller or digital signal controller product ("Device") which is integrated into Licensee's product; or
- (ii) ONLY the Software driver source files ENC28J60.c, ENC28J60.h,
• ENCX24J600.c and ENCX24J600.h ported to a non-Microchip device
• used in conjunction with a Microchip ethernet controller for
• the sole purpose of interfacing with the ethernet controller.

* You should refer to the license agreement accompanying this
• Software for additional information regarding your rights and
• obligations.

* THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
• WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
• LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
• PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
• MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
• CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
• PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
• BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
• THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
• SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
• (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
V5.36 ---- STACK_USE_MPFS has been removed

## Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMUNITY_TYPE</td>
<td>This is type COMMUNITY_TYPE.</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td></td>
</tr>
<tr>
<td>GENERIC_TRAP_NOTIFICATION_TYPE</td>
<td>This is type GENERIC_TRAP_NOTIFICATION_TYPE.</td>
</tr>
<tr>
<td>INOUT_SNMP_PDU</td>
<td>This is type INOUT_SNMP_PDU.</td>
</tr>
<tr>
<td>SNMP_ACTION</td>
<td>This is the list of SNMP action a remote NMS can perform. This information is passed to application via callback SNMPValidateCommunity should validate the action for given community string.</td>
</tr>
<tr>
<td>SNMP_ERR_STATUS</td>
<td></td>
</tr>
<tr>
<td>VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE</td>
<td>This is type VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE.</td>
</tr>
</tbody>
</table>

## Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SNMPDuplexInit</td>
<td>Prepare for full duplex transfer.</td>
</tr>
<tr>
<td>_SNMPGet</td>
<td>Read byte from snmp udp socket rx buffer.</td>
</tr>
<tr>
<td>_SNMPPut</td>
<td>Copy byte to tx buffer.</td>
</tr>
<tr>
<td>GetDataTypeInfo</td>
<td>Get ASN data type info.</td>
</tr>
<tr>
<td>GetNextLeaf</td>
<td>Searches for the next leaf node in the MIP tree.</td>
</tr>
<tr>
<td>GetOIDStringByAddr</td>
<td>Get OID string from MPFS using the</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetOIDStringByID</td>
<td>Get complete notification variable OID string from MPFS using var id.</td>
</tr>
<tr>
<td>getSnmpV2GenTrapOid</td>
<td>Resolves generic trap code to generic trap OID.</td>
</tr>
<tr>
<td>IsValidInt</td>
<td>Verifies variable datatype as INT and retrieves its value.</td>
</tr>
<tr>
<td>IsValidLength</td>
<td>Retrieves the packet length and actual pdu length.</td>
</tr>
<tr>
<td>IsValidStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>OIDLookup</td>
<td>To search and validate whether the requested OID is in the MIB database.</td>
</tr>
<tr>
<td>ProcessGetBulkVar</td>
<td>This routine process the SNMPv2c Get Bulk Request.</td>
</tr>
<tr>
<td>ProcessGetNextVar</td>
<td>Retrieves next node from the MIB database.</td>
</tr>
<tr>
<td>ProcessGetVar</td>
<td>Processes snmp Get request pdu.</td>
</tr>
<tr>
<td>ProcessSetVar</td>
<td>Processes snmp Set request pdu.</td>
</tr>
<tr>
<td>ProcessSnmpv3MsgData</td>
<td>This routine processes the snmpv3 request and parallely creates the response pdu.</td>
</tr>
<tr>
<td>SetErrorStatus</td>
<td>Set snmp error status in the response pdu.</td>
</tr>
<tr>
<td>SNMPGetExactIndex</td>
<td>To search for exact index node in case of a Sequence variable.</td>
</tr>
<tr>
<td>SNMPGetNextIndex</td>
<td>To search for next index node in case of a Sequence variable.</td>
</tr>
<tr>
<td>SNMPGetTrapTime</td>
<td>Returns trap resolve get time.</td>
</tr>
<tr>
<td>SNMPGetVar</td>
<td>Used to Get/collection OID variable information.</td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>SNMPIdRecrdValidation</strong></td>
<td>Used to Restrict the access dynamic and non dynamic OID string for a particular SNMP Version.</td>
</tr>
<tr>
<td><strong>SNMPInit</strong></td>
<td>Initialize SNMP module internals.</td>
</tr>
<tr>
<td><strong>SNMPIsNotifyReady</strong></td>
<td>Resolves given remoteHost IP address into MAC address.</td>
</tr>
<tr>
<td><strong>SNMPIsValidSetLen</strong></td>
<td>Validates the set variable data length to data type.</td>
</tr>
<tr>
<td><strong>SNMPNotify</strong></td>
<td>Creates and Sends TRAP pdu.</td>
</tr>
<tr>
<td><strong>SNMPNotifyPrepare</strong></td>
<td>Collects trap notification info and send ARP to remote host.</td>
</tr>
<tr>
<td><strong>SNMPSendTrap</strong></td>
<td>Prepare, validate remote node which will receive trap and send trap pdu.</td>
</tr>
<tr>
<td><strong>SNMPSetVar</strong></td>
<td>This routine Set the mib variable with the requested value.</td>
</tr>
<tr>
<td><strong>SNMPTask</strong></td>
<td>Polls for every snmp pdu received.</td>
</tr>
<tr>
<td><strong>Snmpv3AESDecryptRxedScopedPdu</strong></td>
<td>Incoming SNMPv3 scoped PDU decryption using AES decryption protocol.</td>
</tr>
<tr>
<td><strong>Snmpv3BufferPut</strong></td>
<td>Copies BYTE data to dynamically allocated memory buffer.</td>
</tr>
<tr>
<td><strong>Snmpv3FormulateEngineID</strong></td>
<td>Formulates the <code>snmpEngineID</code> for the SNMPV3 engine.</td>
</tr>
<tr>
<td><strong>Snmpv3GetAuthEngineTime</strong></td>
<td>Updates the snmp engine time variable <code>snmpEngineTime</code> for the SNMPV3 engine.</td>
</tr>
<tr>
<td><strong>Snmpv3GetBufferData</strong></td>
<td>Reads BYTE data from dynamically allocated memory buffer.</td>
</tr>
<tr>
<td><strong>Snmpv3MsgProcessingModelProcessPDU</strong></td>
<td>This routine collects or populates the message processing model information from the received SNMPv3 request PDU or to the response PDU respectively.</td>
</tr>
<tr>
<td></td>
<td>Creates and Sends SNMPv3 TRAP</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Snmpv3Notify</td>
<td>pdu.</td>
</tr>
<tr>
<td>Snmpv3ScopedPduProcessing</td>
<td>This routine collects the scoped pdu header information from the received SNMPv3 request PDU or populates to the response PDU respectively.</td>
</tr>
<tr>
<td>Snmpv3TrapScopedpdu</td>
<td>TRAP PDU scoped pdu header construction.</td>
</tr>
<tr>
<td>Snmpv3UserSecurityModelProcessPDU</td>
<td>This routine collects or populates the security model parameters information from the received SNMPv3 request PDU or to the response PDU respectively.</td>
</tr>
<tr>
<td>Snmpv3UsmAesEncryptDecryptInitVector</td>
<td>AES Encryption and decryption init vector.(RFC 3826 )</td>
</tr>
<tr>
<td>Snmpv3UsmOutMsgAuthenticationParam</td>
<td>Both MD5 and SHA1 is used for the outgoing message authentication.</td>
</tr>
<tr>
<td>Snmpv3ValidateEngineId</td>
<td>Validate engine ID.</td>
</tr>
<tr>
<td>Snmpv3ValidateSecNameAndSecLvl</td>
<td>Validate security name with Security level.</td>
</tr>
<tr>
<td>Snmpv3ValidateSecurityName</td>
<td>Validate SNMPV3 user name or security name.</td>
</tr>
<tr>
<td>SNMPValidateCommunity</td>
<td>Validates community name for access control.</td>
</tr>
</tbody>
</table>

### Macros

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>_SNMPGetTxOffset</td>
<td>This is macro _SNMPGetTxOffset.</td>
</tr>
<tr>
<td>_SNMPSetTxOffset</td>
<td>This is macro _SNMPSetTxOffset.</td>
</tr>
<tr>
<td>AGENT_NOTIFY_PORT</td>
<td>This is macro AGENT_NOTIFY_PORT.</td>
</tr>
<tr>
<td>ASN_INT</td>
<td>This is macro ASN_INT.</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ASN_NULL</td>
<td>This is macro ASN_NULL.</td>
</tr>
<tr>
<td>ASN_OID</td>
<td>This is macro ASN_OID.</td>
</tr>
<tr>
<td>DATA_TYPE_TABLE_SIZE</td>
<td>This is macro DATA_TYPE_TABLE_SIZE.</td>
</tr>
<tr>
<td>GET_BULK_REQUEST</td>
<td>This is macro GET_BULK_REQUEST.</td>
</tr>
<tr>
<td>GET_NEXT_REQUEST</td>
<td>This is macro GET_NEXT_REQUEST.</td>
</tr>
<tr>
<td>GET_REQUEST</td>
<td>This is macro GET_REQUEST.</td>
</tr>
<tr>
<td>GET_RESPONSE</td>
<td>This is macro GET_RESPONSE.</td>
</tr>
<tr>
<td>IS_AGENT_PDU</td>
<td>This is macro IS_AGENT_PDU.</td>
</tr>
<tr>
<td>IS ASN_INT</td>
<td>This is macro IS ASN_INT.</td>
</tr>
<tr>
<td>IS ASN_NULL</td>
<td>This is macro IS ASN_NULL.</td>
</tr>
<tr>
<td>IS_GET_NEXT_REQUEST</td>
<td>This is macro IS_GET_NEXT_REQUEST.</td>
</tr>
<tr>
<td>IS_GET_REQUEST</td>
<td>This is macro IS_GET_REQUEST.</td>
</tr>
<tr>
<td>IS_GET_RESPONSE</td>
<td>This is macro IS_GET_RESPONSE.</td>
</tr>
<tr>
<td>IS OCTET_STRING</td>
<td>This is macro IS OCTET_STRING.</td>
</tr>
<tr>
<td>IS_OID</td>
<td>This is macro IS_OID.</td>
</tr>
<tr>
<td>IS_SET_REQUEST</td>
<td>This is macro IS_SET_REQUEST.</td>
</tr>
<tr>
<td>IS_SNMPV3_AUTH_STRUCTURE</td>
<td>This is macro IS_SNMPV3_AUTH_STRUCTURE.</td>
</tr>
<tr>
<td>IS_STRUCTURE</td>
<td>This is macro IS_STRUCTURE.</td>
</tr>
<tr>
<td>IS_TRAP</td>
<td>This is macro IS_TRAP.</td>
</tr>
<tr>
<td>OCTET_STRING</td>
<td>This is macro OCTET_STRING.</td>
</tr>
<tr>
<td>OID_MAX_LEN</td>
<td>Change this to match your OID.</td>
</tr>
<tr>
<td>Macro Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>REPORT_RESPONSE</td>
<td>This is macro REPORT_RESPONSE.</td>
</tr>
<tr>
<td>SET_REQUEST</td>
<td>This is macro SET_REQUEST.</td>
</tr>
<tr>
<td>SNMP_AGENT_PORT</td>
<td>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 MPFS image.</td>
</tr>
<tr>
<td>SNMP_BIB_FILE_NAME</td>
<td></td>
</tr>
<tr>
<td>SNMP_COUNTER32</td>
<td>This is macro SNMP_COUNTER32.</td>
</tr>
<tr>
<td>SNMP_END_OF_VAR</td>
<td>This is macro SNMP_END_OF_VAR.</td>
</tr>
<tr>
<td>SNMP_GAUGE32</td>
<td>This is macro SNMP_GAUGE32.</td>
</tr>
<tr>
<td>SNMP_INDEX_INVALID</td>
<td>This is macro SNMP_INDEX_INVALID.</td>
</tr>
<tr>
<td>SNMP_IP_ADDR</td>
<td>This is macro SNMP_IP_ADDR.</td>
</tr>
<tr>
<td>SNMP_MAX_MSG_SIZE</td>
<td>SNMP MIN and MAX message size 484 bytes in size As per RFC 3411 <a href="https://tools.ietf.org/html/rfc3411">snmpEngineMaxMessageSize</a> and RFC 1157 (section 4- protocol specification) and implementation supports more than 484 whenever feasible.</td>
</tr>
<tr>
<td>SNMP_MAX_OID_LEN_MEM_USE</td>
<td>This macro will be used to avoid SNMP OID memory buffer corruption.</td>
</tr>
<tr>
<td>SNMP_NMS_PORT</td>
<td>This is macro SNMP_NMS_PORT.</td>
</tr>
<tr>
<td>SNMP_NSAP_ADDR</td>
<td>This is macro SNMP_NSAP_ADDR.</td>
</tr>
<tr>
<td>SNMP_OPAQUE</td>
<td>This is macro SNMP_OPAQUE.</td>
</tr>
<tr>
<td>string length.</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SNMP_START_OF_VAR</td>
<td>SNMP_START_OF_VAR.</td>
</tr>
<tr>
<td>SNMP_TIME_TICKS</td>
<td>This is macro SNMP_TIME_TICKS.</td>
</tr>
<tr>
<td>SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE</td>
<td>This macro will be used to avoid SNMP OID memory buffer corruption</td>
</tr>
<tr>
<td>SNMP_V1</td>
<td>This is macro SNMP_V1.</td>
</tr>
<tr>
<td>SNMP_V2C</td>
<td>This is macro SNMP_V2C.</td>
</tr>
<tr>
<td>SNMP_V3</td>
<td>This is macro SNMP_V3.</td>
</tr>
<tr>
<td>STRUCTURE</td>
<td>This is macro TRAP.</td>
</tr>
<tr>
<td>TRAP</td>
<td>This is macro TRAP.</td>
</tr>
<tr>
<td>TRAP_COMMUNITY_MAX_LEN</td>
<td>This is macro TRAP_COMMUNITY_MAX_LEN.</td>
</tr>
<tr>
<td>TRAP_TABLE_SIZE</td>
<td>This is macro TRAP_TABLE_SIZE.</td>
</tr>
</tbody>
</table>

### Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_TYPE_INFO</td>
<td></td>
</tr>
<tr>
<td>OID_INFO</td>
<td></td>
</tr>
<tr>
<td>PDU_INFO</td>
<td></td>
</tr>
<tr>
<td>reqVarErrStatus</td>
<td></td>
</tr>
<tr>
<td>SNMP_NOTIFY_INFO</td>
<td></td>
</tr>
<tr>
<td>SNMP_NONMIBRECINFO</td>
<td>This is type SNMP_NONMIBRECINFO.</td>
</tr>
<tr>
<td>SNMPV3MSGDATA</td>
<td>SNMPv3</td>
</tr>
<tr>
<td>TRAP_INFO</td>
<td>This is type TRAP_INFO.</td>
</tr>
</tbody>
</table>
### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP_ID</td>
<td>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 rule when BIB was generated.</td>
</tr>
<tr>
<td>SNMP_INDEX</td>
<td>This is type SNMP_INDEX.</td>
</tr>
</tbody>
</table>

### Unions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX_INFO</td>
<td></td>
</tr>
<tr>
<td>MIB_INFO</td>
<td></td>
</tr>
<tr>
<td>SNMP_STATUS</td>
<td></td>
</tr>
<tr>
<td>SNMP_VAL</td>
<td>This is type SNMP_VAL.</td>
</tr>
</tbody>
</table>

Stack API > SNMP > Files > SNMP.h

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
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` 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` 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.

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNTP Client Public Members</td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td>SNTP Client Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td>SNTP Client Internal Members</td>
<td>Functions and variables internal to the SNTP Client module</td>
</tr>
</tbody>
</table>

Stack API > SNTP Client
The following functions and variables are available to the stack application.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNTPGetUTCSeconds</td>
<td>Obtains the current time from the SNTP module.</td>
</tr>
</tbody>
</table>

### Module

**SNTP Client**

Stack API > SNTP Client > Public Members
SNTPGetUTCSeconds Function

C

DWORD SNTPGetUTCSeconds();

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.

Preconditions

None

Returns

The number of seconds since the Epoch. (Default 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.
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.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNTPClient</td>
<td>Periodically checks the current time from a pool of servers.</td>
</tr>
</tbody>
</table>

### Module

**SNTP Client**

Stack API > SNTP Client > Stack Members
SNTPClient Function

C

```c
void SNTPClient();
```

Description

This function periodically checks a pool of time servers to obtain the current date/time.

Preconditions

UDP is initialized.

Returns

None

Remarks

This function requires once available UDP socket while processing, but frees that socket when the SNTP module is idle.
SNTP Client Internal Members

The following functions and variables are designated as internal to the SNTP Client module.

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTP_EPOCH</td>
<td>Reference Epoch to use. (default: 01-Jan-1970 00:00:00)</td>
</tr>
<tr>
<td>NTP_FAST_QUERY_INTERVAL</td>
<td>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.</td>
</tr>
<tr>
<td>NTP_QUERY_INTERVAL</td>
<td>Defines how frequently to resynchronize the date/time (default: 10 minutes)</td>
</tr>
<tr>
<td>NTP_REPLY_TIMEOUT</td>
<td>Defines how long to wait before assuming the query has failed</td>
</tr>
<tr>
<td>NTP_SERVER</td>
<td>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.</td>
</tr>
<tr>
<td>NTP_SERVER_PORT</td>
<td>Port for contacting NTP servers</td>
</tr>
</tbody>
</table>

Module

SNTP Client

Structures
### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwLastUpdateTick</td>
<td>Tick count of last update</td>
</tr>
<tr>
<td>dwSNTPSeconds</td>
<td>Seconds value obtained by last update</td>
</tr>
</tbody>
</table>

Stack API > SNTP Client > Internal Members
NTP_PACKET Structure

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;
```

Description

Defines the structure of an NTP packet

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
</table>
| struct {
  BYTE mode : 3;
  BYTE versionNumber : 3;
  BYTE leapIndicator : 2;
} flags; | Flags for the packet |
<p>| BYTE mode : 3; | NTP mode |
| BYTE versionNumber : 3; | SNTP version number |
| BYTE leapIndicator : 2; | Leap second indicator |</p>
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>stratum; Stratum level of local clock</td>
</tr>
<tr>
<td>CHAR</td>
<td>poll; Poll interval</td>
</tr>
<tr>
<td>CHAR</td>
<td>precision; Precision (seconds to nearest power of 2)</td>
</tr>
<tr>
<td>DWORD</td>
<td>root_delay; Root delay between local machine and server</td>
</tr>
<tr>
<td>DWORD</td>
<td>root Dispersion; Root Dispersion (maximum error)</td>
</tr>
<tr>
<td>DWORD</td>
<td>ref_identifier; Reference clock identifier</td>
</tr>
<tr>
<td>DWORD</td>
<td>ref_ts_secs; Reference timestamp (in seconds)</td>
</tr>
<tr>
<td>DWORD</td>
<td>ref_ts_fraq; Reference timestamp (fractions)</td>
</tr>
<tr>
<td>DWORD</td>
<td>orig_ts_secs; Origination timestamp (in seconds)</td>
</tr>
<tr>
<td>DWORD</td>
<td>orig_ts_fraq; Origination timestamp (fractions)</td>
</tr>
<tr>
<td>DWORD</td>
<td>recv_ts_secs; Time at which request arrived at sender (seconds)</td>
</tr>
<tr>
<td>DWORD</td>
<td>recv_ts_fraq; Time at which request arrived at sender (fractions)</td>
</tr>
<tr>
<td>DWORD</td>
<td>tx_ts_secs; Time at which request left sender (seconds)</td>
</tr>
<tr>
<td>DWORD</td>
<td>tx_ts_fraq; Time at which request left sender (fractions)</td>
</tr>
</tbody>
</table>
dwLastUpdateTick Variable

C

DWORD dwLastUpdateTick = 0;

Description

Tick count of last update

Stack API > SNTP Client > Internal Members > dwLastUpdateTick Variable
**dwSNTPSeconds Variable**

```c
DWORD dwSNTPSeconds = 0;
```

**Description**

Seconds value obtained by last update
#define NTP_EPOCH (86400ul * (365ul * 70ul + 17ul))

## Description

Reference Epoch to use. (default: 01-Jan-1970 00:00:00)
NTP_FAST_QUERY_INTERVAL Macro

```c
#define NTP_FAST_QUERY_INTERVAL (14ull * 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.

Stack API > SNTP Client > Internal Members > NTP_FAST_QUERY_INTERVAL Macro
NTP_QUERY_INTERVAL Macro

```c
#define NTP_QUERY_INTERVAL (10ull*60ull * TICK_SECOND)
```

Description

Defines how frequently to resynchronize the date/time (default: 10 minutes)
NTP_REPLY_TIMEOUT Macro

#define NTP_REPLY_TIMEOUT (6ul*TICK_SECOND)

Description

Defines how long to wait before assuming the query has failed
NTP_SERVER Macro

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.
NTP_SERVER_PORT Macro

C

#define NTP_SERVER_PORT (123ul)

Description

Port for contacting NTP servers
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. This implementation of SSL supports the RSA asymmetric encryption protocol and the ARCFOUR symmetric encryption protocol.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>SSL Server Maximum RSA key length (SSL_RSA_KEY_SIZE)</th>
<th>SSL Client Maximum RSA key length (SSL_RSA_CLIENT_SIZE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C18</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>C30</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>C32</td>
<td>2048</td>
<td>2048</td>
</tr>
</tbody>
</table>

Previous versions of the MLA software distribution required that files containing cryptographic algorithm implementations be distributed separately to comply with U.S. Export Controls. In this version, the cryptographic modules are included with the TCP/IP Stack.

**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 on all processors; key lengths up to 2048 bits are supported on PIC32 microcontrollers. The `SSL_RSA_CLIENT_SIZE` macro in `SSLClientSize.h` sets the maximum certificate key length that your client should process.

```c
#define SSL_RSA_CLIENT_SIZE (1024ul) // Size of Encryption Buffer
```

Once the handshake has started, call `TCPSSLIsHandshaking` 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 called UseSSL. That field controls whether or not the SMTP client module will attempt to make an SSL connection before transmitting any data.

Note that TCP sockets using SSL may required an increase in TX/RX buffer size to support SSL. You can adjust the size of your sockets using the TCP/IP Configuration Utility included with the stack.

**SSL Server Support**

To initiate an SSL server, first open a TCP socket for listening using TCPOpen. Then call TCPAddSSLLListener to listen for incoming SSL connections on an alternate port. This allows a single socket to share application-level resources and listen 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 you application will not accept 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.

SSL server support is automatically enabled for HTTP2 when STACK_USE_SSL_SERVER is defined. By default, the HTTP2 module will then listen for unsecured traffic on port 80 and secured connections on port 443.

This SSL server implementation supports key lengths up to 1024 bits on most PIC microcontrollers, and 2048 bits on PIC32 microcontrollers.
The `SSL_RSA_KEY_SIZE` macro in TCPIPConfig.h sets the server certificate key length.

```c
#define SSL_RSA_KEY_SIZE 512ul
```

Note that TCP sockets using SSL may required an increase in TX/RX buffer size to support SSL. You can adjust the size of your sockets using the TCP/IP Configuration Utility included with the stack.

### 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.

**Topics**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Generating Server Certificates</strong></td>
<td>Describes how to import an SSL certificate into your code.</td>
</tr>
<tr>
<td><strong>SSL Public Members</strong></td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td><strong>SSL Stack Members</strong></td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td><strong>SSL Internal Members</strong></td>
<td>Functions and variables internal to the SSL module</td>
</tr>
<tr>
<td><strong>Files</strong></td>
<td>The following table lists files in this documentation.</td>
</tr>
</tbody>
</table>

**Files**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SSLClientSize.h</strong></td>
<td>This is file SSLClientSize.h.</td>
</tr>
</tbody>
</table>

**Stack API > SSL**

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] [Index] [Home]
Generating Server Certificates

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](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 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](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_qInv 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 = 33 prim: INTEGER
:D77566780029FCD610200B66D89507D
915E3E5BB6FAB233B5DFA2E081DF7
```

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:
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:
10. This information must be copied verbatim into the `SSL_CERT[]` array. Note that this is binary data (not a large integer) so it does not get endian-swapped like the private key parameters.

```
ROM BYTE SSL_CERT[524] = {
  0x30, 0x82, 0x02, 0x08, 0x30, 0x01, 0xb2, 0x02, 0x09, 0x00, 0xa5, 0x6a, 0xea, 0x1a, 0xa9,
  0x52, 0x9d, 0x1e, 0x30, 0x00, 0x09, 0x02, 0x0a, 0x08, 0x00, 0xa5, 0x6a, 0xea, 0x1a, 0xa9,
  0x52, 0x9d, 0x1e, 0x30, 0x00, 0x09, 0x02, 0x0a, 0x08, 0x00, 0xa5, 0x6a, 0xea, 0x1a, 0xa9,
  0x52, 0x9d, 0x1e, 0x30, 0x00, 0x09, 0x02, 0x0a, 0x08, 0x00, 0xa5, 0x6a, 0xea, 0x1a, 0xa9,
};
```
11. Update the `SSL_CERT_LEN` variable to contain the correct value.
SSL Public Members

The following functions and variables are available to the stack application.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_SUPPLEMENTARY_DATA_TYPES</td>
<td>This is type SSL_SUPPLEMENTARY_DATA_TYPES.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPAddSSLListener</td>
<td>Listens for SSL connection on a specific port.</td>
</tr>
<tr>
<td>TCPSSLIsHandshaking</td>
<td>Determines if an SSL session is still handshaking.</td>
</tr>
<tr>
<td>TCPStartSSLClient</td>
<td>Begins an SSL client session.</td>
</tr>
<tr>
<td>TCPIsSSL</td>
<td>Determines if a TCP connection is secured with SSL.</td>
</tr>
<tr>
<td>SSLStartSession</td>
<td>Begins a new SSL session for the given TCP connection.</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_INVALID_ID</td>
<td>Identifier for invalid SSL allocations</td>
</tr>
<tr>
<td>SSL_RSA_KEY_SIZE</td>
<td>Bits in SSL RSA key. This parameter is used for SSL sever connections only. The only valid value is 512 bits (768 and 1024 bits do not work at this time). Note, however, that SSL client operations do currently work up to 1024 bit RSA key length.</td>
</tr>
<tr>
<td>SSL_RSA_CLIENT_SIZE</td>
<td>Size of Encryption Buffer (must be larger than key size)</td>
</tr>
</tbody>
</table>
## Module

**SSL**

### Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_PKEY_INFO</td>
<td>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.</td>
</tr>
</tbody>
</table>

Stack API > SSL > Public Members

---

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] [Index] [Home]
SSL_INVALID_ID Macro

C

#define SSL_INVALID_ID (0xFFu) // Identifier for invalid SSL allocations

Description

Identifier for invalid SSL allocations

Stack API > SSL > Public Members > SSL_INVALID_ID Macro
TCPAddSSLListener Function

C

```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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to secure</td>
</tr>
<tr>
<td>port</td>
<td>SSL port to listen on</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>SSL port was added.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The socket was not a listening socket.</td>
</tr>
</tbody>
</table>
C

BOOL TCPSSLIsHandshaking(TCP_SOCKET hTCP);

Description

Call this function after calling TCPStartSSLClient 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to check</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>SSL handshake is still progressing</td>
</tr>
<tr>
<td>FALSE</td>
<td>SSL handshake has completed</td>
</tr>
</tbody>
</table>

Stack API > SSL > Public Members > TCPSSLIsHandshaking Function
TCPStartSSLClient Function

```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.

**Preconditions**

TCP is initialized and hTCP is already connected.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to secure</td>
</tr>
<tr>
<td>host</td>
<td>Expected host name on certificate (currently ignored)</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>an SSL connection was initiated</td>
</tr>
<tr>
<td>FALSE</td>
<td>Insufficient SSL resources (stubs) were available</td>
</tr>
</tbody>
</table>

**Remarks**

The host parameter is currently ignored and is not validated.
TCPIsSSL Function

```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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to check</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Connection is secured via SSL</td>
</tr>
<tr>
<td>FALSE</td>
<td>Connection is not secured</td>
</tr>
</tbody>
</table>

Stack API > SSL > Public Members > TCPIsSSL Function
SSLStartSession Function

```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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the socket to begin the SSL connection on</td>
</tr>
<tr>
<td>buffer</td>
<td>pointer to a supplementary data buffer</td>
</tr>
<tr>
<td>supDataType</td>
<td>type of supplementary data to store</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_INVALID_ID</td>
<td>insufficient SSL resources to start a new connection</td>
</tr>
<tr>
<td>others</td>
<td>the allocated SSL stub ID</td>
</tr>
</tbody>
</table>

Stack API > SSL > Public Members > SSLStartSession Function
SSL_SUPPLEMENTARY_DATA_TYPES

Enumeration

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.
SSL_PKEY_INFO Structure

C

typedef struct {
    WORD pub_size_bytes;
    BYTE pub_key[SSL_RSA_CLIENT_SIZE/8];
    BYTE pub_e[3];
    BYTE pub_guid;
} SSL_PKEY_INFO;

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.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE pub_guid;</td>
<td>This is used as a TCP_SOCKET which is a BYTE</td>
</tr>
</tbody>
</table>

Stack API > SSL > Public Members > SSL_PKEY_INFO Structure
SSL_RSA_KEY_SIZE Macro

```
C
#define SSL_RSA_KEY_SIZE (512ul)
```

**Description**

Bits in SSL RSA key. This parameter is used for SSL sever connections only. The only valid value is 512 bits (768 and 1024 bits do not work at this time). Note, however, that SSL client operations do currently work up to 1024 bit RSA key length.
#define SSL_RSA_CLIENT_SIZE (1024ul)  // Size of Encryption Buffer (must be larger than key size)
SSL Stack Members

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.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_STATE</td>
<td>This is type SSL_STATE.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLInit</td>
<td>Initializes the SSL engine.</td>
</tr>
<tr>
<td>SSLPeriodic</td>
<td>Performs any periodic tasks for the SSL module.</td>
</tr>
<tr>
<td>TCPRequestSSLMessage</td>
<td>Requests an SSL message to be transmitted.</td>
</tr>
<tr>
<td>TCPSSLGetPendingTxSize</td>
<td>Determines how many bytes are pending for a future SSL record.</td>
</tr>
<tr>
<td>TCPSSLHandleIncoming</td>
<td>Hands newly arrive TCP data to the SSL module for processing.</td>
</tr>
<tr>
<td>TCPSSLHandshakeComplete</td>
<td>Clears the SSL handshake flag.</td>
</tr>
<tr>
<td>TCPSSLInPlaceMACEncrypt</td>
<td>Encrypts and MACs data in place in the TCP TX buffer.</td>
</tr>
<tr>
<td>TCPSSLPutRecordHeader</td>
<td>Writes an SSL record header and sends an SSL record.</td>
</tr>
<tr>
<td>TCPStartSSLServer</td>
<td>Begins an SSL server session.</td>
</tr>
</tbody>
</table>

Macros
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_MIN_SESSION_LIFETIME</td>
<td>Minimum lifetime for SSL Sessions cannot be reallocated until this much time has elapsed</td>
</tr>
<tr>
<td>SSL_RSA_LIFETIME_EXTENSION</td>
<td>Lifetime extension for RSA operations Sessions lifetime is extended by this amount when an RSA calculation is made</td>
</tr>
</tbody>
</table>

### Module

**SSL**

Stack API > SSL > Stack Members
SSL_STATE Enumeration

C

typedef enum {
    SSL_NONE = 0,
    SSL_HANDSHAKING,
    SSL_ESTABLISHED,
    SSL_CLOSED
} SSL_STATE;

Description

This is type SSL_STATE.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_NONE = 0</td>
<td>No security is enabled</td>
</tr>
<tr>
<td>SSL_HANDSHAKING</td>
<td>Handshake is progressing (no application data allowed)</td>
</tr>
<tr>
<td>SSL_ESTABLISHED</td>
<td>Connection is established and secured</td>
</tr>
<tr>
<td>SSL_CLOSED</td>
<td>Connection has been closed (no application data is allowed)</td>
</tr>
</tbody>
</table>

Stack API > SSL > Stack Members > SSL_STATE Enumeration
SSLInit Function

C

```c
void SSLInit();
```

Description

Initializes the SSL engine.

Preconditions

None

Returns

None

Remarks

This function is called only one during lifetime of the application.
SSLPeriodic Function

C

```c
void SSLPeriodic(
    TCP_SOCKET hTCP,
    BYTE sslStubID
);
```

Description

This function performs periodic tasks for the SSL module. This includes processing for RSA operations.

Preconditions

SSL has already been initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the socket for which to perform periodic functions</td>
</tr>
<tr>
<td>id</td>
<td>the SSL stub to use</td>
</tr>
</tbody>
</table>

Returns

None

Stack API > SSL > Stack Members > SSLPeriodic Function
TCPRequestSSLMessage Function

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to use</td>
</tr>
<tr>
<td>msg</td>
<td>One of the SSL_MESSAGE types to transmit.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The message was requested.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Another message is already pending transmission.</td>
</tr>
</tbody>
</table>

Stack API > SSL > Stack Members > TCPRequestSSLMessage Function
TCPSSLGetPendingTxSize Function

C

WORD TCPSSLGetPendingTxSize(
    TCP_SOCKET hTCP
);

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to check</td>
</tr>
</tbody>
</table>

Returns

None
TCP_SSLHandleIncoming Function

C

```c
void TCP_SSLHandleIncoming(
    TCP_SOCKET hTCP
);
```

Description

This function processes incoming TCP data as an SSL record and performs any necessary repositioning and decrypting.

Preconditions

TCP is initialized, and hTCP is connected with an active SSL session.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to handle incoming data on</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

This function should never be called by an application. It is used only by the SSL module itself.
TCPSSLHandshakeComplete Function

C

```c
void TCPSSLHandshakeComplete(
    TCP_SOCKET hTCP
);
```

Description

This function clears the flag indicating that an SSL handshake is complete.

Preconditions

TCP is initialized and hTCP is connected.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to set</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

This function should never be called by an application. It is used only by the SSL module itself.
TCPSSLInPlaceMACEncrypt Function

```c
void TCPSSLInPlaceMACEncrypt(
    TCP_SOCKET hTCP,
    ARCFOUR_CTX* ctx,
    BYTE* MACSecret,
    WORD len
);
```

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.

Preconditions

TCP is initialized, hTCP is connected, and ctx's Sbox is loaded.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to encrypt in</td>
</tr>
<tr>
<td>ctx</td>
<td>ARCFOUR encryption context to use</td>
</tr>
<tr>
<td>MACSecret</td>
<td>MAC encryption secret to use</td>
</tr>
<tr>
<td>len</td>
<td>Number of bytes to crypt</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

This function should never be called by an application. It is used only by
the SSL module itself.
**TCPSSLPutRecordHeader Function**

```c
void TCPSSLPutRecordHeader(
    TCP_SOCKET hTCP,
    BYTE* hdr,
    BOOL recDone
);
```

**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.

**Preconditions**

TCP is initialized, and hTCP is connected with an active SSL session.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to write the header and transmit with</td>
</tr>
<tr>
<td>hdr</td>
<td>Record header (5 bytes) to send or NULL to just move the pointerctx</td>
</tr>
<tr>
<td>recDone</td>
<td>TRUE if the record is done, FALSE otherwise</td>
</tr>
</tbody>
</table>

**Returns**
None

**Remarks**

This function should never be called by an application. It is used only by the SSL module itself.
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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to secure</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>an SSL connection was initiated</td>
</tr>
<tr>
<td>FALSE</td>
<td>Insufficient SSL resources (stubs) were available</td>
</tr>
</tbody>
</table>
#define SSL_MIN_SESSION_LIFETIME (1*TICK_SECOND)

**Description**

Minimum lifetime for SSL Sessions Sessions cannot be reallocated until this much time has elapsed
SSL_RSA_LIFETIME_EXTENSION Macro

```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.
SSL Internal Members

The following functions and variables are designated as internal to the SSL module.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM_SSL_RX_SERVER_HELLO</td>
<td>State machine for SSLRxServerHello</td>
</tr>
<tr>
<td>SSL_ALERT_LEVEL</td>
<td>Describes the two types of Alert records</td>
</tr>
<tr>
<td>SSL_MESSAGES</td>
<td>Describes the types of SSL messages (handshake and alerts)</td>
</tr>
<tr>
<td>SSL_SESSION_TYPE</td>
<td>SSL Session Type Enumeration</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalculateFinishedHash</td>
<td>Calculates the handshake hash over the data. hashID can be either MD5 or SHA-1, and this function will calculate accordingly.</td>
</tr>
<tr>
<td>GenerateHashRounds</td>
<td>Generates hash rounds to find either the Master Secret or the Key Block.</td>
</tr>
<tr>
<td>GenerateSessionKeys</td>
<td>Generates the session write keys and MAC secrets</td>
</tr>
<tr>
<td>HSEnd</td>
<td>Hashes the message contents into the Handshake hash structures and begins a new handshake hash.</td>
</tr>
<tr>
<td>HGet</td>
<td>Reads data from socket, transparently hashing it into the handshake hashes.</td>
</tr>
</tbody>
</table>

- Function: static WORD HSGetArray(TCP_SOCKET skt, BYTE *data, WORD len)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| **HSGetArray** | • 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 |
| **HSGetWord** | Reads data from socket, transparently hashing it into the handshake hashes. |
| **HSPut** | Writes data to socket, transparently hashing it into the handshake hashes. |
| **Function: static WORD** | HSPutArray(TCP_SOCKET skt, BYTE *data, BYTE len) |
| **PreCondition: None** | |
| **Input: skt - socket to write data to** |  
**data - data to write**  
**len - number of bytes to write** |
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HSPutArray</strong></td>
<td>Output: Number of bytes written</td>
</tr>
<tr>
<td></td>
<td>Side Effects: None</td>
</tr>
<tr>
<td></td>
<td>Overview: Writes data to socket, transparently hashing it into the handshake hashes.</td>
</tr>
<tr>
<td></td>
<td>Note: None</td>
</tr>
<tr>
<td><strong>HSPutROMArray</strong></td>
<td>This is function HSPutROMArray.</td>
</tr>
<tr>
<td><strong>HSPutWord</strong></td>
<td>Writes data to socket, transparently hashing it into the handshake hashes.</td>
</tr>
<tr>
<td><strong>HSStart</strong></td>
<td>Sets up the buffer to store data for handshake hash tracking</td>
</tr>
<tr>
<td><strong>LoadOffChip</strong></td>
<td>Copies data from Ethernet RAM to local RAM</td>
</tr>
<tr>
<td><strong>SaveOffChip</strong></td>
<td>Copies data in PIC RAM to the Ethernet RAM</td>
</tr>
<tr>
<td><strong>SSLBufferAlloc</strong></td>
<td>Allocates a buffer for use.</td>
</tr>
<tr>
<td><strong>SSLBufferFree</strong></td>
<td>Specified buffer is released</td>
</tr>
<tr>
<td><strong>SSLBufferSync</strong></td>
<td>Specified buffer is loaded to RAM. Only loads if necessary, and saves any current buffer before switching.</td>
</tr>
<tr>
<td><strong>SSLHashAlloc</strong></td>
<td>Allocates a hash for use.</td>
</tr>
<tr>
<td><strong>SSLHashFree</strong></td>
<td>Specified hash is released</td>
</tr>
<tr>
<td><strong>SSLHashSync</strong></td>
<td>Specified hash is loaded to RAM. Only loads if necessary, and saves any current hash before switching.</td>
</tr>
<tr>
<td><strong>SSLKeysSync</strong></td>
<td>Specified key set is loaded to RAM. Only loads if necessary, and saves any current key set before</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SSLMACAdd</td>
<td>This is function SSLMACAdd.</td>
</tr>
<tr>
<td>SSLMACBegin</td>
<td>This is function SSLMACBegin.</td>
</tr>
<tr>
<td>SSLMACCalc</td>
<td>This is function SSLMACCalc.</td>
</tr>
<tr>
<td>SSLRSAOperation</td>
<td>Pauses connection processing until RSA calculation is complete.</td>
</tr>
<tr>
<td>SSLxAlert</td>
<td>Receives an alert message and decides what to do</td>
</tr>
<tr>
<td>SSLxAntiqueClientHello</td>
<td>Receives the SSLv2 ClientHello message, initiating a new SSL session with a client</td>
</tr>
<tr>
<td>SSLxCCS</td>
<td>Receives a ChangeCipherSpec from the remote server</td>
</tr>
<tr>
<td>SSLxClientHello</td>
<td>Receives the ClientHello message, initiating a new SSL session with a client</td>
</tr>
<tr>
<td>SSLxClientKeyExchange</td>
<td>Receives the ClientKeyExchange message and begins the decryption process.</td>
</tr>
<tr>
<td>SSLxFinished</td>
<td>Receives the Finished message from remote node</td>
</tr>
<tr>
<td>SSLxHandshake</td>
<td>Receives a handshake message.</td>
</tr>
<tr>
<td>SSLxRecord</td>
<td>Receives an SSL record.</td>
</tr>
<tr>
<td>SSLxServerCertificate</td>
<td>Receives ServerCertificate from the remote server, locates the public key information, and executes RSA operation.</td>
</tr>
<tr>
<td>SSLxServerHello</td>
<td>Receives the ServerHello from the remote server</td>
</tr>
<tr>
<td>SSLSessionMatchID</td>
<td>Locates a cached SSL session for reuse. Syncs found session into RAM.</td>
</tr>
<tr>
<td>SSLSessionMatchIP</td>
<td>Locates a cached SSL session for reuse</td>
</tr>
<tr>
<td>SSLSessionNew</td>
<td>Finds space for a new SSL session</td>
</tr>
<tr>
<td>SSLSessionSync</td>
<td>Specified session is loaded to RAM. Only loads if necessary, and saves any current session before</td>
</tr>
</tbody>
</table>

switching.
switching if it has been updated.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLStartPartialRecord</td>
<td>Begins a long SSL record.</td>
</tr>
<tr>
<td>SSLStubAlloc</td>
<td>Allocates a stub for use.</td>
</tr>
<tr>
<td>SSLStubFree</td>
<td>Specified stub is released</td>
</tr>
<tr>
<td>SSLStubSync</td>
<td>Specified stub is loaded to RAM. Only loads if necessary, and saves any current stub before switching.</td>
</tr>
<tr>
<td>SSLTerminate</td>
<td>Terminates an SSL connection and releases allocated resources.</td>
</tr>
<tr>
<td>SSLTxCCSFin</td>
<td>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.</td>
</tr>
<tr>
<td>SSLTxClientHello</td>
<td>Transmits the ClientHello message to initiate a new SSL session with the server.</td>
</tr>
<tr>
<td>SSLTxClientKeyExchange</td>
<td>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.</td>
</tr>
<tr>
<td>SSLTxMessage</td>
<td>Transmits an SSL message.</td>
</tr>
<tr>
<td>SSLTxRecord</td>
<td>Transmits an SSL record.</td>
</tr>
<tr>
<td>SSLTxServerCertificate</td>
<td>Transmits the Certificate message with the server’s specified public key certificate.</td>
</tr>
<tr>
<td>SSLTxServerHello</td>
<td>Transmits the ServerHello message.</td>
</tr>
<tr>
<td>SSLTxServerHelloDone</td>
<td>Transmits the ServerHelloDone message.</td>
</tr>
</tbody>
</table>

**Macros**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESERVED_SSL_MEMORY</td>
<td>Total space needed by all SSL storage.</td>
</tr>
<tr>
<td>Requirement</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SSL_ALERT</strong></td>
<td>Protocol code for Alert records</td>
</tr>
<tr>
<td><strong>SSL_APPLICATION</strong></td>
<td>Protocol code for Application data records</td>
</tr>
<tr>
<td><strong>SSL_BASE_BUFFER_ADDR</strong></td>
<td>Base address for SSL buffers</td>
</tr>
<tr>
<td><strong>SSL_BASE_HASH_ADDR</strong></td>
<td>Base address for SSL hashes</td>
</tr>
<tr>
<td><strong>SSL_BASE_KEYS_ADDR</strong></td>
<td>Base address for SSL keys</td>
</tr>
<tr>
<td><strong>SSL_BASE_SESSION_ADDR</strong></td>
<td>Base address for SSL sessions</td>
</tr>
<tr>
<td><strong>SSL_BASE_STUB_ADDR</strong></td>
<td>Base address for SSL stubs</td>
</tr>
<tr>
<td><strong>SSL_BUFFER_SIZE</strong></td>
<td>Amount of space needed by a single SSL buffer</td>
</tr>
<tr>
<td><strong>SSL_BUFFER_SPACE</strong></td>
<td>Amount of space needed by all SSL buffers</td>
</tr>
<tr>
<td><strong>SSL_CHANGE_CIPHER_SPEC</strong></td>
<td>Protocol code for Change Cipher Spec records</td>
</tr>
<tr>
<td><strong>SSL_HANDSHAKE</strong></td>
<td>Protocol code for Handshake records</td>
</tr>
<tr>
<td><strong>SSL_HASH_SIZE</strong></td>
<td>Amount of space needed by a single SSL hash</td>
</tr>
<tr>
<td><strong>SSL_HASH_SPACE</strong></td>
<td>Amount of space needed by all SSL hashes</td>
</tr>
<tr>
<td><strong>SSL_KEYS_SIZE</strong></td>
<td>Amount of space needed by a single SSL key</td>
</tr>
<tr>
<td><strong>SSL_KEYS_SPACE</strong></td>
<td>Amount of space needed by all SSL keys</td>
</tr>
<tr>
<td><strong>SSL_RSA_EXPORT_WITH_ARCFOUR_40_MD5</strong></td>
<td>This is macro SSL_RSA_EXPORT_WITH_ARCFOUR_40_MD5</td>
</tr>
<tr>
<td><strong>SSL_RSA_WITH_ARCFOUR_128_MD5</strong></td>
<td>This is macro SSL_RSA_WITH_ARCFOUR_128_MD5</td>
</tr>
<tr>
<td><strong>SSL_SESSION_SIZE</strong></td>
<td>Amount of space needed by a single SSL session</td>
</tr>
<tr>
<td><strong>SSL_SESSION_SPACE</strong></td>
<td>Amount of space needed by all SSL sessions</td>
</tr>
<tr>
<td><strong>SSL_STUB_SIZE</strong></td>
<td>Amount of space needed by a single SSL stub</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SSL_STUB_SPACE</strong></td>
<td>Amount of space needed by all SSL stubs.</td>
</tr>
<tr>
<td><strong>SSL_VERSION</strong></td>
<td>SSL version number.</td>
</tr>
<tr>
<td><strong>SSL_VERSION_HI</strong></td>
<td>SSL version number (high byte).</td>
</tr>
<tr>
<td><strong>SSL_VERSION_LO</strong></td>
<td>SSL version number (low byte).</td>
</tr>
<tr>
<td><strong>SSLFinishPartialRecord</strong></td>
<td>This is macro SSLFinishPartialRecord.</td>
</tr>
<tr>
<td><strong>SSLFlushPartialRecord</strong></td>
<td>This is macro SSLFlushPartialRecord.</td>
</tr>
<tr>
<td><strong>SSLSessionUpdated</strong></td>
<td>This is macro SSLSessionUpdated.</td>
</tr>
</tbody>
</table>

### Module

**SSL**

### Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SSL_KEYS</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>SSL_SESSION</strong></td>
<td>Storage space for SSL Session identifiers. (The SessionID and MasterSecret)</td>
</tr>
<tr>
<td><strong>SSL_SESSION_STUB</strong></td>
<td>Stub value for an SSL_SESSION. 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.</td>
</tr>
<tr>
<td><strong>SSL_STUB</strong></td>
<td>Memory holder for general information associated with an SSL connection.</td>
</tr>
</tbody>
</table>
### Unions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_BUFFER</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isBufferUsed</td>
<td>Indicates which buffers are in use</td>
</tr>
<tr>
<td>isHashUsed</td>
<td>Indicates which hashes are in use</td>
</tr>
<tr>
<td>isStubUsed</td>
<td>Indicates which stubs are in use</td>
</tr>
<tr>
<td>masks</td>
<td>Masks for each bit in the is*Used variables</td>
</tr>
<tr>
<td>ptrHS</td>
<td>Used in buffering handshake results</td>
</tr>
<tr>
<td>SSL_CERT</td>
<td>RSA public certificate data ?</td>
</tr>
<tr>
<td>SSL_CERT_LEN</td>
<td>RSA public certificate length ?</td>
</tr>
<tr>
<td>sslBufferID</td>
<td>Which buffer is loaded</td>
</tr>
<tr>
<td>sslHash</td>
<td>Hash storage</td>
</tr>
<tr>
<td>sslHashID</td>
<td>Which hash is loaded</td>
</tr>
<tr>
<td>sslKeys</td>
<td>The current SSL session</td>
</tr>
<tr>
<td>sslKeysID</td>
<td>Which SSL_KEYS are loaded</td>
</tr>
<tr>
<td>sslRSAStubID</td>
<td>Which stub is using RSA, if any</td>
</tr>
<tr>
<td>sslSession</td>
<td>Current session data</td>
</tr>
<tr>
<td>sslSessionID</td>
<td>Which session is loaded</td>
</tr>
<tr>
<td>Symbol</td>
<td>Variable Name</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>🌷</td>
<td>sslSessionStubs</td>
</tr>
<tr>
<td>🌷</td>
<td>sslSessionUpdated</td>
</tr>
<tr>
<td>🌷</td>
<td>sslStub</td>
</tr>
<tr>
<td>🌷</td>
<td>sslStubID</td>
</tr>
</tbody>
</table>

**Stack API > SSL > Internal Members**
CalculateFinishedHash Function

C

```c
static void CalculateFinishedHash(
    BYTE hashID,
    BOOL fromClient,
    BYTE * result
);
```

Description

Calculates the handshake hash over the data. hashID can be either MD5 or SHA-1, and this function will calculate accordingly.

Preconditions

hashID has all handshake data hashed so far and the current session is synced in.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hashID</td>
<td>the hash sum to use</td>
</tr>
<tr>
<td>fromClient</td>
<td>TRUE if client is sender</td>
</tr>
<tr>
<td>result</td>
<td>where to store results</td>
</tr>
</tbody>
</table>

Returns

None

Side Effects

None

Remarks
GenerateHashRounds Function

```c
static void GenerateHashRounds(
    BYTE num,
    BYTE* rand1,
    BYTE* rand2
);
```

Description

Generates hash rounds to find either the Master Secret or the Key Block.

Preconditions

The SSL buffer is allocated for temporary usage and the data to run rounds on is in sslSession.masterSecret

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>how many rounds to compute</td>
</tr>
<tr>
<td>rand1</td>
<td>the first random data block</td>
</tr>
<tr>
<td>rand2</td>
<td>the second random data block</td>
</tr>
</tbody>
</table>

Returns

None

Side Effects

Destroys the SSL Buffer space

Remarks
This function will overflow the buffer after 7 rounds, but in practice num = 3 or num = 4.
GenerateSessionKeys Function

C

static void GenerateSessionKeys();

Description

Generates the session write keys and MAC secrets

Preconditions

The SSL buffer is allocated for temporary usage, session keys are synced, and the TX and RX buffers are allocated for S-boxes.

Returns

None

Side Effects

Destroys the SSL Buffer Space

Remarks

None
HSEnd Function

| c
| static void HSEnd();

Description

Hashes the message contents into the Handshake hash structures and begins a new handshake hash.

Preconditions

None

Returns

None

Side Effects

None

Remarks

None
HSGet Function

C

```c
static WORD HSGet(TCP_SOCKET skt, BYTE * b);
```

Description

Reads data from socket, transparently hashing it into the handshake hashes.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>skt</td>
<td>socket to read data from</td>
</tr>
<tr>
<td>b</td>
<td>byte to read into</td>
</tr>
</tbody>
</table>

Returns

Number of bytes read

Side Effects

None

Remarks

None
HSGetArray Function

C

```c
static WORD HSGetArray(
    TCP_SOCKET skt,
    BYTE * data,
    WORD len
);
```

Description

- Function: static WORD HSGetArray(TCP_SOCKET skt, BYTE *data, WORD len)

- PreCondition: None

- Input: skt - socket to read data from
data - array to read into, or NULLlen - 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

Stack API > SSL > Internal Members > HSGetArray Function
HSGetWord Function

C

```c
static WORD HSGetWord(
    TCP_SOCKET skt,
    WORD * w
);
```

Description

Reads data from socket, transparently hashing it into the handshake hashes.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>skt</td>
<td>socket to read data from</td>
</tr>
<tr>
<td>w</td>
<td>word to read into</td>
</tr>
</tbody>
</table>

Returns

Number of bytes read

Side Effects

None

Remarks

None

Stack API > SSL > Internal Members > HSGetWord Function
HSPut Function

C

```c
static WORD HSPut(
    TCP_SOCKET skt,
    BYTE b
);
```

**Description**

Writes data to socket, transparently hashing it into the handshake hashes.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>skt</td>
<td>socket to write data to</td>
</tr>
<tr>
<td>b</td>
<td>byte to write</td>
</tr>
</tbody>
</table>

**Returns**

Number of bytes written

**Side Effects**

None

**Remarks**

None

[Stack API] > SSL > Internal Members > HSPut Function
HSPutArray Function

C

```c
static WORD HSPutArray(
    TCP_SOCKET skt,
    BYTE * data,
    WORD len
);
```

Description

- Function: static WORD HSPutArray(TCP_SOCKET 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
HSPutROMArray Function

C

```c
static WORD HSPutROMArray(
    TCP_SOCKET skt,
    ROM BYTE * data,
    WORD len
);
```

Description

This is function HSPutROMArray.
HSPutWord Function

C

```
static WORD HSPutWord(
    TCP_SOCKET skt,
    WORD w
);
```

Description

Writes data to socket, transparently hashing it into the handshake hashes.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>skt</td>
<td>socket to write data to</td>
</tr>
<tr>
<td>w</td>
<td>word to write</td>
</tr>
</tbody>
</table>

Returns

Number of bytes written

Side Effects

None

Remarks

None

Stack API > SSL > Internal Members > HSPutWord Function
HSStart Function

```c
static void HSStart();
```

Description

Sets up the buffer to store data for handshake hash tracking

Preconditions

None

Returns

None

Side Effects

None

Remarks

None
isBufferUsed Variable

```c
WORD isBufferUsed;
```

**Description**

Indicates which buffers are in use
isHashUsed Variable

C
WORD isHashUsed;

Description

Indicates which hashes are in use
isStubUsed Variable

```c
WORD isStubUsed;
```

Description

Indicates which stubs are in use

Stack API > SSL > Internal Members > isStubUsed Variable
masks Variable

| C ROM WORD masks[16] = { 0x0001, 0x0002, 0x0004, 0x0008, 0x0010, 0x0020, |

Description

Masks for each bit in the is*Used variables
ptrHS Variable

```c
BYTE * ptrHS;
```

Description

Used in buffering handshake results

Stack API > SSL > Internal Members > ptrHS Variable
RESERVED_SSL_MEMORY Macro

```
C
#define RESERVED_SSL_MEMORY ((DWORD)(SSL_STUB_SPACE + SSL_KEYS_SPACE + ...
```

Description

Total space needed by all SSL storage requirements

Stack API > SSL > Internal Members > RESERVED_SSL_MEMORY Macro
LoadOffChip Function

```c
static void LoadOffChip(
    BYTE * ramAddr,
    PTR_BASE ethAddr,
    WORD len
);
```

**Description**

Copies data from Ethernet RAM to local RAM

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ramAddr</td>
<td>destination address in RAM</td>
</tr>
<tr>
<td>ethAddr</td>
<td>source address in Ethernet RAM</td>
</tr>
<tr>
<td>len</td>
<td>number of bytes to copy</td>
</tr>
</tbody>
</table>

**Returns**

None

**Side Effects**

None

**Remarks**

None
SaveOffChip Function

C

```c
static void SaveOffChip(
  BYTE * ramAddr,
  PTR_BASE ethAddr,
  WORD len
);
```

Description

Copies data in PIC RAM to the Ethernet RAM

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ramAddr</td>
<td>source address in RAM</td>
</tr>
<tr>
<td>ethAddr</td>
<td>destination address in Ethernet RAM</td>
</tr>
<tr>
<td>len</td>
<td>number of bytes to copy</td>
</tr>
</tbody>
</table>

Returns

None

Side Effects

None

Remarks

None
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

Stack API > SSL > Internal Members > SM_SSL_RX_SERVER_HELLO Enumeration
# SSL_ALERT Macro

```c
#define SSL_ALERT 21u  // Protocol code for Alert records
```

## Description

Protocol code for Alert records

---

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] | [Index] | [Home]
SSL_ALERT_LEVEL Enumeration

**C**

typedef enum {
    SSL_ALERT.Warning = 1u,
    SSL_ALERT.Fatal = 2u
} SSL_ALERT_LEVEL;

**Description**

Describes the two types of Alert records

**Members**

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_ALERT.WARNING = 1u</td>
<td>Alert message is a warning (session can be resumed)</td>
</tr>
<tr>
<td>SSL_ALERT_FATAL = 2u</td>
<td>Alert message is fatal (session is non-resumable)</td>
</tr>
</tbody>
</table>

Stack API > SSL > Internal Members > SSL_ALERT_LEVEL Enumeration

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
SSL_APPLICATION Macro

C

#define SSL_APPLICATION 23u // Protocol code for Application data records

Description

Protocol code for Application data records

Stack API > SSL > Internal Members > SSL_APPLICATION Macro
SSL_BASE_BUFFER_ADDR Macro

```
#define SSL_BASE_BUFFER_ADDR (BASE_SSLB_ADDR + SSL_STUB_SPACE + SSL_KEYS_SPACE)
```

Description

Base address for SSL buffers

Stack API > SSL > Internal Members > SSL_BASE_BUFFER_ADDR Macro
SSL_BASE_HASH_ADDR Macro

```c
#define SSL_BASE_HASH_ADDR (BASE_SSLB_ADDR + SSL_STUB_SPACE + SSL_KEYS
```

Description

Base address for SSL hashes

Stack API > SSL > Internal Members > SSL_BASE_HASH_ADDR Macro
SSL_BASE_KEYS_ADDR Macro

C

#define SSL_BASE_KEYS_ADDR (BASE_SSLB_ADDR + SSL_STUB_SPACE)

Description

Base address for SSL keys
#SSL_BASE_SESSION_ADDR Macro

```c
#define SSL_BASE_SESSION_ADDR (BASE_SSLB_ADDR + SSL_STUB_SPACE + SSL_KEYS_SPACE)
```

##Description

Base address for SSL sessions
SSL_BASE_STUB_ADDR Macro

C

#define SSL_BASE_STUB_ADDR (BASE_SSLB_ADDR)

Description

Base address for SSL stubs
### SSL_BUFFER Union

C

```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.
SSL_BUFFER_SIZE Macro

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

Stack API > SSL > Internal Members > SSL_BUFFER_SIZE Macro
#SSL_BUFFER_SPACE Macro

```c
#define SSL_BUFFER_SPACE (SSL_BUFFER_SIZE*MAX_SSL_BUFFERS) // A
```

**Description**

Amount of space needed by all SSL buffer

[Stack API > SSL > Internal Members > SSL_BUFFER_SPACE Macro](#)
### SSL_CERT Variable

| ROM BYTE SSL_CERT[];

#### Description

RSA public certificate data?

Stack API > SSL > Internal Members > SSL_CERT Variable
**SSL_CERT_LEN Variable**

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM WORD SSL_CERT_LEN;</td>
</tr>
</tbody>
</table>

**Description**

RSA public certificate length ?

Stack API > SSL > Internal Members > SSL_CERT_LEN Variable

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
SSL_CHANGE_CIPHER_SPEC Macro

C

#define SSL_CHANGE_CIPHER_SPEC 20u // Protocol code for Change Cipher Spec records

Description

Protocol code for Change Cipher Spec records

Stack API > SSL > Internal Members > SSL_CHANGE_CIPHER_SPEC Macro
SSL_HANDSHAKE Macro

C

#define SSL_HANDSHAKE 22u // Protocol code for Handshake records

Description

Protocol code for Handshake records

Stack API > SSL > Internal Members > SSL_HANDSHAKE Macro
SSL_HASH_SIZE Macro

```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
# SSL_HASH_SPACE Macro

```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

---

**Stack API > SSL > Internal Members > SSL_HASH_SPACE Macro**

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
**SSL_KEYS Structure**

```c
typedef struct {
    union {
        struct {
            BYTE MACSecret[16];
            DWORD sequence;
            ARCFOUR_CTX cryptCtx;
            BYTE reserved[6];
        } app;
        BYTE random[32];
    } Local;
    union {
        struct {
            BYTE MACSecret[16];
            DWORD sequence;
            ARCFOUR_CTX cryptCtx;
            BYTE reserved[6];
        } app;
        BYTE random[32];
    } Remote;
} SSL_KEYS;
```

**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.

**Members**

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE MACSecret[16];</td>
<td>Server's MAC write secret</td>
</tr>
<tr>
<td>DWORD sequence;</td>
<td>Server's write sequence number</td>
</tr>
<tr>
<td>ARCFOUR_CTX cryptCtx;</td>
<td>Server's write encryption context</td>
</tr>
<tr>
<td>Description</td>
<td>Purpose</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>BYTE reserved[6];</td>
<td>Future expansion</td>
</tr>
<tr>
<td>BYTE random[32];</td>
<td>Server.random value</td>
</tr>
<tr>
<td>BYTE MACSecret[16];</td>
<td>Client's MAC write secret</td>
</tr>
<tr>
<td>DWORD sequence;</td>
<td>Client's write sequence number</td>
</tr>
<tr>
<td>ARCFOUR_CTX cryptCtx;</td>
<td>Client's write encryption context</td>
</tr>
<tr>
<td>BYTE reserved[6];</td>
<td>Future expansion</td>
</tr>
<tr>
<td>BYTE random[32];</td>
<td>Client.random value</td>
</tr>
</tbody>
</table>

Stack API > SSL > Internal Members > SSL_KEYS Structure

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
SSL_KEYS_SIZE Macro

```
#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

Stack API > SSL > Internal Members > SSL_KEYS_SIZE Macro
SSL_KEY_SIZE_MACRO Macro

C

#define SSL_KEY_SIZE_MACRO (SSL_KEY_SIZE * MAX_SSL_CONNECTIONS)  // Amount of space needed by all SSL key

Description

Amount of space needed by all SSL key

Stack API > SSL > Internal Members > SSL_KEY_SIZE_MACRO
SSL_MESSAGES Enumeration

C

```c
typedef enum {
    SSL_HELLO_REQUEST = 0u,
    SSL_CLIENT_HELLO = 1u,
    SSL_ANTIQUE_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;
```

Description

Describes the types of SSL messages (handshake and alerts)

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_HELLO_REQUEST = 0u</td>
<td>HelloRequest handshake message (not currently supported)</td>
</tr>
<tr>
<td>SSL_CLIENT_HELLO = 1u</td>
<td>ClientHello handshake message</td>
</tr>
<tr>
<td>SSL_ANTIQUE_CLIENT_HELLO = 18u</td>
<td>SSLv2 ClientHello handshake message (Supported for backwards compatibility. This is an internally defined value.)</td>
</tr>
<tr>
<td>SSL_SERVER_HELLO = 2u</td>
<td>ServerHello handshake message</td>
</tr>
<tr>
<td>SSL_CERTIFICATE = 11u</td>
<td>ServerCertificate handshake message</td>
</tr>
<tr>
<td>SSL_SERVER_HELLO_DONE = 14u</td>
<td>ServerHelloDone handshake message</td>
</tr>
<tr>
<td>SSL_CLIENT_KEY_EXCHANGE = 16u</td>
<td>ClientKeyExchange handshake message</td>
</tr>
<tr>
<td>SSL_FINISHED = 20u</td>
<td>Finished handshake message</td>
</tr>
<tr>
<td>SSL_ALERT_CLOSE_NOTIFY = 0u+0x80</td>
<td>CloseNotify alert message (dummy value used internally)</td>
</tr>
<tr>
<td>SSL_ALERT_UNEXPECTED_MESSAGE = 10u+0x80</td>
<td>UnexpectedMessage alert message (dummy value used internally)</td>
</tr>
<tr>
<td>SSL_ALERT_BAD_RECORD_MAC = 20u+0x80</td>
<td>BadRecordMAC alert message (dummy value used internally)</td>
</tr>
<tr>
<td>SSL_ALERT_HANDSHAKE_FAILURE = 40u+0x80</td>
<td>HandshakeFailure alert message (dummy value used internally)</td>
</tr>
<tr>
<td>SSL_NO_MESSAGE = 0xff</td>
<td>No message is currently requested (internally used value)</td>
</tr>
</tbody>
</table>
SSL_RSA_EXPORT_WITH_ARCFOUR_40_MD5

Macro

C

#define SSL_RSA_EXPORT_WITH_ARCFOUR_40_MD5 0x0003u

Description

This is macro SSL_RSA_EXPORT_WITH_ARCFOUR_40_MD5.
SSL_RSA_WITH_ARCFOUR_128_MD5 Macro

#define SSL_RSA_WITH_ARCFOUR_128_MD5 0x0004u

Description

This is macro SSL_RSA_WITH_ARCFOUR_128_MD5.
SSL_SESSION Structure

C

typedef struct {
    BYTE sessionID[32];
    BYTE masterSecret[48];
} SSL_SESSION;

Description

Storage space for SSL Session identifiers. (The SessionID and MasterSecret)

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE sessionID[32];</td>
<td>The SSL Session ID for this session</td>
</tr>
<tr>
<td>BYTE masterSecret[48];</td>
<td>Associated Master Secret for this session</td>
</tr>
</tbody>
</table>

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
SSL_SESSION_SIZE Macro

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
SSL_SESSION_SPACE Macro

```c
#define SSL_SESSION_SPACE (SSL_SESSION_SIZE*MAX_SSL_SESSIONS)
```

Description

Amount of space needed by all SSL session

Stack API > SSL > Internal Members > SSL_SESSION_SPACE Macro
SSL_SESSION_STUB Structure

C

```c
typedef struct {
    DWORD_VAL tag;
    DWORD lastUsed;
} SSL_SESSION_STUB;
```

Description

Stub value for an SSL_SESSION. 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.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD_VAL tag;</td>
<td>Identifying tag for connection When we're a client, this is the remote IP</td>
</tr>
<tr>
<td></td>
<td>When we're a host, this is 0x00 followed by first 3 bytes of session ID</td>
</tr>
<tr>
<td></td>
<td>When this stub is free/expired, this is 0x00</td>
</tr>
<tr>
<td>DWORD lastUsed;</td>
<td>Tick count when session was last used</td>
</tr>
</tbody>
</table>

Stack API > SSL > Internal Members > SSL_SESSION_STUB Structure
```c
typedef enum {
    SSL_CLIENT,
    SSL_SERVER
} SSL_SESSION_TYPE;
```

## Description

SSL Session Type Enumeration

## Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_CLIENT</td>
<td>Local device is the SSL client</td>
</tr>
<tr>
<td>SSL_SERVER</td>
<td>Local device is the SSL host</td>
</tr>
</tbody>
</table>

[Stack API] > [SSL] > [Internal Members] > [SSL_SESSION_TYPE Enumeration]
### SSL_STUB Structure

```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;
```

### Description

Memory holder for general information associated with an SSL connections.

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
</table>

...
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD wRxBytesRem;</td>
<td>Bytes left to read in current record</td>
</tr>
<tr>
<td>WORD wRxHsBytesRem;</td>
<td>Bytes left to read in current Handshake submessage</td>
</tr>
<tr>
<td>BYTE rxProtocol;</td>
<td>Protocol for message being read</td>
</tr>
<tr>
<td>BYTE rxHSType;</td>
<td>Handshake message being received</td>
</tr>
<tr>
<td>BYTE idSession;</td>
<td>ID for associated session</td>
</tr>
<tr>
<td>BYTE idRxHash;</td>
<td>ID for MAC hash (TX needs no persistence)</td>
</tr>
<tr>
<td>DWORD_VAL dwTemp;</td>
<td>Used for state machine in RxCertificate</td>
</tr>
<tr>
<td>unsigned char bIsServer : 1;</td>
<td>We are the server</td>
</tr>
<tr>
<td>unsigned char bClientHello : 1;</td>
<td>ClientHello has been sent/received</td>
</tr>
<tr>
<td>unsigned char bServerHello : 1;</td>
<td>ServerHello has been sent/received</td>
</tr>
<tr>
<td>unsigned char bServerCertificate : 1;</td>
<td>ServerCertificate has been sent/received</td>
</tr>
<tr>
<td>unsigned char bServerHelloDone : 1;</td>
<td>ServerHelloDone has been sent/received</td>
</tr>
<tr>
<td>unsigned char bClientKeyExchange : 1;</td>
<td>ClientKeyExchange has been sent/received</td>
</tr>
<tr>
<td>unsigned char bRemoteChangeCipherSpec : 1;</td>
<td>Remote node has sent a ChangeCipherSpec message</td>
</tr>
<tr>
<td>unsigned char bRemoteFinished : 1;</td>
<td>Remote node has sent a Finished message</td>
</tr>
<tr>
<td>unsigned char bLocalChangeCipherSpec : 1;</td>
<td>We have sent a ChangeCipherSpec message</td>
</tr>
<tr>
<td>unsigned char bLocalFinished : 1;</td>
<td>We have sent a Finished message</td>
</tr>
<tr>
<td>unsigned char bExpectingMAC : 1;</td>
<td>We expect a MAC at end of message</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>unsigned char bNewSession : 1;</td>
<td>TRUE if a new session, FALSE if resuming</td>
</tr>
<tr>
<td>unsigned char bCloseNotify : 1;</td>
<td>Whether or not a CloseNotify has been sent/received</td>
</tr>
<tr>
<td>unsigned char bDone : 1;</td>
<td>TRUE if the connection is closed</td>
</tr>
<tr>
<td>unsigned char bRSAInProgress : 1;</td>
<td>TRUE when RSA op is in progress</td>
</tr>
<tr>
<td>unsigned char bKeysValid : 1;</td>
<td>TRUE if the session keys have been generated</td>
</tr>
<tr>
<td>BYTE requestedMessage;</td>
<td>Currently requested message to send, or 0xff</td>
</tr>
</tbody>
</table>
SSL_STUB_SIZE Macro

```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
SSL_STUB_SPACE Macro

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
SSL_VERSION Macro

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define SSL_VERSION (0x0300u)  // SSL version number</td>
</tr>
</tbody>
</table>

Description

SSL version number

Stack API > SSL > Internal Members > SSL_VERSION Macro
### SSL_VERSION_HI Macro

```c
#define SSL_VERSION_HI (0x03u)  // SSL version number (high byte)
```

#### Description

SSL version number (high byte)
SSL_VERSION_LO Macro

C

#define SSL_VERSION_LO (0x00u) // SSL version number (low byte)

Description

SSL version number (low byte)
**SSLBufferAlloc Function**

```c
static void SSLBufferAlloc(
    BYTE * id
);
```

**Description**

Allocates a buffer for use.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Where to store the allocated ID</td>
</tr>
</tbody>
</table>

**Returns**

id - Allocated buffer ID, or **SSL_INVALID_ID** if none available

**Side Effects**

None

**Remarks**

None
### SSLBufferFree Function

```c
static void SSLBufferFree(
    BYTE * id
);
```

#### Description

Specified buffer is released

#### Preconditions

None

#### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the buffer ID to free</td>
</tr>
</tbody>
</table>

#### Outputs

| id | SSL_INVALID_ID |

#### Side Effects

None

#### Remarks

None

---

Stack API > SSL > Internal Members > SSLBufferFree Function

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
sslBufferID Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE sslBufferID;</td>
</tr>
</tbody>
</table>

Description

Which buffer is loaded

Stack API > SSL > Internal Members > sslBufferID Variable
### SSLBufferSync Function

```
static void SSLBufferSync(
    BYTE id
);
```

### Description

Specified buffer is loaded to RAM. Only loads if necessary, and saves any current buffer before switching.

### Preconditions

None

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the buffer ID to sync to RAM</td>
</tr>
</tbody>
</table>

### Returns

None

### Side Effects

None

### Remarks

None
SSLFinishPartialRecord Macro

```c
#define SSLFinishPartialRecord(a) TCPSSLPutRecordHeader(a, NULL, TRUE)
```

**Description**

This is macro SSLFinishPartialRecord.
SSLFlushPartialRecord Macro

```c
#define SSLFlushPartialRecord(a) TCPSSLPutRecordHeader(a, NULL, FALSE)
```

Description

This is macro SSLFlushPartialRecord.
## sslHash Variable

```c
HASH_SUM sslHash;
```

### Description

Hash storage

See [Stack API](#) > [SSL](#) > [Internal Members](#) > [sslHash Variable](#)
SSLHashAlloc Function

```c
static void SSLHashAlloc(
    BYTE * id
);
```

Description

Allocates a hash for use.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Where to store the allocated ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>id - Allocated hash ID, or SSL_INVALID_ID if none available</td>
</tr>
</tbody>
</table>

Side Effects

None

Remarks

None

Stack API > SSL > Internal Members > SSLHashAlloc Function
SSLHashFree Function

C

```
static void SSLHashFree(
    BYTE * id
);
```

Description

Specified hash is released

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the hash ID to free</td>
</tr>
</tbody>
</table>

Side Effects

None

Remarks

None
## sslHashID Variable

| C | BYTE sslHashID; |

### Description

Which hash is loaded

[Stack API > SSL > Internal Members > sslHashID Variable](#)
### SSLHashSync Function

```c
static void SSLHashSync(
    BYTE id
);
```

#### Description

Specified hash is loaded to RAM. Only loads if necessary, and saves any current hash before switching.

#### Preconditions

None

#### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the hash ID to sync to RAM</td>
</tr>
</tbody>
</table>

#### Returns

None

#### Side Effects

None

#### Remarks

None
sslKeys Variable

C

SSL_KEYS sslKeys;

Description

The current SSL session
sslKeysID Variable

C
BYTE sslKeysID;

Description

Which SSL KEYS are loaded

Stack API > SSL > Internal Members > sslKeysID Variable
### SSLKeysSync Function

```c
static void SSLKeysSync(
    BYTE id
);
```

#### Description

Specified key set is loaded to RAM. Only loads if necessary, and saves any current key set before switching.

#### Preconditions

None

#### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the key set ID to sync to RAM</td>
</tr>
</tbody>
</table>

#### Returns

None

#### Side Effects

None

#### Remarks

None
SSLMACAdd Function

C

```c
void SSLMACAdd(
    BYTE* data,
    WORD len
);
```

**Description**

This is function SSLMACAdd.

[Stack API > SSL > Internal Members > SSLMACAdd Function](#)
SSLMAACBegin Function

C

```c
void SSLMAACBegin(
    BYTE* MACSecret,
    DWORD seq,
    BYTE protocol,
    WORD len
);
```

Description

This is function SSLMAACBegin.

Stack API > SSL > Internal Members > SSLMAACBegin Function

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
SSLMACCalc Function

C

```c
void SSLMACCalc(
    BYTE* MACSecret,
    BYTE* result
);
```

Description

This is function SSLMACCalc.

Stack API > SSL > Internal Members > SSLMACCalc Function
### SSLRSAOperation Function

```c
static RSA_STATUS SSLRSAOperation();
```

#### Description

Pauses connection processing until RSA calculation is complete.

#### 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` to be synchronized.

#### Returns

None

#### Side Effects

None

#### Remarks

This function exists outside of the handshaking functions so that the system does not incur the expense of resuming and suspending handshake hashes.
sslRSASStubID Variable

C

BYTE sslRSASStubID;

Description

Which stub is using RSA, if any

Stack API > SSL > Internal Members > sslRSASStubID Variable
SSLRxAlert Function

```c
static void SSLRxAlert(TCP_SOCKET hTCP);
```

**Description**

Receives an alert message and decides what to do

**Preconditions**

`sslStub` is synchronized

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to read from</td>
</tr>
</tbody>
</table>

**Returns**

None

**Side Effects**

None

**Remarks**

None
SSLRxAntiqueClientHello Function

C

```c
static void SSLRxAntiqueClientHello(
    TCP_SOCKET hTCP
);
```

Description

Receives the SSLv2 ClientHello message, initiating a new SSL session with a client

Preconditions

Handshake hasher is started, and SSL has a stub assigned.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to send the message to</td>
</tr>
</tbody>
</table>

Returns

None

Side Effects

None

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.
SSLRxCCS Function

```c
static void SSLRxCCS(
   TCP_SOCKET hTCP
);
```

Description

Receives a ChangeCipherSpec from the remote server

Preconditions

`sslStub` is synchronized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to read from</td>
</tr>
</tbody>
</table>

Returns

None

Side Effects

None

Remarks

None

Stack API > SSL > Internal Members > SSLRxCCS Function
SSLRxClientHello Function

```c
static void SSLRxClientHello(TCP_SOCKET hTCP);
```

**Description**

Receives the ClientHello message, initiating a new SSL session with a client.

**Preconditions**

Handshake hasher is started, and SSL has a stub assigned.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to send the message to</td>
</tr>
</tbody>
</table>

**Returns**

None

**Side Effects**

None

**Remarks**

None
**SSLRxClientKeyExchange Function**

```c
static void SSLRxClientKeyExchange(
    TCP_SOCKET hTCP
);
```

**Description**

Receives the ClientKeyExchange message and begins the decryption process.

**Preconditions**

- `sslStub` is synchronized and `HSStart()` has been called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to read from</td>
</tr>
</tbody>
</table>

**Returns**

None

**Side Effects**

None

**Remarks**

None
SSLRxFinished Function

C

static void SSLRxFinished(
    TCP_SOCKET hTCP
);

Description

Receives the Finished message from remote node

Preconditions

sslStub is synchronized and HSSStart() has been called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to read from</td>
</tr>
</tbody>
</table>

Returns

None

Side Effects

None

Remarks

None

Stack API > SSL > Internal Members > SSLRxFinished Function
C

```c
void SSLRxHandshake(
    TCP_SOCKET hTCP
);
```

**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**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The TCP socket to read a handshake message from</td>
</tr>
</tbody>
</table>

**Returns**

None
SSLRxRecord Function

C

| WORD SSLRxRecord( |
| TCP_SOCKET hTCP, |
| BYTE sslStubID |
| ); |

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.

Preconditions

The specified SSL stub is initialized and the TCP socket is connected.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The TCP socket from which to read</td>
</tr>
<tr>
<td>id</td>
<td>The active SSL stub ID</td>
</tr>
</tbody>
</table>

Returns

WORD indicating the number of data bytes there were decrypted but left in the stream.

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.
SSLRxServerCertificate Function

```c
static void SSLRxServerCertificate(
    TCP_SOCKET hTCP
);
```

Description

Receives ServerCertificate from the remote server, locates the public key information, and executes RSA operation.

Preconditions

`sslStub` is synchronized and `HSStart()` has been called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to read from</td>
</tr>
</tbody>
</table>

Returns

None

Side Effects

None

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.
SSLRxServerHello Function

```c
static void SSLRxServerHello(
    TCP_SOCKET hTCP
);
```

**Description**

Receives the ServerHello from the remote server

**Preconditions**

- `sslStub` is synchronized and `HSSStart()` has been called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to read from</td>
</tr>
</tbody>
</table>

**Returns**

None

**Side Effects**

None

**Remarks**

None
sslSession Variable

C

SSL_SESSION sslSession;

Description

Current session data
### sslSessionID Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE sslSessionID;</td>
</tr>
</tbody>
</table>

#### Description

Which session is loaded

[Stack API] > SSL > Internal Members > sslSessionID Variable
SSLSessionMatchID Function

C

```c
static BYTE SSLSessionMatchID(
    BYTE* SessionID
);
```

Description

Locates a cached SSL session for reuse. Syncs found session into RAM.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionID</td>
<td>the session identifier to match</td>
</tr>
</tbody>
</table>

Returns

The matched session ID, or `SSL_INVALID_ID` if not found

Side Effects

None

Remarks

None
### SSLSessionMatchIP Function

```c
static BYTE SSLSessionMatchIP(
    IP_ADDR ip
);
```

**Description**

Locates a cached SSL session for reuse

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>the host session to match</td>
</tr>
</tbody>
</table>

**Returns**

The matched session ID, or `SSL_INVALID_ID` if not found

**Side Effects**

None

**Remarks**

None
SSLSessionNew Function

C

```c
static BYTE SSLSessionNew();
```

**Description**

Finds space for a new SSL session

**Preconditions**

None

**Returns**

Allocated Session ID, or `SSL_INVALID_ID` if none available

**Side Effects**

None

**Remarks**

None
sslSessionStubs Variable

8 byte session stubs

Stack API > SSL > Internal Members > sslSessionStubs Variable

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
SSLSessionSync Function

```c
static void SSLSessionSync(
    BYTE id
);
```

Description

Specified session is loaded to RAM. Only loads if necessary, and saves any current session before switching if it has been updated.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the session ID to sync to RAM</td>
</tr>
</tbody>
</table>

Returns

None

Side Effects

None

Remarks

None

Stack API > SSL > Internal Members > SSLSessionSync Function
SSLSessionUpdated Macro

```c
#define SSLSessionUpdated sslSessionUpdated = TRUE;
```

Description

This is macro SSLSessionUpdated.

Stack API > SSL > Internal Members > SSLSessionUpdated Macro

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
sslSessionUpdated Variable

C

BOOL sslSessionUpdated;

Description

Whether or not it has been updated
SSLStartPartialRecord Function

C

```c
void SSLStartPartialRecord(
    TCP_SOCKET hTCP,
    BYTE sslStubID,
    BYTE txProtocol,
    WORD wLen
);
```

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 and SSLFinishPartialRecord must be called to write remaining data, finalize, and prepare for a new record.

Preconditions

The specified SSL stub is initialized and the TCP socket is connected.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The TCP socket with data waiting to be transmitted</td>
</tr>
<tr>
<td>id</td>
<td>The active SSL stub ID</td>
</tr>
<tr>
<td>txPortocol</td>
<td>The SSL protocol number to attach to this record</td>
</tr>
<tr>
<td>wLen</td>
<td>The length of all the data to be sent</td>
</tr>
</tbody>
</table>

Returns

None
Remarks

Partial messages do not support the current cipher spec, so this can only be used during the handshake procedure.
sslStub Variable

The current SSL stub

Description

C

SSL_STUB sslStub;

Stack API > SSL > Internal Members > sslStub Variable

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
**SSLStubAlloc Function**

```c
static BOOL SSLStubAlloc();
```

**Description**

Allocates a stub for use.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs</td>
<td>None</td>
</tr>
</tbody>
</table>

**Returns**

TRUE if stub was allocated, FALSE otherwise

**Side Effects**

None

**Remarks**

None
**SSLStubFree Function**

```c
static void SSLStubFree(
    BYTE id
);
```

**Description**

Specified stub is released

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the stub ID to free</td>
</tr>
</tbody>
</table>

**Outputs**

None

**Returns**

None

**Side Effects**

None

**Remarks**

None
sslStubID Variable

C
BYTE sslStubID;

Description

Which SSL_STUB is loaded

Stack API > SSL > Internal Members > sslStubID Variable
### SSLStubSync Function

```c
static void SSLStubSync(BYTE id);
```

**Description**

Specified stub is loaded to RAM. Only loads if necessary, and saves any current stub before switching.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the stub ID to sync to RAM</td>
</tr>
</tbody>
</table>

**Returns**

None

**Side Effects**

None

**Remarks**

None
SSLTerminate Function

C

```c
void SSLTerminate(
    BYTE sslStubId
);
```

Description

Terminates an SSL connection and releases allocated resources.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the SSL stub ID to terminate</td>
</tr>
</tbody>
</table>

Returns

None
SSLTxCCSFin Function

```c
static void SSLTxCCSFin(
    TCP_SOCKET hTCP
);
```

**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.

**Preconditions**

sslStub is synchronized, and the current session has a valid pre-master secret to use.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to write the message to</td>
</tr>
</tbody>
</table>

**Returns**

None

**Side Effects**

None

**Remarks**

None
SSLTxClientHello Function

```c
static void SSLTxClientHello(
    TCP_SOCKET hTCP
);
```

### Description

Transmits the ClientHello message to initiate a new SSL session with the server.

### Preconditions

Enough space is available in hTCP to write the entire message.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to send the message to</td>
</tr>
</tbody>
</table>

### Returns

None

### Side Effects

None

### Remarks

None

---

Stack API > SSL > Internal Members > SSLTxClientHello Function
SSLTxClientKeyExchange Function

```c
static void SSLTxClientKeyExchange(TCP_SOCKET hTCP);
```

**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.

**Preconditions**

`sslStub` 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**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to write the message to</td>
</tr>
</tbody>
</table>

**Returns**

None

**Side Effects**

None

**Remarks**

None
SSLTxMessage Function

```c
void SSLTxMessage(
    TCP_SOCKET hTCP,
    BYTE sslStubID,
    BYTE msg
);
```

Description

This function transmits a specific SSL message for handshakes and alert messages. Supported messages are listed in `SSL_MESSAGES`.

Preconditions

The specified SSL stub is initialized and the TCP socket is connected.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The TCP socket with data waiting to be transmitted</td>
</tr>
<tr>
<td>id</td>
<td>The active SSL stub ID</td>
</tr>
<tr>
<td>msg</td>
<td>One of the <code>SSL_MESSAGES</code> types to send</td>
</tr>
</tbody>
</table>

Returns

None
void SSLTxRecord(
    TCP_SOCKET hTCP,
    BYTE sslStubID,
    BYTE txProtocol
);

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The TCP socket with data waiting to be transmitted</td>
</tr>
<tr>
<td>id</td>
<td>The active SSL stub ID</td>
</tr>
<tr>
<td>txPortocol</td>
<td>The SSL protocol number to attach to this record</td>
</tr>
</tbody>
</table>

Returns

None

Stack API > SSL > Internal Members > SSLTxRecord Function
### SSLTxServerCertificate Function

```c
static void SSLTxServerCertificate(
    TCP_SOCKET hTCP
);
```

#### Description
Transmits the Certificate message with the server's specified public key certificate.

#### Preconditions
None

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to send the message to</td>
</tr>
</tbody>
</table>

#### Returns
None

#### Side Effects
None

#### 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.
SSLTxServerHello Function

```c
static void SSLTxServerHello(
    TCP_SOCKET hTCP
);
```

**Description**

Transmits the ServerHello message.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to send the message to</td>
</tr>
</tbody>
</table>

**Returns**

None

**Side Effects**

None

**Remarks**

None
SSLTxServerHelloDone Function

```c
static void SSLTxServerHelloDone(
    TCP_SOCKET hTCP
);
```

**Description**

Transmits the ServerHelloDone message.

**Preconditions**

None

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>the TCP Socket to send the message to</td>
</tr>
</tbody>
</table>

**Returns**

None

**Side Effects**

None

**Remarks**

None
## Files

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLClientSize.h</td>
<td>This is file SSLClientSize.h.</td>
</tr>
</tbody>
</table>

## Module

**SSL**

Stack API > SSL > Files
This is file SSLClientSize.h.

## Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_RSA_CLIENT_SIZE</td>
<td>Size of Encryption Buffer (must be larger than key size)</td>
</tr>
</tbody>
</table>

Stack API > SSL > Files > SSLClientSize.h
TCP

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** are opened using **TCPOpen**. 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** will be sufficient, but **TCPWasReset** may also be used for listening sockets that may turn over quickly.

To write data, call **TCPIsPutReady** to check how much space is available. Then, call any of the **TCPPut** family of functions to write data as space is available. Once complete, call **TCPFlush** 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** to determine how many bytes are ready to be retrieved. Then use the **TCPGet** family of functions to read data from the socket, and/or the **TCPFind** 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`, 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` or `GenericTCPServer` examples, or read the associated RFC.

### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP Public Members</td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td>TCP Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td>TCP Internal Members</td>
<td>Functions and variables internal to the TCP module</td>
</tr>
<tr>
<td>Functions</td>
<td>The following table lists functions in this documentation.</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WFGetTCBSize</code></td>
<td>Returns number of bytes available in TCP Control Block (TCB) so higher-layer code can determine if the number of bytes available can support the structures designated to be stored in the TCB.</td>
</tr>
</tbody>
</table>

**Stack API > TCP**

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
TCP Public Members

The following functions and variables are available to the stack application.

**Functions**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPAdjustFIFOSize</td>
<td>Adjusts the relative sizes of the RX and TX buffers.</td>
</tr>
<tr>
<td>TCPClose</td>
<td>Disconnects an open socket and destroys the socket handle, including server mode socket handles.</td>
</tr>
<tr>
<td>TCPDiscard</td>
<td>Discards any pending data in the TCP RX FIFO.</td>
</tr>
<tr>
<td>TCPDisconnect</td>
<td>Disconnects an open socket.</td>
</tr>
<tr>
<td>TCPFindArrayEx</td>
<td>Searches for a string in the TCP RX buffer.</td>
</tr>
<tr>
<td>TCPFindEx</td>
<td>Searches for a byte in the TCP RX buffer.</td>
</tr>
<tr>
<td>TCPFindROMArrayEx</td>
<td>Searches for a ROM string in the TCP RX buffer.</td>
</tr>
<tr>
<td>TCPFlush</td>
<td>Immediately transmits all pending TX data.</td>
</tr>
<tr>
<td>TCPGet</td>
<td>Retrieves a single byte to a TCP socket.</td>
</tr>
<tr>
<td>TCPGetArray</td>
<td>Reads an array of data bytes from a TCP socket's receive FIFO. The data is removed from the FIFO in the process.</td>
</tr>
<tr>
<td>TCPGetRemoteInfo</td>
<td>Obtains information about a currently open socket.</td>
</tr>
<tr>
<td>TCPGetRxFIFOFree</td>
<td>Determines how many bytes are free in the RX FIFO.</td>
</tr>
<tr>
<td>TCPGetTxFIFOFull</td>
<td>Determines how many bytes are pending in the TCP TX FIFO.</td>
</tr>
<tr>
<td>TCPIsConnected</td>
<td>Determines if a socket has an established connection.</td>
</tr>
<tr>
<td>TCPIsGetReady</td>
<td>Determines how many bytes can be read from the TCP RX buffer.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TCPIsPutReady</td>
<td>Determines how much free space is available in the TCP TX buffer.</td>
</tr>
<tr>
<td>TCPOpen</td>
<td>Opens a TCP socket for listening or as a client.</td>
</tr>
<tr>
<td>TCP Peek</td>
<td>Peaks at one byte in the TCP RX FIFO without removing it from the buffer.</td>
</tr>
<tr>
<td>TCP Peek Array</td>
<td>Reads a specified number of data bytes from the TCP RX FIFO without removing them from the buffer.</td>
</tr>
<tr>
<td>TCP Put</td>
<td>Writes a single byte to a TCP socket.</td>
</tr>
<tr>
<td>TCP Put Array</td>
<td>Writes an array from RAM to a TCP socket.</td>
</tr>
<tr>
<td>TCP Put ROM Array</td>
<td>Writes an array from ROM to a TCP socket.</td>
</tr>
<tr>
<td>TCP Put ROM String</td>
<td>Writes a null-terminated string from ROM to a TCP socket. The null-terminator is not copied to the socket.</td>
</tr>
<tr>
<td>TCP Put String</td>
<td>Writes a null-terminated string from RAM to a TCP socket. The null-terminator is not copied to the socket.</td>
</tr>
<tr>
<td>TCP RAM Copy</td>
<td>Copies data to/from various memory mediums.</td>
</tr>
<tr>
<td>TCP RAM Copy ROM</td>
<td>Copies data to/from various memory mediums.</td>
</tr>
<tr>
<td>TCP Was Reset</td>
<td>Self-clearing semaphore indicating socket reset.</td>
</tr>
</tbody>
</table>

**Macros**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_SOCKET</td>
<td>The socket is invalid or could not be opened</td>
</tr>
<tr>
<td>UNKNOWN_SOCKET</td>
<td>The socket is not known</td>
</tr>
<tr>
<td>TCP_ADJUST_GIVE_REST_TO_RX</td>
<td>Resize flag: extra bytes go to RX</td>
</tr>
<tr>
<td>TCP_ADJUST_GIVE_REST_TO_TX</td>
<td>Resize flag: extra bytes go to TX</td>
</tr>
<tr>
<td>TCP_ADJUST_PRESERVE_RX</td>
<td>Resize flag: attempt to preserve RX buffer</td>
</tr>
<tr>
<td>TCP_ADJUST_PRESERVE_TX</td>
<td>Resize flag: attempt to preserve TX buffer</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TCP_OPEN_IP_ADDRESS</strong></td>
<td>Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_IP_ADDRESS while STACK_CLIENT_MODE feature is not enabled.</td>
</tr>
<tr>
<td><strong>TCP_OPEN_NODE_INFO</strong></td>
<td>Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_NODE_INFO while STACK_CLIENT_MODE feature is not enabled.</td>
</tr>
<tr>
<td><strong>TCP_OPEN_RAM_HOST</strong></td>
<td>Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_RAM_HOST while STACK_CLIENT_MODE feature is not enabled.</td>
</tr>
<tr>
<td><strong>TCP_OPEN_ROM_HOST</strong></td>
<td>Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_ROM_HOST while STACK_CLIENT_MODE feature is not enabled.</td>
</tr>
<tr>
<td><strong>TCP_OPEN_SERVER</strong></td>
<td>Create a server socket and ignore dwRemoteHost.</td>
</tr>
<tr>
<td><strong>TCPConnect</strong></td>
<td>Alias to TCPOpen as a client.</td>
</tr>
<tr>
<td><strong>TCPFind</strong></td>
<td>Alias to TCPFindEx with no length parameter.</td>
</tr>
<tr>
<td><strong>TCPFindArray</strong></td>
<td>Alias to TCPFindArrayEx with no length parameter.</td>
</tr>
<tr>
<td><strong>TCPFindROMArray</strong></td>
<td>Alias to TCPFindROMArrayEx with no length parameter.</td>
</tr>
<tr>
<td><strong>TCPGetRxFIFOFull</strong></td>
<td>Alias to TCPIsGetReady provided for API completeness</td>
</tr>
<tr>
<td><strong>TCPGetTxFIFOFree</strong></td>
<td>Alias to TCPIsPutReady provided for API completeness</td>
</tr>
<tr>
<td><strong>TCPListen</strong></td>
<td>Alias to TCPOpen as a server.</td>
</tr>
</tbody>
</table>

**Module**

**TCP**
INVALID_SOCKET Macro

```c
#define INVALID_SOCKET (0xFE)  // The socket is invalid or could not
```

**Description**

The socket is invalid or could not be opened

---

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] [Index] [Home]
C

#define UNKNOWN_SOCKET (0xFF)  // The socket is not known

Description

The socket is not known

Stack API > TCP > Public Members > UNKNOWN_SOCKET Macro

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
TCP_ADJUST_GIVE_REST_TO_RX Macro

C

#define TCP_ADJUST_GIVE_REST_TO_RX 0x01u // Resize flag: extra byte

Description

Resize flag: extra bytes go to RX

Stack API > TCP > Public Members > TCP_ADJUST_GIVE_REST_TO_RX Macro
TCP_ADJUST_GIVE_REST_TO_TX Macro

```
#define TCP_ADJUST_GIVE_REST_TO_TX 0x02u  // Resize flag: extra byte
```

Description

Resize flag: extra bytes go to TX
TCP_ADJUST_PRESERVE_RX Macro

C

#define TCP_ADJUST_PRESERVE_RX 0x04u  // Resize flag: attempt to preserve RX buffer

Description

Resize flag: attempt to preserve RX buffer
TCP_ADJUST_PRESERVE_TX Macro

C

#define TCP_ADJUST_PRESERVE_TX 0x08u // Resize flag: attempt to preserve TX buffer

Description

Resize flag: attempt to preserve TX buffer

Stack API > TCP > Public Members > TCP_ADJUST_PRESERVE_TX Macro
TCP_OPEN_IP_ADDRESS Macro

C

#define TCP_OPEN_IP_ADDRESS You_need_to_enable_STACK_CLIENT_MODE_to_use

Description

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 Macro

C

#define TCP_OPEN_NODE_INFO You_need_to_enable_STACK_CLIENT_MODE_to_use

Description

Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_NODE_INFO while STACK_CLIENT_MODE feature is not enabled.

Stack API > TCP > Public Members > TCP_OPEN_NODE_INFO Macro

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
### TCP_OPEN_RAM_HOST Macro

```c
#define TCP_OPEN_RAM_HOST You_need_to_enable_STACK_CLIENT_MODE_to_use_
```

#### Description

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 Macro

```c
#define TCP_OPEN_ROM_HOST You_need_to_enable_STACK_CLIENT_MODE_to_use
```

**Description**

Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_ROM_HOST while STACK_CLIENT_MODE feature is not enabled.

[Stack API > TCP > Public Members > TCP_OPEN_ROM_HOST Macro](#)
TCP_OPEN_SERVER Macro

C

#define TCP_OPEN_SERVER 0u

Description

Create a server socket and ignore dwRemoteHost.
TCPAdjustFIFOSize Function

C

BOOL TCPAdjustFIFOSize(
    TCP_SOCKET hTCP,
    WORD wMinRXSize,
    WORD wMinTXSize,
    BYTE vFlags
);  

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 and TCP_ADJUST_GIVE_REST_TO_RX 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.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to be adjusted</td>
</tr>
</tbody>
</table>
### Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The FIFOs were adjusted successfully</td>
</tr>
<tr>
<td>FALSE</td>
<td>Minimum RX, Minimum TX, or flags couldn't be accommodated and therefore the socket was left unchanged.</td>
</tr>
</tbody>
</table>

### Side Effects

Any unacknowledged or untransmitted data in the TX FIFO is always deleted.

### 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.
TCPConnect Macro

```c
#define TCPConnect(remote,port) TCPOpen((DWORD)remote, TCP_OPEN_NODE_I
```

**Description**

This function is an alias to [TCPOpen](#) for client sockets. It is provided for backwards compatibility with older versions of the stack. New applications should use the [TCPOpen](#) API instead.
TCPClose Function

```c
void TCPClose(TCP_SOCKET hTCP);
```

### Description

Disconnects an open socket and destroys the socket handle, including server mode socket handles. This function performs identically to the `TCPDisconnect()` function, except that both client and server mode socket handles are relinquished to the TCP/IP stack upon return.

### Preconditions

None

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>Handle to the socket to disconnect and close.</td>
</tr>
</tbody>
</table>

### Returns

None
**TCPDiscard Function**

```c
void TCPDiscard(TCP_SOCKET hTCP);
```

**Description**

Discards any pending data in the TCP RX FIFO.

**Preconditions**

TCP is initialized.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket whose RX FIFO is to be cleared.</td>
</tr>
</tbody>
</table>

**Returns**

None
TCPDisconnect Function

```c
void TCPDisconnect(
    TCP_SOCKET hTCP
);
```

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() API instead of TCPDisconnect().

Preconditions

None

Parameters
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>Handle of the socket to disconnect.</td>
</tr>
</tbody>
</table>

**Returns**

None

**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.
TCPFind Macro

```c
#define TCPFind(a, b, c, d) TCPFindEx(a, b, c, 0, d)
```

Description

This function is an alias to `TCPFindEx` with no length parameter. It is provided for backwards compatibility with an older API.
TCPFindArray Macro

C

#define TCPFindArray(a, b, c, d, e) TCPFindArrayEx(a, b, c, d, 0, e)

Description

This function is an alias to TCPFindArrayEx with no length parameter. It is provided for backwards compatibility with an older API.
TCPFindArrayEx Function

C

```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.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to search within.</td>
</tr>
<tr>
<td>cFindArray</td>
<td>The array of bytes to find in the buffer.</td>
</tr>
<tr>
<td>wLen</td>
<td>Length of cFindArray.</td>
</tr>
<tr>
<td>wStart</td>
<td>Zero-indexed starting position within the buffer.</td>
</tr>
<tr>
<td>wSearchLen</td>
<td>Length from wStart to search in the buffer.</td>
</tr>
</tbody>
</table>
bTextCompare | TRUE for case-insensitive text search, FALSE for binary search

## Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF</td>
<td>Search array not found</td>
</tr>
<tr>
<td>Otherwise</td>
<td>Zero-indexed position of the first occurrence</td>
</tr>
</tbody>
</table>

## 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.
TCPFindEx Function

```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.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to search within.</td>
</tr>
<tr>
<td>cFind</td>
<td>The byte to find in the buffer.</td>
</tr>
<tr>
<td>wStart</td>
<td>Zero-indexed starting position within the buffer.</td>
</tr>
<tr>
<td>wSearchLen</td>
<td>Length from wStart to search in the buffer.</td>
</tr>
<tr>
<td>bTextCompare</td>
<td>TRUE for case-insensitive text search, FALSE for binary search</td>
</tr>
</tbody>
</table>
## Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF</td>
<td>Search array not found</td>
</tr>
<tr>
<td>Otherwise</td>
<td>Zero-indexed position of the first occurrence</td>
</tr>
</tbody>
</table>

## 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.
TCPFindROMArray Macro

C

#define TCPFindROMArray(a, b, c, d, e) TCPFindArray(a, (BYTE*)b, c, d, e)

Description

This function is an alias to TCPFindROMArrayEx with no length parameter. It is provided for backwards compatibility with an older API.
TCPFindROMArrayEx Function

```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.

**Preconditions**

TCP is initialized.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to search within.</td>
</tr>
<tr>
<td>cFindArray</td>
<td>The array of bytes to find in the buffer.</td>
</tr>
<tr>
<td>wLen</td>
<td>Length of cFindArray.</td>
</tr>
<tr>
<td>wStart</td>
<td>Zero-indexed starting position within the buffer.</td>
</tr>
<tr>
<td>wSearchLen</td>
<td>Length from wStart to search in the buffer.</td>
</tr>
</tbody>
</table>
Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF</td>
<td>Search array not found</td>
</tr>
<tr>
<td>Otherwise</td>
<td>Zero-indexed position of the first occurrence</td>
</tr>
</tbody>
</table>

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](#) on non-PIC18 platforms.
TCPFlush Function

C

```c
void TCPFlush(
    TCP_SOCKET hTCP
);
```

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` (default: 40ms) has elapsed.

Preconditions

TCP is initialized and the socket is connected.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket whose data is to be transmitted.</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

SSL application data is automatically flushed, so this function has no effect for SSL sockets.
TCPGet Function

C

BOOL TCPGet(
    TCP_SOCKET hTCP,
    BYTE* byte
);  

Description

Retrieves a single byte to a TCP socket.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket from which to read.</td>
</tr>
<tr>
<td>byte</td>
<td>Pointer to location in which the read byte should be stored.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>A byte was read from the buffer.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The buffer was empty, or the socket is not connected.</td>
</tr>
</tbody>
</table>

Stack API > TCP > Public Members > TCPGet Function
TCPGetArray Function

C

```c
WORD TCPGetArray(
    TCP_SOCKET hTCP,
    BYTE* buffer,
    WORD count
);
```

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket from which data is to be read.</td>
</tr>
<tr>
<td>buffer</td>
<td>Pointer to the array to store data that was read.</td>
</tr>
<tr>
<td>len</td>
<td>Number of bytes to be read.</td>
</tr>
</tbody>
</table>

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.
TCPGetRemoteInfo Function

C

```c
SOCKET_INFO* TCPGetRemoteInfo(TCP_SOCKET hTCP);
```

Description

Returns the `SOCKET_INFO` 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to check.</td>
</tr>
</tbody>
</table>

Returns

The `SOCKET_INFO` structure associated with this socket. This structure is allocated statically by the function and is valid only until the next time `TCPGetRemoteInfo()` is called.

Stack API > TCP > Public Members > TCPGetRemoteInfo Function
TCPGetRxFIFOFree Function

C

```c
WORD TCPGetRxFIFOFree(
    TCP_SOCKET hTCP
);
```

Description

Determines how many bytes are free in the RX FIFO.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to check.</td>
</tr>
</tbody>
</table>

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 family functions.
TCPGetRxFIFOFull Macro

```c
#define TCPGetRxFIFOFull(a) TCPIsGetReady(a)
```

### Description

Alias to `TCPIsGetReady` provided for API completeness.
TCPGetTxFIFOFree Macro

C

```c
#define TCPGetTxFIFOFree(a) TCPIsPutReady(a)
```

Description

Alias to `TCPIsPutReady` provided for API completeness.
TCPGetTxFIFOFull Function

C

WORD TCPGetTxFIFOFull(
    TCP_SOCKET hTCP
);

Description

Determines how many bytes are pending in the TCP TX FIFO.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to check.</td>
</tr>
</tbody>
</table>

Returns

Number of bytes pending to be flushed in the TCP TX FIFO.
TCPIsConnected Function

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 to determine when the connection is set up and ready for use. This function was historically used to check for disconnections, but TCPWasReset is now a more appropriate solution.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to check.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The socket has an established connection to a remote node.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The socket is not currently connected.</td>
</tr>
</tbody>
</table>

Remarks

A socket is said to be connected only if it is in the TCP_ESTABLISHED state. Sockets in the process of opening or closing will return FALSE.
TCPIsGetReady Function

C

WORD TCPIsGetReady(
   TCP_SOCKET hTCP
);

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to check.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes available to be read from the TCP RX buffer.
TCPIsPutReady Function

C

WORD TCPIsPutReady(
    TCP_SOCKET hTCP
);

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to check.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes available to be written in the TCP TX buffer.

Stack API > TCP > Public Members > TCPIsPutReady Function
TCPListen Macro

```c
#define TCPListen(port) TCPOpen(0, TCP_OPEN_SERVER, port, TCP_PURPOSE_
```

Description

This function is an alias to TCPOpen for server sockets. It is provided for backwards compatibility with older versions of the stack. New applications should use the TCPOpen API instead.
TCPOpen Function

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 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 TCPISSConnected returning TRUE). Server sockets ignore this destination parameter and listen 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 are statically allocated on boot, but can be claimed with this function and freed using TCPDisconnect or TCPClose (for client sockets). Server sockets can be freed using TCPClose only (calls to TCPDisconnect will return server sockets to the listening state, allowing reuse).

Preconditions

TCP is initialized.
# Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwRemoteHost</td>
<td>For client sockets only. Provide a pointer to a null-terminated string of the remote host name (ex: &quot;www.microchip.com&quot; or &quot;192.168.1.123&quot;), 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 <strong>TCPIsConnected</strong> returns TRUE. This parameter is ignored for server sockets.</td>
</tr>
<tr>
<td>vRemoteHostType</td>
<td>Any one of the following flags to identify the meaning of the dwRemoteHost parameter:</td>
</tr>
<tr>
<td></td>
<td>• <strong>TCP_OPEN_SERVER</strong> - Open a server socket and ignore the dwRemoteHost parameter.</td>
</tr>
<tr>
<td></td>
<td>• <strong>TCP_OPEN_RAM_HOST</strong> - Open a client socket and <strong>connect</strong> it to a remote host who's name is stored as a null terminated string in a RAM array. Ex: &quot;www.microchip.com&quot; or &quot;192.168.0.123&quot; (BYTE* type)</td>
</tr>
<tr>
<td></td>
<td>• <strong>TCP_OPEN_ROM_HOST</strong> - Open a client socket and <strong>connect</strong> it to a remote host who's name is stored as a null terminated string in a literal string or ROM array. Ex: &quot;www.microchip.com&quot; or &quot;192.168.0.123&quot; (ROM BYTE* type)</td>
</tr>
<tr>
<td></td>
<td>• <strong>TCP_OPEN_IP_ADDRESS</strong> - Open a client socket and <strong>connect</strong> it to a remote IP address. Ex: 0x7B01A8C0 for 192.168.1.123 (DWORD type). Note that the byte ordering is big endian.</td>
</tr>
<tr>
<td></td>
<td>• <strong>TCP_OPEN_NODE_INFO</strong> - Open a client socket and <strong>connect</strong> it to a remote node or gateway.</td>
</tr>
</tbody>
</table>
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() 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.

TCP port to listen on or connect to:
- 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.
- Server sockets - the local TCP port on which to listen for connections.

vSocketPurpose Any of the TCP_PURPOSE_ * constants defined in TCPIPConfig.h or the TCPIPConfig utility (see TCPSocketInitializer[] array).

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_SOCKET</td>
<td>No sockets of the specified type were available to be opened.</td>
</tr>
<tr>
<td>Otherwise</td>
<td>A TCP_SOCKET handle. Save this handle and use it when calling all other TCP APIs.</td>
</tr>
</tbody>
</table>
**Remarks**

This function replaces the old [TCPConnect](#) and [TCPListen](#) functions.

If [TCP_OPEN_RAM_HOST](#) or [TCP_OPEN_ROM_HOST](#) are used for the destination type, the DNS client module must also be enabled (STACK_USE_DNS must be defined in TCPIPConfig.h).

**Example**

```c
// 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 soc
skt = TCPOpen((DWORD)(PTR_BASE)&cache.remote, TCP_OPEN_NODE_INFO,
              cache.remotePort.Val, TCP_PURPOSE_DEFAULT);
```

[Stack API > TCP > Public Members > TCPOpen Function](#)
TCP Peek Function

C

```
BYTE TCPPeek(
    TCP_SOCKET hTCP,
    WORD wStart
);
```

Description

Peaks at one byte in the TCP RX FIFO without removing it from the buffer.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to peak from (read without removing from stream).</td>
</tr>
<tr>
<td>wStart</td>
<td>Zero-indexed starting position within the FIFO to peek from.</td>
</tr>
</tbody>
</table>

Remarks

Use the `TCPPeekArray()` function to read more than one byte. It will perform better than calling `TCPPeek()` in a loop.
TCPPeekArray Function

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).

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to peak from (read without removing from stream).</td>
</tr>
<tr>
<td>vBuffer</td>
<td>Destination to write the peeked data bytes.</td>
</tr>
<tr>
<td>wLen</td>
<td>Length of bytes to peak from the RX FIFO and copy to vBuffer.</td>
</tr>
<tr>
<td>wStart</td>
<td>Zero-indexed starting position within the FIFO to start peeking from.</td>
</tr>
</tbody>
</table>

Remarks

None
TCPPut Function

C

```c
BOOL TCPPut(
    TCP_SOCKET hTCP,
    BYTE byte
);
```

Description

 Writes a single byte to a TCP socket.

Preconditions

 TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to which data is to be written.</td>
</tr>
<tr>
<td>byte</td>
<td>The byte to write.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The byte was written to the transmit buffer.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The transmit buffer was full, or the socket is not connected.</td>
</tr>
</tbody>
</table>
TCPPutArray Function

C

WORD TCPPutArray(
    TCP_SOCKET hTCP,
    BYTE* Data,
    WORD Len
);

Description

Writes an array from RAM to a TCP socket.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to which data is to be written.</td>
</tr>
<tr>
<td>data</td>
<td>Pointer to the array to be written.</td>
</tr>
<tr>
<td>len</td>
<td>Number of bytes to be written.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes written to the socket. If less than len, the buffer became full or the socket is not conected.
TCPPutROMArray Function

C

WORD TCPPutROMArray(
    TCP_SOCKET hTCP,
    ROM BYTE* Data,
    WORD Len
);

Description

Writes an array from ROM to a TCP socket.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to which data is to be written.</td>
</tr>
<tr>
<td>data</td>
<td>Pointer to the array to be written.</td>
</tr>
<tr>
<td>len</td>
<td>Number of bytes to be written.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes written to the socket. If less than len, the buffer became full or the socket is not connected.

Remarks

This function is aliased to TCPPutArray on non-PIC18 platforms.
TCPPutROMString Function

C

ROM BYTE* TCPPutROMString(
    TCP_SOCKET hTCP,
    ROM BYTE* Data
);

Description

Writes a null-terminated string from ROM to a TCP socket. The null-terminator is not copied to the socket.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to which data is to be written.</td>
</tr>
<tr>
<td>data</td>
<td>Pointer to the string to be written.</td>
</tr>
</tbody>
</table>

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.

Remarks

The return value of this function differs from that of TCPPutArray. 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** on non-PIC18 platforms.
TCPPutString Function

C

```c
BYTE* TCPPutString(
    TCP_SOCKET hTCP,
    BYTE* Data
);
```

Description

Writes a null-terminated string from RAM to a TCP socket. The null-terminator is not copied to the socket.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to which data is to be written.</td>
</tr>
<tr>
<td>data</td>
<td>Pointer to the string to be written.</td>
</tr>
</tbody>
</table>

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.

Remarks

The return value of this function differs from that of TCPPutArray. 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.
### TCPRAMCopy Function

```c
static void TCPRAMCopy(
    PTR_BASE wDest,
    BYTE vDestType,
    PTR_BASE wSource,
    BYTE vSourceType,
    WORD wLength
);
```

### Description

This function copies data between memory mediums (PIC RAM, SPI RAM, and Ethernet buffer RAM).

### Preconditions

TCP is initialized.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptrDest</td>
<td>Address to write to</td>
</tr>
<tr>
<td>vDestType</td>
<td>Destination medium (TCP_PIC_RAM, TCP_ETH_RAM, TCP_SPI_RAM)</td>
</tr>
<tr>
<td>ptrSource</td>
<td>Address to copy from</td>
</tr>
<tr>
<td>vSourceType</td>
<td>Source medium (TCP_PIC_RAM, TCP_ETH_RAM, or TCP_SPI_RAM)</td>
</tr>
<tr>
<td>wLength</td>
<td>Number of bytes to copy</td>
</tr>
</tbody>
</table>

### Returns

None
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.
TCPRAMCopyROM Function

```c
static void TCPRAMCopyROM(
    PTR_BASE wDest,
    BYTE wDestType,
    ROM BYTE* wSource,
    WORD wLength
);
```

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.

Preconditions

TCP is initialized.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wDest</td>
<td>Address to write to</td>
</tr>
<tr>
<td>wDestType</td>
<td>Destination medium (TCP_PIC_RAM, TCP_ETH_RAM, TCP_SPI_RAM)</td>
</tr>
<tr>
<td>wSource</td>
<td>Address to copy from</td>
</tr>
<tr>
<td>wLength</td>
<td>Number of bytes to copy</td>
</tr>
</tbody>
</table>

Returns

None

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** on non-PIC18 platforms.
TCPWasReset Function

C

```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, 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. 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 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 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>The socket to check.</td>
</tr>
</tbody>
</table>
## Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The socket has been disconnected since the previous call.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The socket has not been disconnected since the previous call.</td>
</tr>
</tbody>
</table>

[Stack API] > [TCP] > [Public Members] > [TCPWasReset Function]

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

[Contents] | [Index] | [Home]
TCP Stack Members

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.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP_STATE</td>
<td>TCP States as defined by RFC 793</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPInit</td>
<td>Initializes the TCP module.</td>
</tr>
<tr>
<td>TCPProcess</td>
<td>Handles incoming TCP segments.</td>
</tr>
<tr>
<td>TCPTick</td>
<td>Performs periodic TCP tasks.</td>
</tr>
<tr>
<td>TCPSSLDecryptMAC</td>
<td>Decrypts and MACs data arriving via SSL.</td>
</tr>
<tr>
<td>TCPStartSSLClientEx</td>
<td>Begins an SSL client session.</td>
</tr>
</tbody>
</table>

Module

TCP

Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOCKET_INFO</td>
<td>Information about a socket</td>
</tr>
<tr>
<td>TCB</td>
<td>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)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TCB_STUB</strong></td>
<td>TCP Control Block stub data storage</td>
</tr>
</tbody>
</table>

A TCP_SOCKET is stored as a single BYTE.
SOCKET_INFO Structure

C

typedef struct {
    NODE_INFO remote;
    WORD_VAL remotePort;
} SOCKET_INFO;

Description

Information about a socket

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_INFO remote;</td>
<td>NODE_INFO structure for remote node</td>
</tr>
<tr>
<td>WORD_VAL remotePort;</td>
<td>Port number associated with remote node</td>
</tr>
</tbody>
</table>

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
TCB Structure

```c
typedef struct {
    DWORD retryInterval;
    DWORD MySEQ;
    DWORD RemoteSEQ;
    PTR_BASE txUnackedTail;
    WORD_VAL remotePort;
    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;
```

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)

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD retryInterval;</td>
<td>How long to wait before retrying transmission</td>
</tr>
<tr>
<td>Variable Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DWORD MySEQ;</td>
<td>Local sequence number</td>
</tr>
<tr>
<td>DWORD RemoteSEQ;</td>
<td>Remote sequence number</td>
</tr>
<tr>
<td>PTR_BASE txUnackedTail;</td>
<td>TX tail pointer for data that is not yet acked</td>
</tr>
<tr>
<td>WORD_VAL remotePort;</td>
<td>Remote port number</td>
</tr>
<tr>
<td>WORD localPort;</td>
<td>Local port number</td>
</tr>
<tr>
<td>WORD remoteWindow;</td>
<td>Remote window size</td>
</tr>
<tr>
<td>WORD wFutureDataSize;</td>
<td>How much out-of-order data has been received</td>
</tr>
<tr>
<td>NODE_INFO niRemoteMACIP;</td>
<td>10 bytes for MAC and IP address</td>
</tr>
<tr>
<td>DWORD dwRemoteHost;</td>
<td>RAM or ROM pointer to a hostname string (ex: “www.microchip.com”)</td>
</tr>
<tr>
<td>SHORT sHoleSize;</td>
<td>Size of the hole, or -1 for none exists. (0 indicates hole has just been filled)</td>
</tr>
<tr>
<td>unsigned char bFINSent : 1;</td>
<td>A FIN has been sent</td>
</tr>
<tr>
<td>unsigned char bSYNSent : 1;</td>
<td>A SYN has been sent</td>
</tr>
<tr>
<td>unsigned char bRemoteHostIsROM : 1;</td>
<td>Remote host is stored in ROM</td>
</tr>
<tr>
<td>unsigned char bRXNoneACKed1 : 1;</td>
<td>A duplicate ACK was likely received</td>
</tr>
<tr>
<td>unsigned char bRXNoneACKed2 : 1;</td>
<td>A second duplicate ACK was likely received</td>
</tr>
<tr>
<td>unsigned char filler : 3;</td>
<td>future use</td>
</tr>
<tr>
<td>WORD wRemoteMSS;</td>
<td>Maximum Segment Size option advertised by the remote node during initial handshaking</td>
</tr>
<tr>
<td>WORD localSSLPort;</td>
<td>Local SSL port number (for listening sockets)</td>
</tr>
<tr>
<td>BYTE retryCount;</td>
<td>Counter for transmission retries</td>
</tr>
<tr>
<td>BYTE vSocketPurpose;</td>
<td>Purpose of socket (as defined in TCPIPCfg.h)</td>
</tr>
</tbody>
</table>
```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;
```

**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.
Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTR_BASE bufferTxStart;</td>
<td>First byte of TX buffer</td>
</tr>
<tr>
<td>PTR_BASE bufferRxStart;</td>
<td>First byte of RX buffer. TX buffer ends 1 byte prior</td>
</tr>
<tr>
<td>PTR_BASE bufferEnd;</td>
<td>Last byte of RX buffer</td>
</tr>
<tr>
<td>PTR_BASE txHead;</td>
<td>Head pointer for TX</td>
</tr>
<tr>
<td>PTR_BASE txTail;</td>
<td>Tail pointer for TX</td>
</tr>
<tr>
<td>PTR_BASE rxHead;</td>
<td>Head pointer for RX</td>
</tr>
<tr>
<td>PTR_BASE rxTail;</td>
<td>Tail pointer for RX</td>
</tr>
<tr>
<td>DWORD eventTime;</td>
<td>Packet retransmissions, state changes</td>
</tr>
<tr>
<td>WORD eventTime2;</td>
<td>Window updates, automatic transmission</td>
</tr>
<tr>
<td>WORD delayedACKTime;</td>
<td>Delayed Acknowledgement timer</td>
</tr>
<tr>
<td>WORD closeWaitTime;</td>
<td>TCP_CLOSE_WAIT timeout timer</td>
</tr>
<tr>
<td>TCP_STATE smState;</td>
<td>State of this socket</td>
</tr>
<tr>
<td>unsigned char vUnackedKeepalives : 3;</td>
<td>Count of how many keepalives have been sent with no response</td>
</tr>
<tr>
<td>unsigned char bServer : 1;</td>
<td>Socket should return to listening state when closed</td>
</tr>
<tr>
<td>unsigned char bTimerEnabled : 1;</td>
<td>Timer is enabled</td>
</tr>
<tr>
<td>unsigned char bTimer2Enabled : 1;</td>
<td>Second timer is enabled</td>
</tr>
<tr>
<td>unsigned char bDelayedACKTimerEnabled : 1;</td>
<td>DelayedACK timer is enabled</td>
</tr>
<tr>
<td>unsigned char</td>
<td>A segment has been received</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bOneSegmentReceived</td>
<td>1;</td>
</tr>
<tr>
<td>unsigned char bHalfFullFlush</td>
<td>Flush is for being half full</td>
</tr>
<tr>
<td>unsigned char bTXASAP</td>
<td>Transmit as soon as possible (for Flush)</td>
</tr>
<tr>
<td>unsigned char bTXASAPWithoutTimerReset</td>
<td>Transmit as soon as possible (for Flush), but do not reset retransmission timers</td>
</tr>
<tr>
<td>unsigned char bTXFIN</td>
<td>FIN needs to be transmitted</td>
</tr>
<tr>
<td>unsigned char bSocketReset</td>
<td>Socket has been reset (self-clearing semaphore)</td>
</tr>
<tr>
<td>unsigned char bSSLHandshaking</td>
<td>Socket is in an SSL handshake</td>
</tr>
<tr>
<td>unsigned char filler</td>
<td>Future expansion</td>
</tr>
<tr>
<td>WORD_VAL remoteHash</td>
<td>Consists of remoteIP, remotePort, localPort for connected sockets. It is a localPort number only for listening server sockets.</td>
</tr>
<tr>
<td>PTR_BASE sslTxHead</td>
<td>Position of data being written in next SSL application record Also serves as cache of localSSLPort when smState = TCP_LISTENING</td>
</tr>
<tr>
<td>PTR_BASE sslRxHead</td>
<td>Position of incoming data not yet handled by SSL</td>
</tr>
<tr>
<td>BYTE sslStubID</td>
<td>Which <a href="#">sslStub</a> is associated with this connection</td>
</tr>
<tr>
<td>BYTE sslReqMessage</td>
<td>Currently requested SSL message</td>
</tr>
<tr>
<td>BYTE vMemoryMedium</td>
<td>Which memory medium the TCB is actually stored</td>
</tr>
</tbody>
</table>
TCP_SOCKET Type

C

typedef BYTE TCP_SOCKET;

Description

A TCP_SOCKET is stored as a single BYTE
**C**

```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;
```

**Description**

TCP States as defined by RFC 793

**Members**

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP_GET_DNS_MODULE</td>
<td>Special state for TCP client mode sockets</td>
</tr>
<tr>
<td>TCP_DNS_RESOLVE</td>
<td>Special state for TCP client mode sockets</td>
</tr>
<tr>
<td>TCP_GATEWAY_SEND_ARP</td>
<td>Special state for TCP client mode sockets</td>
</tr>
<tr>
<td>TCP_GATEWAY_GET_ARP</td>
<td>Special state for TCP client mode sockets</td>
</tr>
<tr>
<td>TCP_LISTEN</td>
<td>Socket is listening for connections</td>
</tr>
<tr>
<td>TCP_SYN_SENT</td>
<td>A SYN has been sent, awaiting an SYN+ACK</td>
</tr>
<tr>
<td>TCP_SYN_RECEIVED</td>
<td>A SYN has been received, awaiting an ACK</td>
</tr>
<tr>
<td>TCP ESTABLISHED</td>
<td>Socket is connected and connection is established</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>TCP FIN WAIT 1</td>
<td>FIN WAIT state 1</td>
</tr>
<tr>
<td>TCP FIN WAIT 2</td>
<td>FIN WAIT state 2</td>
</tr>
<tr>
<td>TCP CLOSING</td>
<td>Socket is closing TCP TIME WAIT, state is not implemented</td>
</tr>
<tr>
<td>TCP CLOSE WAIT</td>
<td>Waiting to close the socket</td>
</tr>
<tr>
<td>TCP LAST ACK</td>
<td>The final ACK has been sent</td>
</tr>
<tr>
<td>TCP CLOSED</td>
<td>Socket is idle and unallocated</td>
</tr>
<tr>
<td>TCP CLOSED BUT RESERVED</td>
<td>Special state for TCP client mode sockets. Socket is idle, but still allocated pending application closure of the handle.</td>
</tr>
</tbody>
</table>
**TCPInit Function**

```c
void TCPInit();
```

### 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.

### Preconditions

None

### Returns

None

### Remarks

This function is called only one during lifetime of the application.
TCPProcess Function

C

```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 for further processing.

Preconditions

TCP is initialized and a TCP segment is ready in the MAC buffer.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote</td>
<td>Remote NODE_INFO structure</td>
</tr>
<tr>
<td>localIP</td>
<td>This stack's IP address (for header checking)</td>
</tr>
<tr>
<td>len</td>
<td>Total length of the waiting TCP segment</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>the segment was properly handled.</td>
</tr>
<tr>
<td>FALSE</td>
<td>otherwise</td>
</tr>
</tbody>
</table>

Stack API > TCP > Stack Members > TCPProcess Function
TCPTick Function

C

void TCPTick();

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.

Returns

None

Stack API > TCP > Stack Members > TCPTick Function
TCPSSLDecryptMAC Function

```c
void TCPSSLDecryptMAC(
    TCP_SOCKET hTCP,
    ARCFOUR_CTX* ctx,
    WORD len
);
```

**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.

**Preconditions**

TCP is initialized, hTCP is connected, and ctx's Sbox is loaded.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to decrypt in</td>
</tr>
<tr>
<td>ctx</td>
<td>ARCFOUR encryption context to use</td>
</tr>
<tr>
<td>len</td>
<td>Number of bytes to crypt</td>
</tr>
<tr>
<td>inPlace</td>
<td>TRUE to write back in place, FALSE to write at end of currently visible data.</td>
</tr>
</tbody>
</table>

**Returns**

None

**Remarks**

This function should never be called by an application. It is used only by
the SSL module itself.
## TCPStartSSLClientEx Function

```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.

### Preconditions

TCP is initialized and hTCP is already connected.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTCP</td>
<td>TCP connection to secure</td>
</tr>
<tr>
<td>host</td>
<td>Expected host name on certificate (currently ignored)</td>
</tr>
<tr>
<td>buffer</td>
<td>Buffer for supplementary data return</td>
</tr>
<tr>
<td>suppDataType</td>
<td>Type of supplementary data to copy</td>
</tr>
</tbody>
</table>

### Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>an SSL connection was initiated</td>
</tr>
<tr>
<td>FALSE</td>
<td>Insufficient SSL resources (stubs) were available</td>
</tr>
</tbody>
</table>
Remarks

The host parameter is currently ignored and is not validated.
TCP Internal Members

The following functions and variables are designated as internal to the TCP module.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseSocket</td>
<td>Closes a TCP socket.</td>
</tr>
<tr>
<td>FindMatchingSocket</td>
<td>Finds a suitable socket for a TCP segment.</td>
</tr>
<tr>
<td>HandleTCPSeg</td>
<td>Processes an incoming TCP segment.</td>
</tr>
<tr>
<td>SendTCP</td>
<td>Transmits a TCP segment.</td>
</tr>
<tr>
<td>SwapTCPHeader</td>
<td>Swaps endian-ness of a TCP header.</td>
</tr>
<tr>
<td>SyncTCB</td>
<td>Flushes MyTCB cache and loads up the specified TCB. Does nothing on cache hit.</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>Acknowledge Flag as defined in RFC</td>
</tr>
<tr>
<td>FIN</td>
<td>FIN Flag as defined in RFC</td>
</tr>
<tr>
<td>LOCAL_PORT_END_NUMBER</td>
<td>End port for client sockets</td>
</tr>
<tr>
<td>LOCAL_PORT_START_NUMBER</td>
<td>Starting port for client sockets</td>
</tr>
<tr>
<td>PSH</td>
<td>Push Flag as defined in RFC</td>
</tr>
<tr>
<td>RST</td>
<td>Reset Flag as defined in RFC</td>
</tr>
<tr>
<td>SENDTCP_KEEP_ALIVE</td>
<td>Instead of transmitting normal data, a garbage octet is transmitted according to RFC 1122 section 4.2.3.6</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SENDTCP_RESET_TIMERS</td>
<td>Indicates if this packet is a retransmission (no reset) or a new packet (reset required)</td>
</tr>
<tr>
<td>SYN</td>
<td>SYN Flag as defined in RFC</td>
</tr>
<tr>
<td>SyncTCBStub</td>
<td>Flushes MyTCBStub cache and loads up the specified TCB_STUB. Does nothing on cache hit.</td>
</tr>
<tr>
<td>TCP_AUTO_TRANSMIT_TIMEOUT_VAL</td>
<td>Timeout before automatically transmitting unflushed data</td>
</tr>
<tr>
<td>TCP_WINDOW_UPDATE_TIMEOUT_VAL</td>
<td>Timeout before automatically transmitting a window update due to a TCPGet() or TCPGetArray() function call</td>
</tr>
<tr>
<td>TCP_CLOSE_WAIT_TIMEOUT</td>
<td>Timeout for the CLOSE_WAIT state</td>
</tr>
<tr>
<td>TCP_DELAYED_ACK_TIMEOUT</td>
<td>Timeout for delayed-acknowledgement algorithm</td>
</tr>
<tr>
<td>TCP_FIN_WAIT_2_TIMEOUT</td>
<td>Timeout for FIN WAIT 2 state</td>
</tr>
<tr>
<td>TCP_KEEP_ALIVE_TIMEOUT</td>
<td>Timeout for keep-alive messages when no traffic is sent</td>
</tr>
<tr>
<td>TCP_MAX_RETRIES</td>
<td>Maximum number of retransmission attempts</td>
</tr>
<tr>
<td>TCP_MAX_SEG_SIZE_RX</td>
<td>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).</td>
</tr>
<tr>
<td>TCP_MAX_SEG_SIZE_TX</td>
<td>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 unhandiably large MSS (ex: &gt; Ethernet)</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP_MAX_SYN_RETRIES</td>
<td>Smaller than all other retries to reduce SYN flood DoS duration</td>
</tr>
<tr>
<td>TCP_MAX_UNACKED_KEEP_ALIVES</td>
<td>Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection</td>
</tr>
<tr>
<td>TCP_OPTIMIZE_FOR_SIZE</td>
<td>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</td>
</tr>
<tr>
<td>TCP_OPTIONS_END_OF_LIST</td>
<td>End of List TCP Option Flag</td>
</tr>
<tr>
<td>TCP_OPTIONS_MAX_SEG_SIZE</td>
<td>Maximum segment size TCP flag</td>
</tr>
<tr>
<td>TCP_OPTIONS_NO_OP</td>
<td>No Op TCP Option</td>
</tr>
<tr>
<td>TCP_SOCKET_COUNT</td>
<td>Determines the number of defined TCP sockets</td>
</tr>
<tr>
<td>TCP_START_TIMEOUT_VAL</td>
<td>Timeout to retransmit unacked data</td>
</tr>
<tr>
<td>TCP_SYN_QUEUE_MAX_ENTRIES</td>
<td>Number of TCP RX SYN packets to save if they cannot be serviced immediately</td>
</tr>
<tr>
<td>TCP_SYN_QUEUE_TIMEOUT</td>
<td>Timeout for when SYN queue entries are deleted if unserviceable</td>
</tr>
<tr>
<td>URG</td>
<td>Urgent Flag as defined in RFC</td>
</tr>
</tbody>
</table>

**Module**

**TCP**
## Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TCP_HEADER</strong></td>
<td>TCP Header Data Structure</td>
</tr>
<tr>
<td><strong>TCP_OPTIONS</strong></td>
<td>TCP Options data structure</td>
</tr>
<tr>
<td><strong>TCP_SYN_QUEUE</strong></td>
<td>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</td>
</tr>
</tbody>
</table>

## Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>hCurrentTCP</strong></td>
<td>Current TCP socket</td>
</tr>
<tr>
<td><strong>MyTCB</strong></td>
<td>Currently loaded TCB</td>
</tr>
<tr>
<td><strong>MyTCBStub</strong></td>
<td>Alias to current TCP stub.</td>
</tr>
<tr>
<td><strong>SYNQueue</strong></td>
<td>Array of saved incoming SYN requests that need to be serviced later</td>
</tr>
<tr>
<td><strong>TCBStubs</strong></td>
<td>This is variable TCBStubs.</td>
</tr>
</tbody>
</table>

*Stack API > TCP > Internal Members*
ACK Macro

```c
#define ACK (0x10) // Acknowledge Flag as defined in RFC
```

Description

Acknowledge Flag as defined in RFC
CloseSocket Function

```c
static void CloseSocket();
```

**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.

**Returns**

None
FIN Macro

C

#define FIN (0x01)     // FIN Flag as defined in RFC

Description

FIN Flag as defined in RFC

Stack API > TCP > Internal Members > FIN Macro
FindMatchingSocket Function

```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` and the associated `MyTCBStub` and MyTCB are loaded. Otherwise, `INVALID_SOCKET` is placed in `hCurrentTCP`.

**Preconditions**

TCP is initialized.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>TCP header to be matched against</td>
</tr>
<tr>
<td>remote</td>
<td>The remote node who sent this header</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>A match was found and is loaded in <code>hCurrentTCP</code></td>
</tr>
<tr>
<td>FALSE</td>
<td>No suitable socket was found and <code>hCurrentTCP</code> is <code>INVALID_SOCKET</code></td>
</tr>
</tbody>
</table>
HandleTCPSeg Function

C

```c
static void HandleTCPSeg(
    TCP_HEADER* h,
    WORD len
);
```

**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**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>The TCP header for this packet</td>
</tr>
<tr>
<td>len</td>
<td>The total buffer length of this segment</td>
</tr>
</tbody>
</table>

**Returns**

None
hCurrentTCP Variable

C

TCP_SOCKET hCurrentTCP = INVALID_SOCKET;

Description

Current TCP socket

Stack API > TCP > Internal Members > hCurrentTCP Variable
LOCAL_PORT_END_NUMBER Macro

C

#define LOCAL_PORT_END_NUMBER (5000u)

Description

End port for client sockets
LOCAL_PORT_START_NUMBER Macro

C

#define LOCAL_PORT_START_NUMBER (1024u)

Description

Starting port for client sockets

Stack API > TCP > Internal Members > LOCAL_PORT_START_NUMBER Macro
MyTCB Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCB MyTCB;</td>
</tr>
</tbody>
</table>

Description

Currently loaded TCB
MyTCBStub Variable

```c
TCB_STUB MyTCBStub;
```

**Description**

Alias to current TCP stub.
PSH Macro

C

#define PSH (0x08)  // Push Flag as defined in RFC

Description

Push Flag as defined in RFC

Stack API > TCP > Internal Members > PSH Macro
RST Macro

```c
#define RST (0x04)  // Reset Flag as defined in RFC
```

Description

Reset Flag as defined in RFC
SendTCP Function

C

```c
static void SendTCP(
    BYTE vTCPFlags,
    BYTE vSendFlags
);
```

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vTCPFlags</td>
<td>Additional TCP flags to include</td>
</tr>
<tr>
<td>vSendFlags</td>
<td>Any combinations of SENDTCP_* constants to modify the transmit behavior or contents.</td>
</tr>
</tbody>
</table>

Returns

None

Stack API > TCP > Internal Members > SendTCP Function
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
#define SENDTCP_RESET_TIMERS 0x01

## Description

Indicates if this packet is a retransmission (no reset) or a new packet (reset required)
SwapTCPHeader Function

```
C

static void SwapTCPHeader(TCP_HEADER* header);
```

Description

This function swaps the endian-ness of a given TCP header for comparison.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>The TCP header that is to be swapped</td>
</tr>
</tbody>
</table>

Returns

None
SYN Macro

```c
#define SYN (0x02)  // SYN Flag as defined in RFC
```

Description

SYN Flag as defined in RFC
SyncTCB Function

C

static void SyncTCB();

Description

Flushes MyTCB cache and loads up the specified TCB. Does nothing on cache hit.
SyncTCBStub Macro

C
#define SyncTCBStub(a) hCurrentTCP = (a)

Description

Flushes MyTCBStub cache and loads up the specified TCB_STUB. Does nothing on cache hit.

Stack API > TCP > Internal Members > SyncTCBStub Macro
SYNQueue Variable

```c
TCP_SYN_QUEUE SYNQueue[TCP_SYN_QUEUE_MAX_ENTRIES];
```

**Description**

Array of saved incoming SYN requests that need to be serviced later
TCBStubs Variable

C

TCB_STUB  TCBStubs[TCP_SOCKET_COUNT];

Description

This is variable TCBStubs.
TCP_AUTO_TRANSMIT_TIMEOUT_VAL Macro

```c
#define TCP_AUTO_TRANSMIT_TIMEOUT_VAL (TICK_SECOND/25ull) // Timeout before automatically transmitting unflushed data
```

**Description**

Timeout before automatically transmitting unflushed data
TCP_WINDOW_UPDATE_TIMEOUT_VAL Macro

```c
#define TCP_WINDOW_UPDATE_TIMEOUT_VAL (TICK_SECOND/5ull) // Timeout
```

**Description**

Timeout before automatically transmitting a window update due to a `TCPGet()` or `TCPGetArray()` function call.
#define TCP_CLOSE_WAIT_TIMEOUT ((DWORD)TICK_SECOND/5) // Timeout for the CLOSE_WAIT state
#define TCP_DELAYED_ACK_TIMEOUT ((DWORD)TICK_SECOND/10) // Timeout

Description

Timeout for delayed-acknowledgement algorithm
TCP_FIN_WAIT_2_TIMEOUT Macro

```c
#define TCP_FIN_WAIT_2_TIMEOUT ((DWORD)TICK_SECOND*5) // Timeout for FIN WAIT 2 state
```

Description

Timeout for FIN WAIT 2 state
TCP_HEADER Structure

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;

Description

TCP Header Data Structure

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD SourcePort;</td>
<td>Local port number</td>
</tr>
<tr>
<td>WORD DestPort;</td>
<td>Remote port number</td>
</tr>
<tr>
<td>DWORD SeqNumber;</td>
<td>Local sequence number</td>
</tr>
<tr>
<td>DWORD AckNumber;</td>
<td>Acknowledging remote sequence number</td>
</tr>
<tr>
<td>Description</td>
<td>C Code</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Data offset flags nibble</td>
<td><code>struct { unsigned char Reserved3 : 4; unsigned char Val : 4; } DataOffset;</code></td>
</tr>
<tr>
<td>TCP Flags as defined in RFC</td>
<td><code>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;</code></td>
</tr>
<tr>
<td>Local free RX buffer window</td>
<td><code>WORD Window;</code></td>
</tr>
<tr>
<td>Data payload checksum</td>
<td><code>WORD Checksum;</code></td>
</tr>
<tr>
<td>Urgent pointer</td>
<td><code>WORD UrgentPointer;</code></td>
</tr>
</tbody>
</table>
TCP_KEEP_ALIVE_TIMEOUT Macro

```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.
#TCP_MAX_RETRIES Macro

```c
#define TCP_MAX_RETRIES (5u) // Maximum number of retransmission attempts
```

##Description

Maximum number of retransmission attempts
TCP_MAX_SEG_SIZE_RX Macro

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).
TCP_MAX_SEG_SIZE_TX Macro

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 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.
#define TCP_MAX_SYN_RETRIES (2u) // Smaller than all other retries

Description

Smaller than all other retries to reduce SYN flood DoS duration
**TCP_MAX_UNACKED_KEEP_ALIVES Macro**

```c
#define TCP_MAX_UNACKED_KEEP_ALIVES (6u)  // Maximum
```

**Description**

Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection.
TCP_OPTIMIZE_FOR_SIZE Macro

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.
TCP_OPTIONS Structure

C

```c
typedef struct {
    BYTE Kind;
    BYTE Length;
    WORD_VAL MaxSegSize;
} TCP_OPTIONS;
```

Description

TCP Options data structure

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE Kind;</td>
<td>Type of option</td>
</tr>
<tr>
<td>BYTE Length;</td>
<td>Length</td>
</tr>
<tr>
<td>WORD_VAL MaxSegSize;</td>
<td>Maximum segment size</td>
</tr>
</tbody>
</table>

Stack API > TCP > Internal Members > TCP_OPTIONS Structure
#define TCP_OPTIONS_END_OF_LIST (0x00u) // End of List TCP Option Flag

Description

End of List TCP Option Flag
TCP_OPTIONS_MAX_SEG_SIZE Macro

C

#define TCP_OPTIONS_MAX_SEG_SIZE (0x02u) // Maximum segment size

Description

Maximum segment size TCP flag
TCP_OPTIONS_NO_OP Macro

```c
#define TCP_OPTIONS_NO_OP (0x01u) // No Op TCP Option
```

Description

No Op TCP Option

Stack API > TCP > Internal Members > TCP_OPTIONS_NO_OP Macro
TCP_SOCKET_COUNT Macro

C

```c
#define TCP_SOCKET_COUNT (sizeof(TCPSocketInitializer)/sizeof(TCPSocketInitializer[0]))
```

Description

Determines the number of defined TCP sockets
TCP_START_TIMEOUT_VAL Macro

```
#define TCP_START_TIMEOUT_VAL ((DWORD)TICK_SECOND*1)  // Timeout to retransmit unacked data
```

Description

Timeout to retransmit unacked data
# TCP_SYN_QUEUE Structure

**C**

```c
typedef struct {
    NODE_INFO niSourceAddress;
    WORD wSourcePort;
    DWORD dwSourceSEQ;
    WORD wDestPort;
    WORD wTimestamp;
} TCP_SYN_QUEUE;
```

**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.

## Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_INFO niSourceAddress;</td>
<td>Remote IP address and MAC address</td>
</tr>
<tr>
<td>WORD wSourcePort;</td>
<td>Remote TCP port number that the response SYN needs to be sent to</td>
</tr>
<tr>
<td>DWORD dwSourceSEQ;</td>
<td>Remote TCP SEQuence number that must be ACKnowledged when we send our response SYN</td>
</tr>
<tr>
<td>WORD wDestPort;</td>
<td>Local TCP port which the original SYN was destined for</td>
</tr>
<tr>
<td>WORD wTimestamp;</td>
<td>Timer to expire old SYN packets that can't be serviced at all</td>
</tr>
</tbody>
</table>

[Stack API] > [TCP] > [Internal Members] > [TCP_SYN_QUEUE Structure]
TCP_SYN_QUEUE_MAX_ENTRIES Macro

```
#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
TCP_SYN_QUEUE_TIMEOUT Macro

```c
#define TCP_SYN_QUEUE_TIMEOUT ((DWORD)TICK_SECOND*3)  // Timeout for
```

**Description**

Timeout for when SYN queue entries are deleted if unserviceable
# URG Macro

```c
#define URG (0x20) // Urgent Flag as defined in RFC
```

## Description

Urgent Flag as defined in RFC

Stack API > TCP > Internal Members > URG Macro
## Functions

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFGetTCBSIZE</td>
<td>Returns number of bytes available in TCP Control Block (TCB) so higher-layer code can determine if the number of bytes available can support the structures designated to be stored in the TCB.</td>
</tr>
</tbody>
</table>

### Module

**TCP**

Stack API > TCP > Functions
WFGetTCBSize Function

C

UINT16 WFGetTCBSize();

Description

Returns number of bytes available in TCP Control Block (TCB) so higher-layer code can determine if the number of bytes available can support the structures designated to be stored in the TCB.

Preconditions

None

Returns

Number of bytes in the TCB

Side Effects

None

Remarks

When running with WiFi the TCB is contained in the Scratch Memory on the MRF24W.

Stack API > TCP > Functions > WFGetTCBSize Function
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 TFTPGetFileOpenReady() returns FALSE, alternate calling StackTask() and that decision block function.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFTP Public Members</td>
<td>Functions and variables accessible by the stack application.</td>
</tr>
<tr>
<td>TFTP Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td>TFTP Internal Members</td>
<td>Functions and variables internal to the TFTP module</td>
</tr>
</tbody>
</table>
The following functions and variables are available to the stack application.

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFTP_ACCESS_ERROR</td>
<td>Standard error codes as defined by TFTP spec. Use to decode value returned by TFTPGetError().</td>
</tr>
<tr>
<td>_TFTP_ACCESS_ERROR</td>
<td>Standard error codes as defined by TFTP spec. Use to decode value returned by TFTPGetError().</td>
</tr>
<tr>
<td>TFTP_FILE_MODE</td>
<td>File open mode as used by TFTPFileOpen().</td>
</tr>
<tr>
<td>_TFTP_FILE_MODE</td>
<td>File open mode as used by TFTPFileOpen().</td>
</tr>
<tr>
<td>TFTP_RESULT</td>
<td>Enum. of results returned by most of the TFTP functions.</td>
</tr>
<tr>
<td>_TFTP_RESULT</td>
<td>Enum. of results returned by most of the TFTP functions.</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFTPCloseFile</td>
<td>Sends file closing messages.</td>
</tr>
<tr>
<td>TFTPGet</td>
<td>Gets a data byte from data that was read.</td>
</tr>
<tr>
<td>TFTPIsFileClosed</td>
<td>Determines if the file was closed.</td>
</tr>
<tr>
<td>TFTPIsFileOpened</td>
<td>Determines if file has been opened.</td>
</tr>
<tr>
<td>TFTPIsGetReady</td>
<td>Determines if a data block is ready to be read.</td>
</tr>
<tr>
<td>TFTPIsOpened</td>
<td>Determines if the TFTP connection is open.</td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TFTPIsPutReady</td>
<td>Determines if data can be written to a file.</td>
</tr>
<tr>
<td>TFTPOpen</td>
<td>Initializes TFTP module.</td>
</tr>
<tr>
<td>TFTPOpenFile</td>
<td>Prepares and sends TFTP file name and mode packet.</td>
</tr>
<tr>
<td>TFTPOpenROMFile</td>
<td>PIC18 ROM argument implementation of TFTPOpenFile</td>
</tr>
<tr>
<td>TFTPut</td>
<td>Write a byte to a file.</td>
</tr>
<tr>
<td>TFTPGetUploadStatus</td>
<td>Returns the TFTP file upload status started by calling the</td>
</tr>
<tr>
<td></td>
<td>TFTPUploadRAMFileToHost() or</td>
</tr>
<tr>
<td></td>
<td>TFTPUploadFragmentedRAMFileToHost() functions.</td>
</tr>
<tr>
<td>TFTPUploadFragmentedRAMFileToHost</td>
<td>Uploads an random, potentially non-continuous, array of RAM bytes as a file to</td>
</tr>
<tr>
<td></td>
<td>a remote TFTP server.</td>
</tr>
<tr>
<td>TFTPUploadRAMFileToHost</td>
<td>Uploads a contiguous array of RAM bytes as a file to a remote TFTP server.</td>
</tr>
</tbody>
</table>

**Macros**

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>TFTPClose</td>
<td>Macro: void TFTPClose(void)</td>
</tr>
<tr>
<td></td>
<td>Closes TFTP client socket.</td>
</tr>
<tr>
<td>TFTPGetError</td>
<td>Macro: WORD TFTPGetError(void)</td>
</tr>
<tr>
<td></td>
<td>Returns previously saved error code.</td>
</tr>
<tr>
<td>TFTPIsFileOpenReady</td>
<td>Macro: BOOL TFTPIsFileOpenReady()</td>
</tr>
<tr>
<td></td>
<td>Checks to see if it is okay to send TFTP open request to remote server.</td>
</tr>
<tr>
<td>TFTP UPLOAD COMPLETE</td>
<td>Status codes for TFTPGetUploadStatus function. Zero means upload success,</td>
</tr>
<tr>
<td></td>
<td>means working and &lt;0 means fatal error.</td>
</tr>
<tr>
<td>TFTP UPLOAD CONNECT</td>
<td>This is macro TFTP UPLOAD_CONNECT</td>
</tr>
<tr>
<td>TFTP UPLOAD CONNECT TIMEOUT</td>
<td>This is macro TFTP UPLOAD_CONNECT_TIMEOUT</td>
</tr>
</tbody>
</table>
## Module

**TFTP**

### Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFTP_CHUNK_DESCRIPTOR</td>
<td>This is type TFTP_CHUNK_DESCRIPTOR.</td>
</tr>
</tbody>
</table>

Stack API > TFTP > Public Members

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
TFTPClose Macro

C

#define TFTPClose(void) UDPClose(_tftpSocket)

Description

Macro: void TFTPClose(void)
Closes TFTP client socket.

Preconditions

TFTPOpen is already called and TFTPIsOpened() returned TFTP_OK.

Returns

None

Side Effects

None

Remarks

Once closed, application must do TFTPOpen 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.
**TFTPCloseFile Function**

```c
void TFTPCloseFile();
```

**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.

**Preconditions**

- `TFTPOpenFile()` was called and `TFTPIsFileOpened()` had returned with TFTP_OK.

**Returns**

None

**Side Effects**

None

**Remarks**

- `TFTPIsFileClosed()` must be called to confirm if file was really closed.
TFTPGet Function

C

BYTE TFTPGet();

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.

Preconditions

TFTPOpenFile() is called with TFTP_FILE_MODE_READ and TFTPIsGetReady() = TRUE

Returns

data byte as received from remote server.

Side Effects

None

Remarks

Use this function to read file from server.
#define TFTPGetError (_tftpError)
The function `TFTPIsFileClosed()` is used to check if a file is closed.

### 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.

### Preconditions

- `TFTPCloseFile()` is already called.

### 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()`
- **TFTP_NOT_READY** if file is not closed yet.

### Side Effects

- None

### Remarks

- None
TFTPIsFileOpened Function

C

TFTP_RESULT TFTPIsFileOpened();

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().

Preconditions

TFTPOpenFile() is called.

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.

Side Effects

None

Remarks

None
**TFTPIsFileOpenReady Macro**

```c
#define TFTPIsFileOpenReady UDPIsPutReady(_tftpSocket)
```

**Description**

Macro: BOOL TFTPIsFileOpenReady(void)

Checks to see if it is okay to send TFTP file open request to remote server.

**Preconditions**

[TFTPOpen](#) is already called and [TFTPIsOpened](#)() returned TFTP_OK.

**Returns**

TRUE, if it is ok to call [TFTPOpenFile](#)()  FALSE, if otherwise.

**Side Effects**

None

**Remarks**

None
TFTPIsGetReady Function

C

```c
TFTP_RESULT TFTPIsGetReady();
```

**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.

**Preconditions**

TFTPOpenFile() is called with TFTP_FILE_MODE_READ and TFTPIsFileOpened() returned with TRUE.

**Returns**

- TFTP_OK if it 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()
- TFTP_NOT_READY if still waiting for new data.

**Side Effects**

None

**Remarks**

By default, this function uses "octet" or binary mode of file transfer.
TFTPIsOpened Function

```c
TFTP_RESULT TFTPIsOpened();
```

Description

Waits for ARP reply and opens a UDP socket to perform further TFTP operations.

Preconditions

TFTPOpen() is already called.

Returns

TFTP_OK if previous call to TFTPOpen 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.

Side Effects

None

Remarks

Once opened, application may keep TFTP socket open and future TFTP operations. If TFTPClose() is called to close the connection TFTPOpen() must be called again before performing any other TFTP operations.
TFTPIsPutReady Function

C

```c
TFTP_RESULT TFTPIsPutReady();
```

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.

Preconditions

TFTPOpenFile() is called with TFTP_FILE_MODE_WRITE and TFTPIsFileOpened() returned with TRUE.

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()
TFTP_NOT_READY if still waiting...

Side Effects

None

Remarks

None
TFTPOpen Function

```c
void TFTPOpen(
    IP_ADDR * host
);
```

**Description**

Initiates ARP for given host and prepares TFTP module for next sequence of function calls.

**Preconditions**

UDP module is already initialized and at least one UDP socket is available.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>IP address of remote TFTP server</td>
</tr>
</tbody>
</table>

**Returns**

None

**Side Effects**

None

**Remarks**

Use `TFTPIsOpenened()` to check if a connection was successfully opened or not.
TFTPOpenFile Function

C

```c
void TFTPOpenFile(
    BYTE * fileName,
    TFTP_FILE_MODE mode
);
```

Description

Prepares and sends TFTP file name and mode packet.

Preconditions

TFPTIsFileOpenReady() = TRUE

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>File name that is to be opened.</td>
</tr>
<tr>
<td>mode</td>
<td>Mode of file access Must be TFTP_FILE_MODE_READ for read TFTP_FILE_MODE_WRITE for write</td>
</tr>
</tbody>
</table>

Returns

None

Side Effects

None

Remarks

By default, this function uses "octet" or binary mode of file transfer. Use TFTPIsFileOpened() to check if file is ready to be read or written.
TFTPOpenROMFile Function

C

```c
void TFTPOpenROMFile(
    ROM BYTE * fileName,
    TFTP_FILE_MODE mode
);
```

Description

PIC18 ROM argument implementation of TFTPOpenFile
TFTPPut Function

```c
void TFTPPut(
    BYTE c
);
```

**Description**

Puts given data byte into TFTP socket. If end of data block is reached, it transmits entire block.

**Preconditions**

TFTPOpenFile() is called with TFTP_FILE_MODE_WRITE and TFTPIsPutReady() = TRUE

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Data byte that is to be written</td>
</tr>
</tbody>
</table>

**Returns**

None

**Side Effects**

None

**Remarks**

Use this function to write file to server.
**TFTP_ACCESS_ERROR Enumeration**

```
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()`.

Stack API > TFTP > Public Members > TFTP_ACCESS_ERROR Enumeration
TFTP_FILE_MODE Enumeration

```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().
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.
TFTPGetUploadStatus Function

Description

Returns the TFTP file upload status started by calling the `TFTPUploadRAMFileToHost()` or `TFTPUploadFragmentedRAMFileToHost()` functions.

Preconditions

None

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`): Upload completed successfully
- 1 (`TFTP_UPLOAD_GET_DNS`): Attempting to obtain DNS client module
- 2 (`TFTP_UPLOAD_RESOLVE_HOST`): Attempting to resolve TFTP hostname
- 3 (`TFTP_UPLOAD_CONNECT`): Attempting to ARP and contact the TFTP server
- 4 (`TFTP_UPLOAD_SEND_FILENAME`): Attempting to send the filename and receive acknowledgement.
- 5 (`TFTP_UPLOAD_SEND_DATA`): Attempting to send the file contents and receive acknowledgement.
- 6 (`TFTP_UPLOAD_WAIT_FOR_CLOSURE`): Attempting to send the final packet of file contents and receive acknowledgement.
- -1 (`TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT`): Couldn't resolve hostname
- -2 (`TFTP_UPLOAD_CONNECT_TIMEOUT`): Couldn't finish ARP and reach server
- -3 (`TFTP_UPLOAD_SERVER_ERROR`): TFTP server returned an error (ex: access denial) or file upload failed due to a timeout (partial file may have been uploaded).
Remarks

The DNS client module must be enabled to use this function. i.e. STACK_USE_DNS must be defined in TCPIPConfig.h.
TFTPUploadFragmentedRAMFileToHost Function

C

```c
void TFTPUploadFragmentedRAMFileToHost(
    ROM BYTE * vRemoteHost,
    ROM BYTE * vFilename,
    TFTP_CHUNK_DESCRIPTOR * vFirstChunkDescriptor
);
```

Description

Uploads an random, potentially non-contiguous, array of RAM bytes as a file to a remote TFTP server.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vRemoteHost</td>
<td>ROM string of the remote TFTP server to upload to (ex: &quot;www.myserver.com&quot;). 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 <code>TFTPGetUploadStatus()</code>).</td>
</tr>
<tr>
<td>vFilename</td>
<td>ROM string of the remote file to create/overwrite (ex: &quot;status.txt&quot;). 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 <code>TFTPGetUploadStatus()</code>).</td>
</tr>
<tr>
<td>vFirstChunkDescriptor</td>
<td>Pointer to a static or global (persistent) array of <code>TFTP_CHUNK_DESCRIPTOR</code> structures describing what RAM memory addresses the file contents should be obtained from. The <code>TFTP_CHUNK_DESCRIPTOR.vDataPointer</code> field should</td>
</tr>
<tr>
<td>vFirstChunkDescriptor</td>
<td>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 array must be terminated by a dummy descriptor whose TFTP_CHUNK_DESCRIPTOR.vDataPointer pointer is set to NULL. Refer to the TFTPUploadRAMFileToHost() API for an example calling sequence since it merely a wrapper to this TFTPUploadFragmentedRAMFileToHost() function.</td>
</tr>
</tbody>
</table>

**Returns**

None

**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()** 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(), you must wait until **TFTPGetUploadStatus()** returns a completion status code (<=0) before calling any other TFTP API functions.
TFTPUploadRAMFileToHost Function

C

```c
void TFTPUploadRAMFileToHost(
    ROM BYTE * vRemoteHost,
    ROM BYTE * vFilename,
    BYTE * vData,
    WORD wDataLength
);
```

Description

Uploads a contiguous array of RAM bytes as a file to a remote TFTP server.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vRemoteHost</td>
<td>ROM string of the remote TFTP server to upload to (ex: &quot;www.myserver.com&quot;). 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()).</td>
</tr>
<tr>
<td>vFilename</td>
<td>ROM string of the remote file to create/overwrite (ex: &quot;status.txt&quot;). 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()).</td>
</tr>
<tr>
<td>vData</td>
<td>Pointer to a RAM array of data to write to the file.</td>
</tr>
<tr>
<td>wDataLength</td>
<td>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</td>
</tr>
</tbody>
</table>
Returns

None

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() 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(), you must wait until TFTPGetUploadStatus() returns a completion status code (<=0) before calling any other TFTP API functions.
TFTP_CHUNK_DESCRIPTOR Structure

C

```c
typedef struct {
    BYTE * vDataPointer;
    WORD wDataLength;
} TFTP_CHUNK_DESCRIPTOR;
```

Description

This is type TFTP_CHUNK_DESCRIPTOR.
TFTP UPLOAD COMPLETE Macro

C

#define TFTP_UPLOAD_COMPLETE 0

Description

Status codes for TFTPGetUploadStatus() function. Zero means upload success, >0 means working and <0 means fatal error.
TFTP_UPLOAD_CONNECT Macro

C

#define TFTP_UPLOAD_CONNECT 3

Description

This is macro TFTP_UPLOAD_CONNECT.
TFTP_UPLOAD_CONNECT_TIMEOUT Macro

C
#define TFTP_UPLOAD_CONNECT_TIMEOUT -2

Description

This is macro TFTP_UPLOAD_CONNECT_TIMEOUT.
TFTP_UPLOAD_GET_DNS Macro

C

#define TFTP_UPLOAD_GET_DNS 1

Description

This is macro TFTP_UPLOAD_GET_DNS.
TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT

Macro

```c
#define TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT -1
```

Description

This is macro TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT.
TFTP_UPLOAD_RESOLVE_HOST Macro

C

#define TFTP_UPLOAD_RESOLVE_HOST 2

Description

This is macro TFTP_UPLOAD_RESOLVE_HOST.
TFTP_UPLOAD_SEND_DATA Macro

C

#define TFTP_UPLOAD_SEND_DATA 5

Description

This is macro TFTP_UPLOAD_SEND_DATA.
### TFTP_UPLOAD_SEND_FILENAME Macro

```c
#define TFTP_UPLOAD_SEND_FILENAME 4
```

### Description

This is macro TFTP_UPLOAD_SEND_FILENAME.
TFTP_UPLOAD_SERVER_ERROR Macro

C

#define TFTP_UPLOAD_SERVER_ERROR -3

Description

This is macro TFTP_UPLOAD_SERVER_ERROR.
# define TFTP_UPLOAD_WAIT_FOR_CLOSURE 6

**Description**

This is macro TFTP_UPLOAD_WAIT_FOR_CLOSURE.
TFTP Stack Members

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.

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFTP_ARP_TIMEOUT_VAL</td>
<td>Number of seconds to wait before declaring TIMEOUT error on Put</td>
</tr>
<tr>
<td>TFTP_GET_TIMEOUT_VAL</td>
<td>Number of seconds to wait before declaring TIMEOUT error on Get.</td>
</tr>
<tr>
<td>TFTP_MAX_RETRIES</td>
<td>Number of attempts before declaring TIMEOUT error.</td>
</tr>
</tbody>
</table>

Module

**TFTP**

Stack API > TFTP > Stack Members
TFTP_ARP_TIMEOUT_VAL Macro

C

#define TFTP_ARP_TIMEOUT_VAL (3u * TICKS_PER_SECOND)

Description

Number of seconds to wait before declaring TIMEOUT error on Put Stack API > TFTP > Stack Members > TFTP_ARP_TIMEOUT_VAL Macro
### TFTP_GET_TIMEOUT_VAL Macro

```c
#define TFTP_GET_TIMEOUT_VAL (3u * TICKS_PER_SECOND)
```

#### Description

Number of seconds to wait before declaring TIMEOUT error on Get.
TFTP_MAX_RETRIES Macro

C

#define TFTP_MAX_RETRIES (3u)

Description

Number of attempts before declaring TIMEOUT error.
TFTP Internal Members

The following functions and variables are designated as internal to the TFTP module.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFTP_OPCODE</td>
<td>Enumeration of TFTP opcodes</td>
</tr>
<tr>
<td>TFTP_STATE</td>
<td>The TFTP state machine</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_TFTPSendAck</td>
<td>Private helper function</td>
</tr>
<tr>
<td>_TFTPSendFileName</td>
<td>Private helper function</td>
</tr>
<tr>
<td>_TFTPSendROMFileName</td>
<td>PIC18 ROM variable argument implementation of _TFTPSendFileName</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFTP_BLOCK_SIZE</td>
<td>The size of a TFTP block - 512 bytes</td>
</tr>
<tr>
<td>TFTP_BLOCK_SIZE_MSB</td>
<td>The MSB of the TFTP_BLOCK_SIZE</td>
</tr>
<tr>
<td>TFTP_CLIENT_PORT</td>
<td>The TFTP Client port - a unique port on this device</td>
</tr>
<tr>
<td>TFTP_SERVER_PORT</td>
<td>The TFTP Server Port</td>
</tr>
</tbody>
</table>

Module

TFTP
## Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MutExVar</td>
<td>Mutually Exclusive variable groups to conserve RAM.</td>
</tr>
<tr>
<td>_tftpError</td>
<td>Variable to preserve error condition causes for later transmission</td>
</tr>
<tr>
<td>_tftpFlags</td>
<td>TFTP status flags</td>
</tr>
<tr>
<td>_tftpRetries</td>
<td>Tracker variable for the number of TFTP retries</td>
</tr>
<tr>
<td>_tftpSocket</td>
<td>TFTP Socket for TFTP server link</td>
</tr>
<tr>
<td>_tftpStartTick</td>
<td>Timing variable used to detect timeout conditions</td>
</tr>
<tr>
<td>tftpState</td>
<td>TFTP state machine tracker variable</td>
</tr>
<tr>
<td>smUpload</td>
<td>This is variable smUpload.</td>
</tr>
<tr>
<td>uploadChunkDescriptor</td>
<td>This is variable uploadChunkDescriptor.</td>
</tr>
<tr>
<td>uploadChunkDescriptorForRetransmit</td>
<td>This is variable uploadChunkDescriptorForRetransmit.</td>
</tr>
<tr>
<td>vUploadFilename</td>
<td>This is variable vUploadFilename.</td>
</tr>
<tr>
<td>vUploadRemoteHost</td>
<td>TFTPUploadRAMFileToHost(), TFTPUploadFragmentedRAMFileToHost() and TFTPGetUploadStatus() 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.</td>
</tr>
<tr>
<td>wUploadChunkOffset</td>
<td>This is variable wUploadChunkOffset.</td>
</tr>
<tr>
<td>wUploadChunkOffsetForRetransmit</td>
<td>This is variable wUploadChunkOffsetForRetransmit.</td>
</tr>
</tbody>
</table>
MutExVar Variable

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.
**TFTP_BLOCK_SIZE Macro**

```c
#define TFTP_BLOCK_SIZE (0x200L)
```

### Description

The size of a TFTP block - 512 bytes
TFTP_BLOCK_SIZE_MSB Macro

C

```c
#define TFTP_BLOCK_SIZE_MSB (0x02u)
```

Description

The MSB of the TFTP_BLOCK_SIZE Stack API > TFTP > Internal Members > TFTP_BLOCK_SIZE_MSB Macro

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
#define TFTP_CLIENT_PORT 65352L

## Description

The TFTP Client port - a unique port on this device
### TFTP_OPCODE Enumeration

#### C

```c
typedef enum {
    TFTP_OPCODE_RRQ = 1,
    TFTP_OPCODE_WRQ,
    TFTP_OPCODE_DATA,
    TFTP_OPCODE_ACK,
    TFTP_OPCODE_ERROR
} TFTP_OPCODE;
```

#### Description

Enumeration of TFTP opcodes

#### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFTP_OPCODE_RRQ = 1</td>
<td>Get</td>
</tr>
<tr>
<td>TFTP_OPCODE_WRQ</td>
<td>Put</td>
</tr>
<tr>
<td>TFTP_OPCODE_DATA</td>
<td>Actual data</td>
</tr>
<tr>
<td>TFTP_OPCODE_ACK</td>
<td>Ack for Get/Put</td>
</tr>
<tr>
<td>TFTP_OPCODE_ERROR</td>
<td>Error</td>
</tr>
</tbody>
</table>
TFTP_SERVER_PORT Macro

C

#define TFTP_SERVER_PORT (69L)

Description

The TFTP Server Port
TFTP_STATE Enumeration

```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
_tftpError Variable

C

WORD _tftpError;

Description

Variable to preserve error condition causes for later transmission
_tftpFlags Variable

C

union {
    struct {
        unsigned int bIsFlushed : 1;
        unsigned int bIsAcked : 1;
        unsigned int bIsClosed : 1;
        unsigned int bIsClosing : 1;
        unsigned int bIsReading : 1;
    } bits;
    BYTE Val;
} _tftpFlags;

Description

TFTP status flags

Stack API > TFTP > Internal Members > _tftpFlags Variable
_tftpRetries Variable

C

BYTE _tftpRetries;

Description

Tracker variable for the number of TFTP retries
_TFTPSendAck Function

```
static void _TFTPSendAck(WORD_VAL blockNumber);
```

Description

Private helper function

Stack API > TFTP > Internal Members > _TFTPSendAck Function
_TFTPSendFileName Function

C

```c
static void _TFTPSendFileName(
    TFTP_OPCODE command,
    BYTE * fileName
);
```

Description

Private helper function

Stack API > TFTP > Internal Members > _TFTPSendFileName Function
C

```c
static void _TFTPSendROMFileName(
    TFTP_OPCODE opcode,
    ROM_BYTE * fileName
);
```

**Description**

PIC18 ROM variable argument implementation of 

[_TFTPSendFileName](#)

---

Microchip TCP/IP Stack 5.42.08 - June 15, 2013

Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
**_tftpSocket Variable**

```
C
UDP_SOCKET _tftpSocket = TFTP_UPLOAD_COMPLETE;
```

**Description**

TFTP Socket for TFTP server link
_tftpStartTick Variable

C

DWORD _tftpStartTick;

Description

Timing variable used to detect timeout conditions
_tftpState Variable

```
_TFTP_STATE _tftpState;
```

**Description**

TFTP state machine tracker variable
smUpload Variable

C
CHAR smUpload = TFTP UPLOAD COMPLETE;

Description

This is variable smUpload.
uploadChunkDescriptor Variable

**Description**

This is variable uploadChunkDescriptor.
uploadChunkDescriptorForRetransmit Variable

C

TFTP_CHUNK_DESCRIPTOR * uploadChunkDescriptorForRetransmit;

Description

This is variable uploadChunkDescriptorForRetransmit.
vUploadFilename Variable

```c
ROM BYTE * vUploadFilename;
```

**Description**

This is variable vUploadFilename.
vUploadRemoteHost Variable

Description

TFTPUploadRAMFileToHost(), TFTPUploadFragmentedRAMFileToHost() and TFTPGetUploadStatus() 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 Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD wUploadChunkOffset;</td>
</tr>
</tbody>
</table>

Description

This is variable wUploadChunkOffset.

Stack API > TFTP > Internal Members > wUploadChunkOffset Variable
wUploadChunkOffsetForRetransmit Variable

C

WORD wUploadChunkOffsetForRetransmit;

Description

This is variable wUploadChunkOffsetForRetransmit.
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 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, TICK_MINUTE, and TICK_HOUR can be used to compare against logical time increments.

The following example implements a delay of 0.5 seconds using the Tick module:

```c
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`, `TickGetDiv256`, and `TickGetDiv64K` 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.

**Topics**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tick Public Members</strong></td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td><strong>Tick Stack Functions</strong></td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td><strong>Tick Internal Members</strong></td>
<td>Functions and variables internal to the Tick module</td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td>The following table lists functions in this documentation.</td>
</tr>
</tbody>
</table>

**Functions**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>attribute</strong></td>
<td>6-byte value to store Ticks. Allows for use over longer periods of time.</td>
</tr>
</tbody>
</table>

Stack API > Tick
Tick Public Members

The following functions and variables are available to the stack application.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TickConvertToMilliseconds</td>
<td>Converts a Tick value or difference to milliseconds.</td>
</tr>
<tr>
<td>TickGet</td>
<td>Obtains the current Tick value.</td>
</tr>
<tr>
<td>TickGetDiv256</td>
<td>Obtains the current Tick value divided by 256.</td>
</tr>
<tr>
<td>TickGetDiv64K</td>
<td>Obtains the current Tick value divided by 64K.</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TICK_HOUR</td>
<td>Represents one hour in Ticks</td>
</tr>
<tr>
<td>TICK_MINUTE</td>
<td>Represents one minute in Ticks</td>
</tr>
<tr>
<td>TICK_SECOND</td>
<td>Represents one second in Ticks</td>
</tr>
</tbody>
</table>

Module

Tick Module

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TICK</td>
<td>This is variable TICK.</td>
</tr>
</tbody>
</table>
TICK Variable

C

DWORD TICK;

Description

This is variable TICK.

Stack API > Tick > Public Members > TICK Variable
TICK_HOUR Macro

#define TICK_HOUR ((QWORD)TICKS_PER_SECOND*3600ull)

Description

Represents one hour in Ticks
# define TICK_MINUTE ((QWORD)TICKS_PER_SECOND*60ull)

## Description

Represents one minute in Ticks
TICK_SECOND Macro

```c
#define TICK_SECOND ((QWORD)TICKS_PER_SECOND)
```

Description

Represents one second in Ticks
TickConvertToMilliseconds Function

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.

Preconditions

None

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwTickValue</td>
<td>Value to convert to milliseconds</td>
</tr>
</tbody>
</table>

Returns

Input value expressed in milliseconds.

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, which will be calculated only once at compile time.
TickGet Function

C

DWORD TickGet();

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 or TickGetDiv64K for longer periods of time.

Preconditions

None

Returns

Lower 32 bits of the current Tick value.
TickGetDiv256 Function

C

DWORD TickGetDiv256();

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 for shorter periods or TickGetDiv64K for longer ones.

Preconditions

None

Returns

Middle 32 bits of the current Tick value.
TickGetDiv64K Function

C

DWORD TickGetDiv64K();

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 or TickGetDiv256 for shorter periods of time.

Preconditions

None

Returns

Upper 32 bits of the current Tick value.
Tick Stack Functions

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.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TickInit</td>
<td>Initializes the Tick manager module.</td>
</tr>
<tr>
<td>TickUpdate</td>
<td>Updates the tick value when an interrupt occurs.</td>
</tr>
</tbody>
</table>

Module

Tick Module

Stack API > Tick > Stack Functions
TickInit Function

```c
void TickInit();
```

Description

Configures the Tick module and any necessary hardware resources.

Preconditions

None

Returns

None

Remarks

This function is called only one during lifetime of the application.
TickUpdate Function

```
C

void TickUpdate();
```

**Description**

Updates the tick value when an interrupt occurs.

**Preconditions**

None

**Returns**

None

Stack API > Tick > Stack Functions > TickUpdate Function
Tick Internal Members

The following functions and variables are designated as internal to the Tick module.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetTickCopy</td>
<td>Reads the tick value.</td>
</tr>
</tbody>
</table>

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| TICKS_PER_SECOND | Internal core clock drives timer with 1:256 prescaler
                  #define TICKS_PER_SECOND (32768ul) // 32kHz
                  crystal drives timer with no scalar |

Module

Tick Module

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwInternalTicks</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.
GetTickCopy Function

C

static void GetTickCopy();

Description

This function performs an interrupt-safe and synchronized read of the 48-bit Tick value.

Preconditions

None

Returns

None
TICKS_PER_SECOND Macro

```c
#define TICKS_PER_SECOND ((GetPeripheralClock()+128ull)/256ull) //
```

**Description**

Internal core clock drives timer with 1:256 prescaler

```c
#define TICKS_PER_SECOND (32768ul) // 32kHz crystal drives timer with no scalar
```

Stack API > Tick > Internal Members > TICKS_PER_SECOND Macro
## Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>attribute</strong></td>
<td>6-byte value to store Ticks. Allows for use over longer periods of time.</td>
</tr>
</tbody>
</table>

## Module

**Tick Module**

Stack API > Tick > Functions
__attribute__ Function

```c
static volatile BYTE vTickReading __attribute__((aligned))
);
```

**Description**

6-byte value to store Ticks. Allows for use over longer periods of time.
UDP

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 are opened using **UDPOpen**. 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**.

Once the socket is opened, you can immediately begin transmitting data. To transmit a segment, call **UDPISPutReady** to determine how many bytes can be written and to designate a currently active socket. Then, use any of the **UDPPut** family of functions to write data to the socket. Once all data has been written, call **UDPFlush** 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** but before calling **UDPFlush**, the data may be lost or the module may behave unpredictably.

To check for received segments, call **UDPISGetReady**. If the return
value is non-zero, your application must consume the segment by reading data with the **UDPGet** 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** to release it back to the pool for future use.

### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UDP Public Members</strong></td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td><strong>UDP Stack Members</strong></td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td><strong>UDP Internal Members</strong></td>
<td>Functions and variables internal to the UDP module</td>
</tr>
<tr>
<td><strong>Types</strong></td>
<td>The following table lists types in this documentation.</td>
</tr>
</tbody>
</table>

#### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UDP_STATE</strong></td>
<td>UDP States</td>
</tr>
</tbody>
</table>

**Stack API > UDP**

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc.  All rights reserved.

[Contents]  |  [Index]  |  [Home]
## UDP Public Members

The following functions and variables are available to the stack application.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDPOpenEx</td>
<td>Opens a UDP socket for a client.</td>
</tr>
<tr>
<td>UDPClose</td>
<td>Closes a UDP socket and frees the handle.</td>
</tr>
<tr>
<td>UDPDiscard</td>
<td>Discards any remaining RX data from a UDP socket.</td>
</tr>
<tr>
<td>UDPFlush</td>
<td>Transmits all pending data in a UDP socket.</td>
</tr>
<tr>
<td>UDPGet</td>
<td>Reads a byte from the currently active socket.</td>
</tr>
<tr>
<td>UDPGetArray</td>
<td>Reads an array of bytes from the currently active socket.</td>
</tr>
<tr>
<td>UDPIsGetReady</td>
<td>Determines how many bytes can be read from the UDP socket.</td>
</tr>
<tr>
<td>UDPIsPutReady</td>
<td>Determines how many bytes can be written to the UDP socket.</td>
</tr>
<tr>
<td>UDPPut</td>
<td>Writes a byte to the currently active socket.</td>
</tr>
<tr>
<td>UDPPutArray</td>
<td>Writes an array of bytes to the currently active socket.</td>
</tr>
<tr>
<td>UDPPutROMArray</td>
<td>Writes an array of bytes from ROM to the currently active socket.</td>
</tr>
<tr>
<td>UDPPutROMString</td>
<td>Writes null-terminated string from ROM to the currently active socket.</td>
</tr>
<tr>
<td>UDPPutString</td>
<td>Writes null-terminated string to the currently active socket.</td>
</tr>
<tr>
<td>UDPSetRxBuffer</td>
<td>Moves the pointer within the RX buffer.</td>
</tr>
<tr>
<td>UDPSetTxBuffer</td>
<td>Moves the pointer within the TX buffer.</td>
</tr>
</tbody>
</table>
**Macros**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_UDP_PORT</td>
<td>Indicates a UDP port that is not valid</td>
</tr>
<tr>
<td>INVALID_UDP_SOCKET</td>
<td>Indicates a UDP socket that is not valid</td>
</tr>
<tr>
<td>UDPOpen</td>
<td>Macro of the legacy version of UDPOpen.</td>
</tr>
<tr>
<td>UDP_OPEN_IP_ADDRESS</td>
<td>Create a client socket and use dwRemoteHost as a literal IP address.</td>
</tr>
<tr>
<td>UDP_OPEN_NODE_INFO</td>
<td>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.</td>
</tr>
<tr>
<td>UDP_OPEN_RAM_HOST</td>
<td>Emit an undeclared identifier diagnostic if code tries to use UDP_OPEN_RAM_HOST while the DNS client module is not enabled.</td>
</tr>
<tr>
<td>UDP_OPEN_ROM_HOST</td>
<td>Emit an undeclared identifier diagnostic if code tries to use UDP_OPEN_ROM_HOST while the DNS client module is not enabled.</td>
</tr>
<tr>
<td>UDP_OPEN_SERVER</td>
<td>Create a server socket and ignore dwRemoteHost.</td>
</tr>
</tbody>
</table>

**Module**

**UDP**

**Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP_SOCKET</td>
<td>Provides a handle to a UDP Socket</td>
</tr>
</tbody>
</table>

Stack API > UDP > Public Members
INVALID_UDP_PORT Macro

C

```c
#define INVALID_UDP_PORT (0ul) // Indicates a UDP port that is
```

Description

Indicates a UDP port that is not valid

Stack API > UDP > Public Members > INVALID_UDP_PORT Macro
INVALID_UDP_SOCKET Macro

```c
#define INVALID_UDP_SOCKET (0xffu) // Indicates a UDP socket that is not valid
```

Description

Indicates a UDP socket that is not valid
UDP_SOCKET Type

C

```
typedef BYTE UDP_SOCKET;
```

Description

Provides a handle to a UDP Socket

Stack API > UDP > Public Members > UDP_SOCKET Type
UDPOpenEx Function

C

```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 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 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 only on the indicated port. Sockets are statically allocated on boot, but can be claimed with this function and freed using UDPClose.

Preconditions

**UDPInit** should be called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remoteHost</td>
<td>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: &quot;www.microchip.com&quot; or &quot;192.168.1.123&quot;), a literal destination IP address (ex: 0x7B01A8C0 or an IP_ADDR data type), or a pointer to a NODE_INFO...</td>
</tr>
<tr>
<td>remoteHostType</td>
<td>Any one of the following flags to identify the meaning of the remoteHost parameter:</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td><strong>UDP_OPEN_SERVER</strong> = Open a server socket and ignore the remoteHost parameter. (e.g. - SNMP agent, DHCP server, Announce)</td>
</tr>
<tr>
<td></td>
<td><strong>UDP_OPEN_IP_ADDRESS</strong> = Open a client socket and connect it to a remote IP address. Ex: 0x7B01A8C0 for 192.168.1.123 (DWORD type). Note that the byte ordering is big endian.</td>
</tr>
<tr>
<td></td>
<td><strong>UDP_OPEN_NODE_INFO</strong> = Open a client socket and connect it to a remote IP and MAC addresses pair stored in a NODE_INFO structure.</td>
</tr>
<tr>
<td></td>
<td><strong>UDP_OPEN_RAM_HOST</strong> = Open a client socket and connect it to a remote host who's name is stored as a null terminated string in a RAM array. Ex: &quot;www.microchip.com&quot; or &quot;192.168.0.123&quot;</td>
</tr>
<tr>
<td></td>
<td><strong>UDP_OPEN_ROM_HOST</strong> = Open a client socket and connect it to a remote host who's name is stored as a null terminated string in a literal string or ROM array. Ex: &quot;www.microchip.com&quot; or &quot;192.168.0.123&quot;</td>
</tr>
</tbody>
</table>

| localPort | UDP port number to listen on. If 0, stack will dynamically assign a unique port number to use. |
| remotePort | For client sockets, the remote port number. |
Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>A UDP socket handle that can be used for subsequent UDP API calls.</td>
</tr>
<tr>
<td>Failure</td>
<td>INVALID_UDP_SOCKET. This function fails when no more UDP socket handles are available. Increase MAX_UDP_SOCKETS to make more sockets available.</td>
</tr>
</tbody>
</table>

Remarks

When finished using the UDP socket handle, call the `UDPClose()` function to free the socket and delete the handle.
**UDPOpen Macro**

```c
#define UDPOpen(localPort,remoteNode,remotePort) UDPOpenEx((DWORD)remoteNode,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.

### Preconditions

**UDPInit()** must have been previously called.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>localPort</td>
<td>UDP port number to listen on. If 0, stack will dynamically assign a unique port number to use.</td>
</tr>
<tr>
<td>remoteNode</td>
<td>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.</td>
</tr>
<tr>
<td>remotePort</td>
<td>For client sockets, the remote port number.</td>
</tr>
</tbody>
</table>

### Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>A UDP socket handle that can be used for subsequent UDP API calls.</td>
</tr>
<tr>
<td>Failure</td>
<td><strong>INVALID_UDP_SOCKET</strong>. This function fails when no more UDP socket handles are available. Increase MAX_UDP_SOCKETS to make more sockets available.</td>
</tr>
</tbody>
</table>
Remarks

When finished using the UDP socket handle, call the `UDPClose()` function to free the socket and delete the handle.
UDPClose Function

C

```
void UDPClose(
    UDP_SOCKET s
);
```

Description

```
UDP_SOCKET UDPOpen(UDP_PORT localPort, NODE_INFO *remoteNode, UDP_PORT 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.

Preconditions

```
UDPInit() must have been previously called.
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>The socket handle to be released. If an illegal handle value is provided, the function safely does nothing.</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

This function does not affect the previously designated active socket.
UDPDiscard Function

```c
void UDPDiscard();
```

**Description**

This function discards any remaining received data in the currently active UDP socket.

**Preconditions**

`UDPIsGetReady()` was previously called to select the currently active socket.

**Returns**

None

**Remarks**

It is safe to call this function more than is necessary. If no data is available, this function does nothing.
UDPFlush Function

C

void UDPFlush();

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.

Preconditions

`UDPIsPutReady()` was previously called to specify the current socket, and data has been written to the socket using the `UDPPut` family of functions.

Returns

None

Remarks

Note that unlike `TCPFlush`, UDPFlush must be called before returning to the main stack loop. There is no auto transmit for UDP segments.
UDPGet Function

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 should be used before calling this function to specify the currently active socket.

Preconditions

UDPIsGetReady() was previously called to specify the current socket.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>The buffer to receive the data being read.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>A byte was successfully read</td>
</tr>
<tr>
<td>FALSE</td>
<td>No data remained in the read buffer</td>
</tr>
</tbody>
</table>

Stack API > UDP > Public Members > UDPGet Function
UDPGetArray Function

C

```c
WORD UDPGetArray(
  BYTE * cData,
  WORD wDataLen
);
```

Description

This function reads an array of bytes from the currently active UDP socket, while decrementing the remaining bytes available. `UDPIsGetReady` should be used before calling this function to specify the currently active socket.

Preconditions

`UDPIsGetReady()` was previously called to specify the current socket.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cData</td>
<td>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 <code>UDPSetRxBuffer()</code> will be more efficient).</td>
</tr>
<tr>
<td>wDataLen</td>
<td>Number of bytes to be read from the socket.</td>
</tr>
</tbody>
</table>

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.
UDPIsGetReady Function

C

```c
WORD UDPIsGetReady(
    UDP_SOCKET s
);
```

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() must have been previously called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>The socket to be made active (which has already been opened or is listening)</td>
</tr>
</tbody>
</table>

Returns

The number of bytes that can be read from this socket.
UDPIsPutReady Function

C

```c
WORD UDPIsPutReady(
    UDP_SOCKET s
);
```

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

UDPInit() must have been previously called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>The socket to be made active</td>
</tr>
</tbody>
</table>

Returns

The number of bytes that can be written to this socket.
UDPPut Function

```
C

BOOL UDPPut (BYTE v);
```

**Description**

This function writes a single byte to the currently active UDP socket, while incrementing the buffer length. **UDPIsPutReady** should be used before calling this function to specify the currently active socket.

**Preconditions**

**UDPIsPutReady()** was previously called to specify the current socket.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>The byte to be loaded into the transmit buffer.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The byte was successfully written to the socket.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The transmit buffer is already full and so the write failed.</td>
</tr>
</tbody>
</table>

Stack API > UDP > Public Members > UDPPut Function
UDPPutArray Function

C

```c
WORD UDPPutArray(
  BYTE * cData,
  WORD wDataLen
);
```

Description

This function writes an array of bytes to the currently active UDP socket, while incrementing the buffer length. UDPIsPutReady should be used before calling this function to specify the currently active socket.

Preconditions

UDPIsPutReady() was previously called to specify the current socket.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cData</td>
<td>The array to write to the socket.</td>
</tr>
<tr>
<td>wDataLen</td>
<td>Number of bytes from cData to be written.</td>
</tr>
</tbody>
</table>

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.
**UDPPutROMArray Function**

```c
WORD UDPPutROMArray(
    ROM BYTE * cData,
    WORD wDataLen
);
```

**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` should be used before calling this function to specify the currently active socket.

**Preconditions**

`UDPIsPutReady()` was previously called to specify the current socket.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cData</td>
<td>The array to write to the socket.</td>
</tr>
<tr>
<td>wDataLen</td>
<td>Number of bytes from cData to be written.</td>
</tr>
</tbody>
</table>

**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.

**Remarks**

This function is aliased to `UDPPutArray` on non-PIC18 platforms.
UDPPutROMString Function

C

ROM BYTE* UDPPutROMString(
    ROM BYTE * strData
);

Description

This function writes a null-terminated string from ROM to the currently active UDP socket, while incrementing the buffer length. UDPIsPutReady should be used before calling this function to specify the currently active socket.

Preconditions

UDPIsPutReady() was previously called to specify the current socket.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cData</td>
<td>Pointer to the string to be written to the socket.</td>
</tr>
</tbody>
</table>

Returns

A pointer to the byte following the last byte written. Note that this is different than the UDPPutArray functions. If this pointer does not dereference to a NULL byte, then the buffer became full and the input data was truncated.

Remarks

This function is aliased to UDPPutString on non-PIC18 platforms.
UDPPutString Function

C

BYTE* UDPPutString(
    BYTE * strData
);

Description

This function writes a null-terminated string to the currently active UDP socket, while incrementing the buffer length. UDPIsPutReady should be used before calling this function to specify the currently active socket.

Preconditions

UDPIsPutReady() was previously called to specify the current socket.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cData</td>
<td>Pointer to the string to be written to the socket.</td>
</tr>
</tbody>
</table>

Returns

A pointer to the byte following the last byte written. Note that this is different than the UDPPutArray functions. If this pointer does not dereference to a NULL byte, then the buffer became full and the input data was truncated.
UDPSetRxBuffer Function

```c
void UDPSetRxBuffer(
    WORD wOffset
);
```

Description

This function allows the read location within the RX buffer to be specified. Future calls to [UDPGet](#) and [UDPGetArray](#) will read data from the indicated location forward.

Preconditions

`UDPInit()` must have been previously called and a socket is currently active.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wOffset</td>
<td>Offset from beginning of UDP packet data payload to place the read pointer.</td>
</tr>
</tbody>
</table>

Returns

None
UDPSSetTxBuffer Function

C

```c
void UDPSSetTxBuffer(
    WORD wOffset
);
```

Description

This function allows the write location within the TX buffer to be specified. Future calls to UDPPut, UDPPutArray, UDPPutString, etc will write data from the indicated location.

Preconditions

`UDPInit()` must have been previously called and a socket is currently active.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wOffset</td>
<td>Offset from beginning of UDP packet data payload to place the write pointer.</td>
</tr>
</tbody>
</table>

Returns

None
**UDPIsOpened Function**

```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](#) to determine when the connection is set up and ready for use.

**Preconditions**

UDP is initialized.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket</td>
<td>The socket to check.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The socket has been opened and ARP has been resolved.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The socket is not currently connected.</td>
</tr>
</tbody>
</table>

**Remarks**

None
UDP_OPEN_IP_ADDRESS Macro

```
#define UDP_OPEN_IP_ADDRESS 3u
```

Description

Create a client socket and use dwRemoteHost as a literal IP address.
UDP_OPEN_NODE_INFO Macro

```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.
UDP_OPEN_RAM_HOST Macro

C

#define UDP_OPEN_RAM_HOST You_need_to_enable_STACK_USE_DNS_to_use_UDP

Description

Emit an undeclared identifier diagnostic if code tries to use UDP_OPEN_RAM_HOST while the DNS client module is not enabled.
#define UDP_OPEN_ROM_HOST  You_need_to_enable_STACK_USE_DNS_to_use_UDP

**Description**

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 Macro

C

#define UDP_OPEN_SERVER 0u

Description

Create a server socket and ignore dwRemoteHost.
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.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDPInit</td>
<td>Initializes the UDP module.</td>
</tr>
<tr>
<td>UDPProcess</td>
<td>Handles an incoming UDP segment.</td>
</tr>
<tr>
<td>UDPTask</td>
<td>Performs state management and housekeeping for UDP.</td>
</tr>
</tbody>
</table>

### Module

**UDP**

[Stack API > UDP > Stack Members](#)
UDPInit Function

```c
void UDPInit();
```

**Description**

Initializes the UDP module. This function initializes all the UDP sockets to the closed state.

**Preconditions**

None

**Returns**

None

**Remarks**

This function is called only one during lifetime of the application.
UDPProcess Function

```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()` has been called an a UDP segment is ready in the MAC buffer.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remoteNode</td>
<td>The remote node that sent this segment.</td>
</tr>
<tr>
<td>localIP</td>
<td>The destination IP address for this segment.</td>
</tr>
<tr>
<td>len</td>
<td>Total length of the UDP segment.</td>
</tr>
</tbody>
</table>

### Return Values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>A valid packet is waiting and the stack applications should be called to handle it.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The packet was discarded.</td>
</tr>
</tbody>
</table>
UDPTask Function

```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).

Preconditions

None

Remarks

UDPTask() is called once per StackTask() iteration to ensure that calls to `UDPIsPutReady()` always update the Ethernet Write pointer location between StackTask() iterations.
UDP Internal Members

The following functions and variables are designated as internal to the UDP module.

**Functions**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindMatchingSocket</td>
<td>Matches an incoming UDP segment to a currently active socket.</td>
</tr>
</tbody>
</table>

**Macros**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCAL_UDP_PORT_END_NUMBER</td>
<td>Last port number for randomized local port number selection</td>
</tr>
<tr>
<td>LOCAL_UDP_PORT_START_NUMBER</td>
<td>First port number for randomized local port number selection</td>
</tr>
</tbody>
</table>

**Module**

**UDP**

**Structures**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP_HEADER</td>
<td>Stores the header of a UDP packet</td>
</tr>
<tr>
<td>UDP_SOCKET_INFO</td>
<td>Stores information about a current UDP socket</td>
</tr>
</tbody>
</table>

**Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP_PORT</td>
<td>Stores a UDP Port Number</td>
</tr>
</tbody>
</table>
## Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activeUDPSocket</td>
<td>Indicates which UDP socket is currently active</td>
</tr>
<tr>
<td>LastPutSocket</td>
<td>Indicates the last socket to which data was written</td>
</tr>
<tr>
<td>SocketWithRxData</td>
<td>Indicates which socket has currently received data for this loop</td>
</tr>
<tr>
<td>UDPRxCount</td>
<td>Number of bytes read from this UDP segment</td>
</tr>
<tr>
<td>UDPSocketInfo</td>
<td>Stores an array of information pertaining to each UDP socket</td>
</tr>
<tr>
<td>UDPTxCount</td>
<td>Number of bytes written to this UDP segment</td>
</tr>
<tr>
<td>wGetOffset</td>
<td>Offset from beginning of payload from where data is to be read.</td>
</tr>
<tr>
<td>wPutOffset</td>
<td>Offset from beginning of payload where data is to be written.</td>
</tr>
</tbody>
</table>
activeUDPSocket Variable

C

```
UDP_SOCKET activeUDPSocket;
```

Description

Indicates which UDP socket is currently active
FindMatchingSocket Function

```c
static UDP_SOCKET FindMatchingSocket(
    UDP_HEADER * h,
    NODE_INFO * remoteNode,
    IP_ADDR * localIP
);
```

### 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>The UDP header that was received.</td>
</tr>
<tr>
<td>remoteNode</td>
<td>IP and MAC of the remote node that sent this segment.</td>
</tr>
<tr>
<td>localIP</td>
<td>IP address that this segment was destined for.</td>
</tr>
</tbody>
</table>

### Returns

A **UDP_SOCKET** handle of a matching socket, or **INVALID_UDP_SOCKET** when no match could be made.
LastPutSocket Variable

```
C

UDP_SOCKET LastPutSocket = INVALID_UDP_SOCKET;
```

Description

Indicates the last socket to which data was written
## LOCAL_UDP_PORT_END_NUMBER Macro

```c
#define LOCAL_UDP_PORT_END_NUMBER (8192u)
```

### Description

Last port number for randomized local port number selection

[Stack API > UDP > Internal Members > LOCAL_UDP_PORT_END_NUMBER Macro](#)
LOCAL_UDP_PORT_START_NUMBER Macro

C

#define LOCAL_UDP_PORT_START_NUMBER (4096u)

Description

First port number for randomized local port number selection

Stack API > UDP > Internal Members > LOCAL_UDP_PORT_START_NUMBER Macro
SocketWithRxData Variable

| C
| UDP_SOCKET SocketWithRxData = INVALID_UDP_SOCKET;

Description

Indicates which socket has currently received data for this loop
UDP_HEADER Structure

C

typedef struct {
    UDP_PORT SourcePort;
    UDP_PORT DestinationPort;
    WORD Length;
    WORD Checksum;
} UDP_HEADER;

Description

Stores the header of a UDP packet

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP_PORT SourcePort;</td>
<td>Source UDP port</td>
</tr>
<tr>
<td>UDP_PORT DestinationPort;</td>
<td>Destination UDP port</td>
</tr>
<tr>
<td>WORD Length;</td>
<td>Length of data</td>
</tr>
<tr>
<td>WORD Checksum;</td>
<td>UDP checksum of the data</td>
</tr>
</tbody>
</table>

Stack API > UDP > Internal Members > UDP_HEADER Structure

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
UDP_PORT Type

typedef WORD UDP_PORT;

Description

Stores a UDP Port Number
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;

Description

Stores information about a current UDP socket

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_INFO remoteNode;</td>
<td>10 bytes for MAC and IP address</td>
</tr>
<tr>
<td>DWORD remoteHost;</td>
<td>RAM or ROM pointer to a hostname string (ex: &quot;www.microchip.com&quot;)</td>
</tr>
<tr>
<td>UDP_PORT remotePort;</td>
<td>Remote node's UDP port number</td>
</tr>
<tr>
<td>UDP_PORT localPort;</td>
<td>Local UDP port number, or INVALID_UDP_PORT when free</td>
</tr>
<tr>
<td>UDP_STATE smState;</td>
<td>State of this socket</td>
</tr>
<tr>
<td>unsigned char bRemoteHostIsROM : 1;</td>
<td>Remote host is stored in ROM</td>
</tr>
</tbody>
</table>
UDPRxCount Variable

C

WORD UDPRxCount;

Description

Number of bytes read from this UDP segment
UDPSocketInfo Variable

C

```
UDP_SOCKET_INFO UDPSocketInfo[MAX_UDP_SOCKETS];
```

Description

Stores an array of information pertaining to each UDP socket
UDPTxCount Variable

C

WORD UDPTxCount;

Description

Number of bytes written to this UDP segment
wGetOffset Variable

```c
WORD wGetOffset;
```

**Description**

Offset from beginning of payload from where data is to be read.
### wPutOffset Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD wPutOffset;</td>
</tr>
</tbody>
</table>

**Description**

Offset from beginning of payload where data is to be written.
Types

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP_STATE</td>
<td>UDP States</td>
</tr>
</tbody>
</table>

Module

UDP

Stack API > UDP > Types
### UDP_STATE Enumeration

C
```c
typedef enum {
    UDP_DNS_IS_RESOLVED,
    UDP_DNS_RESOLVE,
    UDP_GATEWAY_SEND_ARP,
    UDP_GATEWAY_GET_ARP,
    UDP_CLOSED,
    UDP_OPENED
} UDP_STATE;
```

### Description

UDP States

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP_DNS_IS_RESOLVED</td>
<td>Special state for UDP client mode sockets</td>
</tr>
<tr>
<td>UDP_DNS_RESOLVE</td>
<td>Special state for UDP client mode sockets</td>
</tr>
<tr>
<td>UDP_GATEWAY_SEND_ARP</td>
<td>Special state for UDP client mode sockets</td>
</tr>
<tr>
<td>UDP_GATEWAY_GET_ARP</td>
<td>Special state for UDP client mode sockets</td>
</tr>
<tr>
<td>UDP_CLOSED</td>
<td>Socket is idle and unallocated</td>
</tr>
</tbody>
</table>
## Wi-Fi API

Feature similarities and differences of MRF24WB0M and MRF24WG0M

<table>
<thead>
<tr>
<th>MRF24WB0M</th>
<th>MRF24WG0M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2 Mbps</td>
<td>802.11b and 802.11g rates</td>
</tr>
<tr>
<td>Ad-hoc (OPEN, WEP)</td>
<td>Ad-hoc (OPEN, WEP)</td>
</tr>
<tr>
<td>Infrastructure (OPEN, WEP, WPA/WPA2)</td>
<td>Infrastructure (OPEN, WEP, WPA/WPA2, WPS)</td>
</tr>
<tr>
<td></td>
<td>SoftAP (OPEN, WEP)</td>
</tr>
<tr>
<td></td>
<td>Wi-Fi Direct (WPS)</td>
</tr>
<tr>
<td></td>
<td>WPA2-Enterprise Security Modes (EAP-PEAP/MSCHAPv2 and EAP-TTLS/MSCHAPv)</td>
</tr>
</tbody>
</table>

Unlike Ethernet, a Wi-Fi 802.11 application needs to initiate a connection to an access point, Wi-Fi Direct or an ad hoc network, before data communications can commence.

The WF_Config.h file has several compile-time constants that can be customized (e.g., MY_DEFAULTSSID_NAME) as needed.

In order to initiate a connection there is a sequence of steps that should be followed.

1) A connection profile must be created (see WF_CPCreate()). 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, WPA2, WPS-PBC, WPS-PIN, etc.)
   
   c. Network type (infrastructure, ad hoc, softAP, Wi-Fi Direct).
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() function must be called to initiate the connection process.

4) After WF_Connect() is called the host application will be notified when the MRF24WB0M / MRF24WG0M has succeeded (or failed) in establishing a connection via the event mechanism. The WF_Config.c file has a function, WF_ProcessEvent(), that is a template for processing MRF24WB0M / MRF24WG0M 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() when the connection succeeded. Please refer to WF_ProcessEvent for more information on WiFi event handling.

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() can be customized to support desired handling of events.
The MRF2WB0M / MRF24WG0M demos (under the Demo App, WiFi Console, WiFi EZ Config and WiFi G 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.

Below describes the host API to the MRF24WB0M / MRF24WG0M on-chip connection manager which creates and maintains Wi-Fi connections. The API is divided into these major sections:

<table>
<thead>
<tr>
<th>API Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wi-Fi Compilation Options</td>
<td>Describes the various Wi-Fi compilation options</td>
</tr>
<tr>
<td><strong>Initialization</strong></td>
<td>Functions to initialize the host API and MRF24WB0M / MRF24WG0M</td>
</tr>
<tr>
<td>Wi-Fi Network Topologies</td>
<td>Describes the various Wi-Fi network topologies</td>
</tr>
<tr>
<td>Wi-Fi Connection Profile</td>
<td>Functions to create and maintain one or more connection profiles</td>
</tr>
<tr>
<td>Wi-Fi Connection Algorithm</td>
<td>Functions to fine tune the connection algorithm</td>
</tr>
<tr>
<td>Wi-Fi Connection Manager</td>
<td>Functions to start and stop an 802.11 connection</td>
</tr>
<tr>
<td>Wi-Fi Scan</td>
<td>Functions to scan for wireless networks</td>
</tr>
<tr>
<td>Wi-Fi Security</td>
<td>Functions to handle wireless 802.11 security</td>
</tr>
<tr>
<td>Wi-Fi Tx Power Control</td>
<td>Functions to control the MRF24WB0M / MRF24WG0M Tx power</td>
</tr>
<tr>
<td>Wi-Fi Power Save</td>
<td>Functions to save power consumption by the MRF24WB0M / MRF24WG0M</td>
</tr>
<tr>
<td>Wi-Fi Driver Management Functions</td>
<td>Functions to provide access to the MRF24W Wi-Fi controller</td>
</tr>
<tr>
<td>Wi-Fi Miscellaneous</td>
<td>Functions to set a custom MAC address, get device information, etc.</td>
</tr>
<tr>
<td><strong>WF_ProcessEvent</strong></td>
<td>Functions to handle events from the MRF24WB0M /</td>
</tr>
</tbody>
</table>
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 / MRF24WG0M. To communicate with the MRF24WB0M / MRF24WG0M, 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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_SpiInit()</td>
<td>Initializes the host CPU SPI controller for usage by the Wi-Fi driver. Called by the Wi-Fi driver during initialization.</td>
</tr>
<tr>
<td>WF_SpiTxRx()</td>
<td>Transmits and/or receives SPI data from the MRF24WB0M / MRF24WG0M.</td>
</tr>
<tr>
<td>WF_SpiEnableChipSelect()</td>
<td>Set slave select line on MRF24WB0M / MRF24WG0M low (start SPI transfer).</td>
</tr>
<tr>
<td></td>
<td>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 / MRF24WG0M SPI context.</td>
</tr>
<tr>
<td></td>
<td>Set slave select line on MRF24WB0M /</td>
</tr>
</tbody>
</table>
WF_SpiDisableChipSelect() | MRF24WG0M 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 WF_SpiEnableChipSelect()).

External Interrupt

The WF_Eint.c file contains functions that the Wi-Fi Driver will use to enable and disable the MRF24WB0M / MRF24WG0M 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 / MRF24WG0M 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_EintInit()</td>
<td>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.</td>
</tr>
<tr>
<td>WF_EintEnable()</td>
<td>Enables the MRF24WB0M / MRF24WG0M external interrupt. If using real interrupts then enable the interrupt. If polling the EXINT pin then this function enables polling of the pin.</td>
</tr>
<tr>
<td>WF_EintDisable()</td>
<td>Disables the MRF24WB0M / MRF24WG0M external interrupt. If using real interrupts then disable the interrupt. If polling the EXINT pin then this function disables polling of the pin.</td>
</tr>
<tr>
<td>WF_EintIsr()</td>
<td>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 WF_EintEnable(). The Wi-Fi driver function, WFEintHandler() must be called.</td>
</tr>
<tr>
<td>WF_EintIsDisabled()</td>
<td>Returns true if the external interrupt is disabled, else returns false.</td>
</tr>
</tbody>
</table>

*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
WFEintHandler() must be called each time the MRF24WB0M / MRF24WG0M interrupt service routine (ISR) occurs.

## Modules

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wi-Fi Connection Profile</td>
<td>Functions to setup, use, and teardown connection profiles</td>
</tr>
<tr>
<td>Wi-Fi Connection Algorithm</td>
<td>Functions to alter the behavior of the connection process</td>
</tr>
<tr>
<td>Wi-Fi Connection Manager</td>
<td>Functions to manage the connection process</td>
</tr>
<tr>
<td>Wi-Fi Scan</td>
<td>Functions to direct the MRF24WB0M / MRF24WG0M to initiate a site survey</td>
</tr>
<tr>
<td>Wi-Fi Tx Power Control</td>
<td>API to control the transmit (Tx) power of the MRF24WB0M / MRF24WG0M</td>
</tr>
<tr>
<td>Wi-Fi Power Save</td>
<td>Functions to alter the power savings features of the MRF24WB0M / MRF24WG0M</td>
</tr>
<tr>
<td>Wi-Fi Miscellaneous</td>
<td>Functions for controlling miscellaneous features of the MRF24WB0M / MRF24WG0M</td>
</tr>
</tbody>
</table>

## Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wi-Fi Compilation Options</td>
<td>Describes the various Wi-Fi compilation options.</td>
</tr>
<tr>
<td>Wi-Fi Network Topologies</td>
<td>Describes the various Wi-Fi network topologies</td>
</tr>
<tr>
<td>Wi-Fi Security</td>
<td>Describes the Wi-Fi security modes supported by MRF24WB0M and MRF24WG0M</td>
</tr>
<tr>
<td>WF_ProcessEvent</td>
<td>Describes how to receive and act on events from the MRF24WB0M / MRF24WG0M</td>
</tr>
<tr>
<td>Access Point Compatibility</td>
<td></td>
</tr>
<tr>
<td>802.11 AP/Router Configuration Settings</td>
<td>Describes some basic tips for setting up and configuring a WiFi network.</td>
</tr>
<tr>
<td>WiFi Troubleshooting Tips</td>
<td>Lists some answers to some common Wi-Fi questions.</td>
</tr>
</tbody>
</table>
Wireless Packets Analysis

Describes the wireless packets transactions according to 802.11 connection protocols
Wi-Fi Compilation Options

This section describes the various Wi-Fi compilation options.

WF_Config.c/h

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 WFApi.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.

<table>
<thead>
<tr>
<th>#define</th>
<th>Controlling Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_USE_SCAN_FUNCTIONS</td>
<td>Scan API</td>
</tr>
<tr>
<td>WF_USE_TX_POWER_CONTROL_FUNCTIONS</td>
<td>Tx power control API</td>
</tr>
<tr>
<td>WF_USE_POWER_SAVE_FUNCTIONS</td>
<td>Power save API</td>
</tr>
<tr>
<td>WF_USE_MULTICAST_FUNCTIONS</td>
<td>Multicast API</td>
</tr>
<tr>
<td>WF_USE_INDIVIDUAL_SET_GETS</td>
<td>Affects all get and set functions, except the following:</td>
</tr>
<tr>
<td></td>
<td>- WF_CPSetElements()</td>
</tr>
<tr>
<td></td>
<td>- WF_CGGetElements()</td>
</tr>
<tr>
<td></td>
<td>- WF_CASetElements()</td>
</tr>
<tr>
<td></td>
<td>- WF_CAGetElements()</td>
</tr>
<tr>
<td>WF_USE_GROUP_SET_GETS</td>
<td>Affects the following functions:</td>
</tr>
<tr>
<td></td>
<td>- WF_CPSetElements()</td>
</tr>
<tr>
<td></td>
<td>- WF_CGGetElements()</td>
</tr>
<tr>
<td></td>
<td>- WF_CASetElements()</td>
</tr>
<tr>
<td></td>
<td>- WF_CAGetElements()</td>
</tr>
</tbody>
</table>

WF_DEBUG

This definition enables the WF_ASSERT macro in the Wi-Fi driver. Users' codeset is free to use this macro for debugging and customization. The WF_ASSERT macro can be compiled in or out via the WF_DEBUG definition. See the comment above the WF_DEBUG define in WFApi.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.

EZ_CONFIG_STORE

EZ_CONFIG_STORE is a feature used in Wi-Fi G Demo Board and TCPIP - WiFi EZConfig demo apps. As an example, when the MRF24W is started up as a SoftAP, MRF24W may choose to be redirected to another AP/router. What EZ_CONFIG_STORE does is to store these wireless network configurations (AppConfig) into the NVM (non-volatile memory). When the MRF24W is powered off and up again, these information (AppConfig) will be retrieved from the NVM, such that the user is not required to repeat the process of having to be redirected to the chosen AP/router again. EZ_CONFIG_STORE retains the wireless network configurations (AppConfig) and thereby saving time to reconnect to the chosen AP/router.

When MRF24W is asking to be redirected to the selected AP/router, the following sequences will take place.

1. (CustomHTTPApp.c) In HTTPPostWifiConfig(), the data struct CFGCXT will be copied to data struct AppConfig and then AppConfig.DataValid will be set to 1.

2. (MainDemo.c) AppConfig.DataValid will set CFGCXT.isWifiDoneConfigure to 1 as well and eventually will lead to WFEasyConfigProcess() being triggered.

3. (WFEasyConfig.c) WFEasyConfigProcess() will initiate SaveAppConfig(), which will program AppConfig into the NVM. Then MRF24W enters into hibernate mode and exit hibernate mode to perform connection to the newly selected network.
4. (MainDemo.c) When MRF24W is powered off and on again, InitAppConfig() / InitAppConfig2() will be initialized with AppConfig, retrieved from the internal program flash memory.

5. (MainDemo.c) When the reset button is pressed to return to default factory conditions. RestoreWifiConfig() will be invoked to erase the internal program flash memory (NVMErasePage()) and then perform a Reset().

Module

**Wi-Fi API**

Wi-Fi API &gt; Wi-Fi Compilation Options

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
Wi-Fi Network Topologies

This section describes the various Wi-Fi network topologies

- Infrastructure
- Ad-hoc
- SoftAP
- Wi-Fi Direct

Module

Wi-Fi API

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infrastructure Network</td>
<td>Describes the infrastructure network and how to set up MRF24W in infrastructure network mode</td>
</tr>
<tr>
<td>Ad-hoc Network</td>
<td>Describes the ad-hoc network and how to set up MRF24W in ad-hoc network mode</td>
</tr>
<tr>
<td>SoftAP Network</td>
<td>Describes the softAP network and how to set up MRF24WG0M in softAP network mode</td>
</tr>
<tr>
<td>Wi-Fi Direct Network</td>
<td>Describes the Wi-Fi Direct network and how to set up MRF24WG0M in Wi-Fi Direct network mode</td>
</tr>
</tbody>
</table>
Infrastructure Network

Configure MRF24W as client in an Infrastructure (CFG_WF_INFRASTRUCTURE) network

In WF_Config.h, to start up as a client in an infrastructure network, define

- **MY_DEFAULT_NETWORK_TYPE** as CFG_WF_INFRASTRUCTURE.
- **MY_DEFAULT_SSID_NAME**

This SSID will be the AP/router's SSID that you wish to be connected to.

- **MY_DEFAULT_WIFI_SECURITY_MODE**

This security mode is the AP/router's wireless security mode.

Infrastructure Network Topology

Below shows an example of the infrastructure network.
This infrastructure network shows a laptop computer and the MRF24W communicating with each other through a wireless access point (AP) and router. This network can gain access to the internet if the AP/router is connected to a WAN.
Ad-hoc Network

Configure MRF24W as Ad-hoc (CFG_WF_ADHOC) device

In WF_Config.h, to start up as a ad-hoc device, define

- MY_DEFAULT_NETWORK_TYPE as CFG_WF_ADHOC.
- MY_DEFAULT_SSID_NAME

Example "MCHPAdhoc_123".

Ad-hoc Network Topology

Below shows an example of the Ad-hoc network.

In this example, we assume that the Microchip development board with MRF24WB0MA/B or MRF24WG0MA/B is the first station to broadcast
that it wants to create the network (and it is successfully able to do so). In this case, the laptop will then join the ad-hoc network after the MRF24W has gone through the steps of setting up the ad-hoc network.

The security mode supported is open mode and WEP security.

According to specifications, ad-hoc only supports 802.11b rates of 1, 2, 5.5 and 11 Mbps.

Android devices do not support ad-hoc network.
SoftAP Network

This is only supported by MRF24WG0M.

SoftAP mode is only supported by MLA v5.42 July 2012 releases or later.

SoftAP tracking of clients' status (DHCPs.c) is only supported by MLA v5.42.06 Mar 2013 releases or later.

Configure MRF24W as softAP (CFG_WF_SOFT_AP)

In WF_Config.h, to start up as a softAP, define

- MY_DEFAULT_NETWORK_TYPE as CFG_WF_SOFT_AP.
- MY_DEFAULT_SSID_NAME

Example "MCHPSoftAP_123".

- MY_DEFAULT_CHANNEL LIST

This specifies the channel MRF24W softAP will reside in.

SoftAP Network Topology

Below shows an example of the softAP network.
SoftAP (software enabled AP) functions can be used to extend wireless coverage and share internet connection with others.

**Clients supported in SoftAP Mode**

For MRF24WG0M FW version 0x3107, softAP can only support 1 client.

For future MRF24WG0M FW versions, softAP can support up to a max of 4 clients.

**SoftAP Operations : Support of Max 1 Client Scenario**

Once the first client is connected to MRF24WG0M SoftAP, softAP will remember client’s MAC address. Only when the client does a disconnect in the 2 scenarios below, softAP will reset the MAC address to NULL. When this happens, another client can connect to
MRF24WG0M softAP.

Below are 2 scenarios in which a client disconnects from MRF24WG0M SoftAP

Scenario A

Client A does a proper disconnection, that is disassociation & disauthentication frames are sent. MRF24WG0M SoftAP, after receiving these frames, will reset the MAC address to NULL. Another client B can then connect to our SoftAP.

Scenario B

Client just powers off, in other words, did NOT inform MRF24WG0M SoftAP it is disconnecting.

(SOFTAP_CHECK_LINK_STATUS) To cater to this situation, MRF24WG0M SoftAP will ping STA by transmitting NULL DATA frames to STA to check whether STA is alive/dead. If STA is alive, it will respond to the NULL DATA frames received by transmitting ACK frames back to softAP. If STA is dead, softAP will not receive any frames from this particular device. Once the PARAM_LINK_DOWN_THRESHOLD is reached, softAP considers the device to be dead. Refer to function prototype `WF_SetLinkDownThreshold()`.

SoftAP Operations : Support of Max 4 Clients Scenario

To know how many clients and their connection status, DHCPs.c has a data struct DHCP_IP_POOL and the variable DhcpIpPool[] that keeps track of clients connected to softAP, such as client’s MAC address, IP address.

typedef struct
{
    MAC_ADDR ClientMAC;
}
IP_ADDR Client_Addr;
BOOL isUsed;
UINT32 Client_Lease_Time;
}DHCP_IP_POOL;
DHCP_IP_POOL DhcpIpPool[MAX_DHCP_CLIENTS_NUMBER];

However, be aware there would be a latency in client's status. For example, a client has disconnected from softAP. But it would take some time before this update in status in reflected in DhcpIpPool[].

**Detection of SoftAP's SSID**

Certain devices may only support active scan. Based on 802.11 specifications, passive scan is mandatory but active scan is optional.

MRF24WG0M FW version 0x3107 softAP only supports passive scan. Such devices may not be able to detect MRF24WG0M softAP.

MRF24WG0M FW version 0x3108 and later softAP supports both passive and active scan. If your device is unable to detect MRF24WG0M softAP SSID, check your MRF24WG0M FW version.

**Consideration of SoftAP consuming more transmit power**

According to 802.11 specifications, APs are expected to transmit beacons during beacon intervals (BI), thereby consuming more transmit power, as compared to being a client in infrastructure network type.

**MRF24W SoftAP channel setting**

MY_DEFAULT_CHANNEL_LIST will indicate the channel the MRF24W softAP will reside in.
For example,

```c
#define MY_DEFAULT_CHANNEL_LIST {6}
```

means that MRF24W softAP will reside in social channel 6.

It is recommended that social channel 1 or 6 or 11 be used for MRF24W softAP channel setting.

- Why does the software hangs at `WF_ProcessEvent()` in line `WF_ASSERT(FALSE)` when in softAP network type?

A possible cause could be the handling of WF event `WF_EVENT_SOFT_AP_EVENT`. This new feature is only available for MRF24WG0M (i) FW version 0x3108 and later and (ii) MLA v5.42.06 release or later. If you are using MRF24WG0M FW version 0x3108 or later and at the same time using prior to MLA v5.42.06 release, MRF24WG0M is generating this event, however `WF_ProcessEvent()` did not handle this event and fall into source code line (default: `WF_ASSERT(FALSE)`). The corrective action is to port over this event handling in `WFEVentHandler.c` and `WF_ProcessEvent()` (WiFi EZConfig) from MLA v5.42.06 release or later.

- Why does the MRF24W in softAP network type, with IP address as 192.168.1.1, on certain wireless network has problems with DHCP client assigning new IP address upon network redirection?

192.168.1.1 is a common IP address with most APs. Potential conflict can arise when there are 2 active DHCP servers in the same wireless network (i.e. AP DHCP server and MRF24W DHCP server). When network redirection is executed, the TCPIP SW may still have the MRF24W DHCP server still active, creating conflicting presence of 2 DHCP servers. This may require stack software change in TCPIP stack to be able to disable the local MRF24W DHCP server after network
Wi-Fi Direct Network

This is only supported by MRF24WG0M as a group client (GC) in a Wi-Fi Direct network type.

Configure MRF24W as Wi-Fi DIRECT (CFG_WF_P2P) Group Client (GC)

In WF_Config.h, to start up as a Wi-Fi direct group client device, define

- MY_DEFAULT_NETWORK_TYPE as CFG_WF_P2P.
- MY_DEFAULT_SSID_NAME as "DIRECT-"

which is an unique specifier to identify a Wi-Fi Direct network.

- MY_DEFAULT_CHANNEL_LIST as {1, 6, 11}

which are specified social channels for Wi-Fi Direct network.

Wi-Fi DIRECT Network Topology

Below shows an example of the WiFi Direct (peer-to-peer P2P) network.

WiFi Direct does not support 802.11b rates and therefore, only MRF24WG0MA/B is able to support such network type.
Wi-Fi Direct allows you to configure a secured wireless network between several devices, such as smart devices, laptops / computers with wireless network adaptors, without using an access point. Wi-Fi Direct supports WPS (WiFi Protected Setup) connection method, which is known as the WSC (WiFi Simple Configuration) Config Methods in the Wi-Fi Peer-to-Peer (P2P) Technical Specifications, in particular WPS push button method with WPA2.

From negotiation process, each device will determine which devices become GO (group owner) or GC (group client). The GroupOwnerIntent field in the P2P IE (information element) will indicate the level of desire
to become the GO. The higher the value, the higher the desire to be the GO. Since MRF24WG0MA/B supports the role of GC only, it implies GroupOwnerIntent=0 (P2P IE).

Within each Wi-Fi Direct network, there can be only 1 group owner, similar to only single AP in the infrastructure network.
Wi-Fi Connection Profile

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 / MRF24WG0M 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_CMDisconnect()`, update the connection profile, then call `WF_CMConnect()`.

Module

Wi-Fi API

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection Profile Public Members</td>
<td></td>
</tr>
<tr>
<td>Connection Profile Internal Members</td>
<td></td>
</tr>
</tbody>
</table>

Wi-Fi API > Wi-Fi Connection Profile

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
## Function Overview

### Connection Profile Public Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_CPCreate</td>
<td>Creates a Connection Profile on the MRF24W.</td>
</tr>
<tr>
<td>WF_CPDelete</td>
<td>Deletes a Connection Profile on the MRF24W.</td>
</tr>
<tr>
<td>WF_CPGGetAdHocBehavior</td>
<td>Gets the desired Ad Hoc behavior</td>
</tr>
<tr>
<td>WF_CPGGetBssid</td>
<td>Gets the BSSID for the specified Connection Profile ID.</td>
</tr>
<tr>
<td>WF_CPGGetElements</td>
<td>Reads the Connection Profile elements for the specified ID.</td>
</tr>
<tr>
<td>WF_CPGGetIds</td>
<td>Retrieves the CP ID bit mask.</td>
</tr>
<tr>
<td>WF_CPGGetNetworkType</td>
<td>Gets the network type for the specified Connection Profile ID.</td>
</tr>
<tr>
<td>WF_CPGGetSecurity</td>
<td>Gets the security for the specified Connection Profile.</td>
</tr>
<tr>
<td>WF_CPGGetSsid</td>
<td>Gets the SSID for the specified Connection Profile ID.</td>
</tr>
<tr>
<td>WF_CPSSetAdHocBehavior</td>
<td>Selects the desired Ad Hoc behavior</td>
</tr>
<tr>
<td>WF_CPSSetBssid</td>
<td>Sets the BSSID for the specified Connection Profile ID.</td>
</tr>
<tr>
<td>WF_CPSSetElements</td>
<td>Writes out data for a specific connection profile element.</td>
</tr>
<tr>
<td>WF_CPSSetNetworkType</td>
<td>Sets the network type for the specified Connection Profile ID.</td>
</tr>
<tr>
<td>WF_CPSSetSecurity</td>
<td>Sets the security for the specified Connection Profile.</td>
</tr>
<tr>
<td>WF_CPSSetSsid</td>
<td>Sets the SSID for the specified Connection Profile ID.</td>
</tr>
</tbody>
</table>
Module

**Wi-Fi Connection Profile**

Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFCPElementsStruct</td>
<td>Connection profile elements structure</td>
</tr>
</tbody>
</table>

**WF_CPGetSsidType**

Gets the SSID type for the specified Connection Profile ID.

**WF_CPSetSsidType**

Sets the SSID type for the specified Connection Profile ID.
WF_CPCreate Function

C

```c
void WF_CPCreate(
    UINT8 * p_CpId
);
```

Description

Connection Profile Functions

Requests the MRF24W 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 MRF24W. Users are encouraged to use 1 profile ID for MRF24W based on v5 stack SW. In v6 stack SW, plan is to have 1 profile ID for MRF24W but stack will be designed to have capability to handle multiple profiles.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_CpId</td>
<td>Pointer to where Connection Profile ID will be written. If function fails, the CP ID will be set to 0xff.</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_CPDelete Function

C

```c
void WF_CPDelete(
    UINT8 CpId
);
```

Description

Deletes the specified Connection Profile. If the Connection Profile was in FLASH it will be erased from FLASH.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile to delete.</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.

Wi-Fi API > Wi-Fi Connection Profile > Connection Profile Public Members > WF_CPDelete Function
WF_CPGetAdHocBehavior Function

```c
void WF_CPGetAdHocBehavior(
    UINT8 CpId,
    UINT8 * p_adHocBehavior
);
```

### 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

### Preconditions

MACInit must be called first.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>adHocBehavior</td>
<td>Pointer to location of the adhoc behavior value for this connection profile.</td>
</tr>
</tbody>
</table>

### Returns

None.

### Remarks

None.
WF_CPGetBssid Function

C

```c
void WF_CPGetBssid(
    UINT8 CpId,
    UINT8 * p_bssid
);
```

Description

Gets the BSSID element in a Connection Profile.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>p_bssid</td>
<td>Pointer to the BSSID</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_CPGetElements Function

C

```c
void WF_CPGetElements(
    UINT8 CpId,
    tWFCPElements * p_elements
);
```

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.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID.</td>
</tr>
<tr>
<td>p_elements</td>
<td>Pointer to Connection Profile elements structure.</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_CPGetIds Function

C

```c
void WF_CPGetIds(
    UINT8 * cpIdList
);
```

Description

Returns a list of all Connection Profile IDs that have been created on the MRF24W. 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. Users are encouraged to use 1 profile ID for MRF24W based on v5 stack SW. In v6 stack SW, plan is to have 1 profile ID for MRF24W but stack will be designed to have capability to handle multiple profiles.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_cpIdList</td>
<td>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 IDs 1 and 2 have been created.</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks
WF_CPGetNetworkType Function

```c
void WF_CPGetNetworkType(
    UINT8 CpId,
    UINT8 * p_networkType
);
```

**Description**

Gets the Network Type element of the Connection Profile. Allowable values are:

- WF_INFRASTRUCTURE
- WF_ADHOC
- WF_P2P
- WF_SOFT_AP

**Preconditions**

MACInit must be called first.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cpid</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>networkType</td>
<td>Type of network to create (infrastructure or adhoc or p2p or softAP)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**Remarks**

None.
WF_CPGetSecurity Function

```c
void WF_CPGetSecurity(
    UINT8 CpId,
    UINT8 * p_securityType,
    UINT8 * p_wepKeyIndex,
    UINT8 * p_securityKey,
    UINT8 * p_securityKeyLength
);
```

Description

Configures security for a Connection Profile.

<table>
<thead>
<tr>
<th>Security</th>
<th>Key</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_SECURITY_OPEN</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>WF_SECURITY_WEP_40</td>
<td>hex</td>
<td>4, 5 byte keys</td>
</tr>
<tr>
<td>WF_SECURITY_WEP_104</td>
<td>hex</td>
<td>4, 13 byte keys</td>
</tr>
<tr>
<td>WF_SECURITY_WPA_WITH_KEY</td>
<td>hex</td>
<td>32 bytes</td>
</tr>
<tr>
<td>WF_SECURITY_WPA_WITH_PASS_PHRASE</td>
<td>ascii</td>
<td>8-63 ascii characters</td>
</tr>
<tr>
<td>WF_SECURITY_WPA2_WITH_KEY</td>
<td>hex</td>
<td>32 bytes</td>
</tr>
<tr>
<td>WF_SECURITY_WPA2_WITH_PASS_PHRASE</td>
<td>ascii</td>
<td>8-63 ascii characters</td>
</tr>
<tr>
<td>WF_SECURITY_WPA_AUTO_WITH_KEY</td>
<td>hex</td>
<td>32 bytes</td>
</tr>
<tr>
<td>WF_SECURITY_WPA_AUTO_WITH_PASS_PHRASE</td>
<td>ascii</td>
<td>8-63 ascii characters</td>
</tr>
<tr>
<td>WF_SECURITY_WPS_PUSH_BUTTON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WF_SECURITY_WPS_PIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WF_SECURITY_WPA2_ENTERPRISE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Preconditions

MACInit must be called first.
### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cpid</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>securityType</td>
<td>Value corresponding to the security type desired.</td>
</tr>
<tr>
<td>wepKeyIndex</td>
<td>Only index 0 is valid. (Applicable for WF_SECURITY_WEP_40 or WF_SECURITY_WEP_104)</td>
</tr>
<tr>
<td>p_securityKey</td>
<td>Binary key or passphrase (not used if security is WF_SECURITY_OPEN)</td>
</tr>
<tr>
<td>securityKeyLength</td>
<td>Number of bytes in p_securityKey (not used if security is WF_SECURITY_OPEN)</td>
</tr>
</tbody>
</table>

### Returns

None.

### Remarks

None.
WF_CPGetSsid Function

C

```c
void WF_CPGetSsid(
    UINT8 CpId,
    UINT8 * p_ssid,
    UINT8 * p_ssidLength
);
```

Description

Gets the SSID and SSID Length elements in the Connection Profile.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>p_ssid</td>
<td>Pointer to the SSID string</td>
</tr>
<tr>
<td>ssidLength</td>
<td>Number of bytes in the SSID</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_CPSetAdHocBehavior Function

```c
void WF_CPSetAdHocBehavior(
    UINT8 CpId,
    UINT8 adHocBehavior
);
```

**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

**Preconditions**

MACInit must be called first.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>adHocBehavior</td>
<td>Value of the adhoc behavior for this connection profile.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**Remarks**

None.
WF_CPSetBssid Function

C

```c
void WF_CPSetBssid(
    UINT8  CpId,
    UINT8  * p_bssid
);
```

Description

Sets the BSSID element in a Connection Profile.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>p_bssid</td>
<td>Pointer to the BSSID</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_CPSetElements Function

C

```c
void WF_CPSetElements(
    UINT8 CpId,
    tWFCPElements * p_elements
);
```

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.

Preconditions

MACInit must be called.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID.</td>
</tr>
<tr>
<td>p_elements</td>
<td>Pointer to Connection Profile elements structure.</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_CPSetNetworkType Function

C

```c
void WF_CPSetNetworkType(
    UINT8 CpId,
    UINT8 networkType
);
```

Description

Sets the Network Type element for the Connection Profile. Allowable values are:

- WF_INFRASTRUCTURE
- WF_ADHOC
- WF_P2P
- WF_SOFT_AP

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>networkType</td>
<td>Type of network to create (infrastructure or adhoc or p2p or softAP)</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_CPSetSecurity Function

```c
void WF_CPSetSecurity(
    UINT8 CpId,
    UINT8 securityType,
    UINT8 wepKeyIndex,
    UINT8 * p_securityKey,
    UINT8 securityKeyLength
);
```

Description

Configures security for a Connection Profile.

<table>
<thead>
<tr>
<th>Security</th>
<th>Key</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_SECURITY_OPEN</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>WF_SECURITY_WEP_40</td>
<td>hex</td>
<td>4, 5 byte keys</td>
</tr>
<tr>
<td>WF_SECURITY_WEP_104</td>
<td>hex</td>
<td>4, 13 byte keys</td>
</tr>
<tr>
<td>WF_SECURITY_WPA_WITH_KEY</td>
<td>hex</td>
<td>32 bytes</td>
</tr>
<tr>
<td>WF_SECURITY_WPA_WITH_PASS_PHRASE</td>
<td>ascii</td>
<td>8-63 ascii characters</td>
</tr>
<tr>
<td>WF_SECURITY_WPA2_WITH_KEY</td>
<td>hex</td>
<td>32 bytes</td>
</tr>
<tr>
<td>WF_SECURITY_WPA2_WITH_PASS_PHRASE</td>
<td>ascii</td>
<td>8-63 ascii characters</td>
</tr>
<tr>
<td>WF_SECURITY_WPA_AUTO_WITH_KEY</td>
<td>hex</td>
<td>32 bytes</td>
</tr>
<tr>
<td>WF_SECURITY_WPA_AUTO_WITH_PASS_PHRASE</td>
<td>ascii</td>
<td>8-63 ascii characters</td>
</tr>
<tr>
<td>WF_SECURITY_WPS_PUSH_BUTTON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WF_SECURITY_WPS_PIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WF_SECURITY_WPA2_ENTERPRISE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Preconditions

MACInit must be called first.
**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>securityType</td>
<td>Value corresponding to the security type desired.</td>
</tr>
<tr>
<td>wepKeyIndex</td>
<td>Only index 0 is valid. (Applicable for WF_SECURITY_WEP_40 or WF_SECURITY_WEP_104)</td>
</tr>
<tr>
<td>p_securityKey</td>
<td>Binary key or passphrase (not used if security is WF_SECURITY_OPEN)</td>
</tr>
<tr>
<td>securityKeyLength</td>
<td>Number of bytes in p_securityKey (not used if security is WF_SECURITY_OPEN)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**Remarks**

None.
WF_CPSetSsid Function

C

```c
void WF_CPSetSsid(
    UINT8 CpId,
    UINT8 * p_ssid,
    UINT8 ssidLength
);
```

Description

Sets the SSID and SSID Length elements in the Connection Profile. Note that 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).

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>p_ssid</td>
<td>Pointer to the SSID string</td>
</tr>
<tr>
<td>ssidLength</td>
<td>Number of bytes in the SSID</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
## WFCPElementsStruct Structure

```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;
    UINT8 hiddenSSID;
    UINT8 wepKeyType;
};
```

### Description

Connection profile elements structure

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8 ssid[WF_MAX_SSID_LENGTH];</td>
<td>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 MRF24W, when using this profile to <code>connect</code>, will scan all channels within its regional domain. Default: SSID not used.</td>
</tr>
<tr>
<td>UINT8 bssid[WF_BSSID_LENGTH];</td>
<td>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)</td>
</tr>
<tr>
<td>Field Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>UINT8 ssidLength;</td>
<td>Number of ASCII bytes in ssid. Set to 0 is SSID is not going to be used. Default: 0</td>
</tr>
<tr>
<td>UINT8 securityType;</td>
<td>Designates the desired security level for the connection. Choices are:</td>
</tr>
<tr>
<td>UINT8 securityKey[WF_MAX_SECURITY_KEY_LENGTH];</td>
<td>Set to NULL if securityType is WF_SECURITY_OPEN. If securityKeyLength is 0 this field is ignored.</td>
</tr>
<tr>
<td>UINT8 securityKeyLength;</td>
<td>Number of bytes used in the securityKey. Set to 0 if securityType is WF_SECURITY_OPEN.</td>
</tr>
<tr>
<td>UINT8 wepDefaultKeyId;</td>
<td>This field is only used if securityType is WF_SECURITY_WEP_40 or WF_SECURITY_WEP_104. This field designates which of the four WEP keys defined in securityKey to use when connecting to a WiFi network. Only WEP key index (wepDefaultKeyId) 0 is used in RF module FW.</td>
</tr>
<tr>
<td>UINT8 networkType;</td>
<td>WF_INFRASTRUCTURE / WF_ADHOC / WF_P2P / WF_SOFT_AP Default: WF_INFRASTRUCTURE</td>
</tr>
<tr>
<td>UINT8 adHocBehavior;</td>
<td>Only applicable if networkType is WF_ADHOC. Configures Adhoc behavior. Choices are:</td>
</tr>
<tr>
<td>UINT8 hiddenSSID;</td>
<td>1 - enable hidden ssid in adhoc mode</td>
</tr>
<tr>
<td>UINT8 wepKeyType;</td>
<td>0- shared key, 1 - open key</td>
</tr>
</tbody>
</table>
WF_CPGetSsidType Function

C

```c
void WF_CPGetSsidType(
    UINT8 CpId,
    UINT8 * hidden
);
```

Description

Gets the SSID type element in the Connection Profile.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>hidden</td>
<td>Pointer to the SSID type</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_CPSetsSsidType Function

C

```c
void WF_CPSetsSsidType(
    UINT8 CpId,
    UINT8 hidden
);
```

Description

Sets the SSID type element in the Connection Profile. This is valid only when we create AdHoc network.

Preconditions

MACInit must be called first.

Returns

None.

Remarks

None.

Wi-Fi API > Wi-Fi Connection Profile > Connection Profile Public Members > WF_CPSetsSsidType Function
### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LowLevel_CPGetElement</td>
<td>Get an element of the connection profile on the MRF24W.</td>
</tr>
<tr>
<td>LowLevel_CPSetElement</td>
<td>Set an element of the connection profile on the MRF24W.</td>
</tr>
</tbody>
</table>

### Module

**Wi-Fi Connection Profile**
LowLevel_CPGetElement Function

C

```c
static void LowLevel_CPGetElement(
    UINT8  CpId,
    UINT8  elementId,
    UINT8  * p_elementData,
    UINT8  elementDataLength,
    UINT8  dataReadAction
);
```

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.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>elementId</td>
<td>Element that is being read</td>
</tr>
<tr>
<td>p_elementData</td>
<td>Pointer to where element data will be written</td>
</tr>
<tr>
<td>elementDataLength</td>
<td>Number of element data bytes that will be read</td>
</tr>
<tr>
<td>dataReadAction</td>
<td>If TRUE then read data per parameters 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.</td>
</tr>
</tbody>
</table>

Returns
None.

Remarks

None.
LowLevel_CPSetElement Function

```c
static void LowLevel_CPSetElement(
    UINT8 CpId,
    UINT8 elementId,
    UINT8 * p_elementData,
    UINT8 elementDataLength
);
```

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.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>elementId</td>
<td>Element that is being set</td>
</tr>
<tr>
<td>p_elementData</td>
<td>Pointer to element data</td>
</tr>
<tr>
<td>elementDataLength</td>
<td>Number of bytes pointed to by p_elementData</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks
None.
Wi-Fi Connection Algorithm

The connection algorithm is used to fine-tune the MRF24WB0M / MRF24WG0M algorithm used in the connection process. The connection algorithm can only be changed when the MRF24WB0M / MRF24WG0M is not connected to an 802.11 network.

Module

Wi-Fi API

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection Algorithm Public Members</td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td>Connection Algorithm Internal Members</td>
<td>Functions and variables internal to the module</td>
</tr>
</tbody>
</table>

Wi-Fi API  >  Wi-Fi Connection Algorithm
## Connection Algorithm Public Members

The following functions and variables are available to the stack application.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_CAGetBeaconTimeout</td>
<td>Reads the beacon timeout value.</td>
</tr>
<tr>
<td>WF_CAGetBeaconTimeoutAction</td>
<td>Reads the Connection Algorithm beacon timeout action.</td>
</tr>
<tr>
<td>WF_CAGetChannelList</td>
<td>Gets the channel list.</td>
</tr>
<tr>
<td>WF_CAGetConnectionProfileList</td>
<td>Not currently supported</td>
</tr>
<tr>
<td>WF_CAGetDeauthAction</td>
<td>Reads the Connection Algorithm deauth action.</td>
</tr>
<tr>
<td>WF_CAGetElements</td>
<td>Reads all Connection Algorithm elements.</td>
</tr>
<tr>
<td>WF_CAGGetEventNotificationAction</td>
<td>Reads the Connection Algorithm event notification action.</td>
</tr>
<tr>
<td>WF_CAGGetListenInterval</td>
<td>Gets the listen interval.</td>
</tr>
<tr>
<td>WF_CAGGetListRetryCount</td>
<td>Gets the list retry count</td>
</tr>
<tr>
<td>WF_CAGetMaxChannelTime</td>
<td>Gets the Max Channel Time (in milliseconds)</td>
</tr>
<tr>
<td>WF_CAGetMinChannelTime</td>
<td>Gets the current Connection Algorithm minimum channel time.</td>
</tr>
<tr>
<td>WF_CAGetProbeDelay</td>
<td>Gets the Probe Delay (in microseconds)</td>
</tr>
<tr>
<td>WF_CAGetRssi</td>
<td>Gets the RSSI threshold</td>
</tr>
<tr>
<td>WF_CAGetScanCount</td>
<td>Gets the scan count</td>
</tr>
<tr>
<td>WF_CAGetScanType</td>
<td>Gets the Connection Algorithm scan type</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WF_CASetBeaconTimeout</td>
<td>Sets the beacon timeout value.</td>
</tr>
<tr>
<td>WF_CASetBeaconTimeoutAction</td>
<td>Action to take if a connection is lost due to a beacon timeout.</td>
</tr>
<tr>
<td>WF_CASetChannelList</td>
<td>Sets the channel list.</td>
</tr>
<tr>
<td>WF_CASetConnectionProfileList</td>
<td>Not currently supported</td>
</tr>
<tr>
<td>WF_CASetDeauthAction</td>
<td>Sets the DeauthAction used by the Connection Algorithm.</td>
</tr>
<tr>
<td>WF_CASetElements</td>
<td>Writes all Connection Algorithm elements.</td>
</tr>
<tr>
<td>WF_CASetEventNotificationAction</td>
<td>Sets the WiFi events that the host wishes to be notified of.</td>
</tr>
<tr>
<td>WF_CASetListenInterval</td>
<td>Sets the <code>listen</code> interval.</td>
</tr>
<tr>
<td>WF_CASetListRetryCount</td>
<td>Sets the list retry count</td>
</tr>
<tr>
<td>WF_CASetMaxChannelTime</td>
<td>Sets the maximum channel time (in milliseconds)</td>
</tr>
<tr>
<td>WF_CASetMinChannelTime</td>
<td>Sets the minimum channel time (in milliseconds)</td>
</tr>
<tr>
<td>WF_CASetProbeDelay</td>
<td>Sets the Probe Delay (in microseconds)</td>
</tr>
<tr>
<td>WF_CASetRssi</td>
<td>Sets the RSSI threshold</td>
</tr>
<tr>
<td>WF_CASetScanCount</td>
<td>Sets the scan count</td>
</tr>
<tr>
<td>WF_CASetScanType</td>
<td>Sets the Connection Algorithm scan type</td>
</tr>
<tr>
<td>WF_CAGetDtimInterval</td>
<td>Gets the dtim interval for MRF24WG0MA/B.</td>
</tr>
<tr>
<td>WF_CASetDtimInterval</td>
<td>Sets the dtim interval for MRF24WG0MA/B.</td>
</tr>
<tr>
<td>WF_CAGetBeaconPeriod</td>
<td>Retrieves beacon period in Adhoc start mode for MRF24WG0MA/B.</td>
</tr>
<tr>
<td>WF_CASetBeaconPeriod</td>
<td>Sets the beacon period in Adhoc start mode for MRF24WG0MA/B.</td>
</tr>
</tbody>
</table>
Module

Wi-Fi Connection Algorithm

Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFCAElementsStruct</td>
<td>Connection Algorithm Elements</td>
</tr>
</tbody>
</table>

Wi-Fi API > Wi-Fi Connection Algorithm > Connection Algorithm Public Members

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
WF_CAGetBeaconTimeout Function

C

```c
void WF_CAGetBeaconTimeout(
    UINT8 * p_beaconTimeout
);
```

Description

Gets the Beacon Timeout used by the Connection Algorithm.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No monitoring of the beacon timeout condition. The host will not be notified of this event.</td>
</tr>
<tr>
<td>1-255</td>
<td>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.</td>
</tr>
</tbody>
</table>

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_beaconTimeout</td>
<td>Pointer where beacon timeout value is written</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_CAGetBeaconTimeoutAction Function

C

```c
void WF_CAGetBeaconTimeoutAction(
    UINT8 * p_beaconTimeoutAction
);
```

Description

Gets the Beacon Timeout Action used by the Connection Algorithm.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_beaconTimeoutAction</td>
<td>Pointer where returned value is written. The value will be either</td>
</tr>
<tr>
<td></td>
<td>• WF_ATTEMPT_TO_RECONNECT</td>
</tr>
<tr>
<td></td>
<td>• WF_DO_NOT_ATTEMPT_TO_RECONNECT</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_CAGetChannelList Function

C

```c
void WF_CAGetChannelList(
    UINT8 * p_channelList,
    UINT8 * p_numChannels
);
```

Description

Gets the Channel List used by the Connection Algorithm.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_channelList</td>
<td>Pointer to where channel list will be returned</td>
</tr>
<tr>
<td>p_numChannels</td>
<td>Pointer to where number of channels in list will be returned</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.

Wi-Fi API > Wi-Fi Connection Algorithm > Connection Algorithm Public Members > WF_CAGetChannelList Function
WF_CAGetConnectionProfileList Function

C

```c
void WF_CAGetConnectionProfileList(
    UINT8 cpList[WF_CP_LIST_LENGTH]
);
```

Description

Not currently supported

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpList</td>
<td>Array of connection profile ID's used to create CP list</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

Not currently supported. The list size is always WF_CP_LIST_SIZE.
WF_CAGetDeauthAction Function

C

```c
void WF_CAGetDeauthAction(
    UINT8 * p_deauthAction
);
```

**Description**

Gets the DeauthAction used by the Connection Algorithm.

**Preconditions**

MACInit must be called first.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_deauthAction</td>
<td>Pointer where returned value is written. The value will be either</td>
</tr>
<tr>
<td>be either</td>
<td>• WF_ATTEMPT_TO_RECONNECT</td>
</tr>
<tr>
<td>be either</td>
<td>• WF_DO_NOT_ATTEMPT_TO_RECONNECT</td>
</tr>
</tbody>
</table>

**Returns**

None.

**Remarks**

None.
WF_CAGetElements Function

```
C

void WF_CAGetElements(
    tWFCAElements * p_elements
);
```

### Description

Sends a message to the MRF24W which requests all the Connection Algorithm elements.

### Preconditions

MACInit must be called first.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_elements</td>
<td>Pointer to the output structure (tWFCAElements) where the connection algorithm elements are written.</td>
</tr>
</tbody>
</table>

### Returns

None

### Remarks

None

Wi-Fi API > Wi-Fi Connection Algorithm > Connection Algorithm Public Members > WF_CAGetElements Function
WF_CAGetEventNotificationAction Function

C

```c
void WF_CAGetEventNotificationAction(
    UINT8 * p_eventNotificationAction
);
```

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:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>WF_NOTIFY_CONNECTION_ATTEMPT_SUCCESSFUL</td>
</tr>
<tr>
<td>1</td>
<td>WF_NOTIFY_CONNECTION_ATTEMPT_FAILED</td>
</tr>
<tr>
<td>2</td>
<td>WF_NOTIFY_CONNECTION_TEMPORARILY-lost</td>
</tr>
<tr>
<td>3</td>
<td>WF_NOTIFY_CONNECTION_PERMANENTLY-lost</td>
</tr>
<tr>
<td>4</td>
<td>WF_NOTIFY_CONNECTION_REESTABLISHED</td>
</tr>
</tbody>
</table>

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_eventNotificationAction</td>
<td>Pointer to where returned value is written.</td>
</tr>
</tbody>
</table>

Returns

None.
Remarks

None.

Wi-Fi API > Wi-Fi Connection Algorithm > Connection Algorithm Public Members > WF_CAGetEventNotificationAction Function

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
WF_CAGetListenInterval Function

C

```c
void WF_CAGetListenInterval(
    UINT16  * p_listenInterval
);
```

**Description**

Gets the Listen Interval used by the Connection Algorithm. This value is measured in 100ms intervals, the default beacon period of APs.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MRF24W wakes up every 100ms to receive buffered messages.</td>
</tr>
<tr>
<td>2</td>
<td>MRF24W wakes up every 200ms to receive buffered messages.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>65535</td>
<td>MRF24W wakes up every 6535.5 seconds (~109 minutes) to receive buffered messages.</td>
</tr>
</tbody>
</table>

**Preconditions**

MACInit must be called first. Only used when PS Poll mode is enabled.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_listenInterval</td>
<td>Pointer to where listen interval is returned</td>
</tr>
</tbody>
</table>

**Returns**

None.

**Remarks**
WF_CAGetListRetryCount Function

C

```c
void WF_CAGetListRetryCount(
    UINT8 * p_listRetryCount
);
```

Description

See description in WF_CASetListRetryCount()

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_listRetryCount</td>
<td>Pointer to where list retry count is written.</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

None
WF_CAGetMaxChannelTime Function

C

```c
void WF_CAGetMaxChannelTime(
    UINT16 * p_minChannelTime
);
```

Description

Gets the maximum time the connection manager waits for a probe response after sending a probe request.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_maxChannelTime</td>
<td>Pointer where maximum channel time is written</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

Default is 400ms

Wi-Fi API > Wi-Fi Connection Algorithm > Connection Algorithm Public Members > WF_CAGetMaxChannelTime Function
**WF_CAGetMinChannelTime Function**

```c
void WF_CAGetMinChannelTime(
    UINT16 * p_minChannelTime
);
```

**Description**

Gets the minimum time the connection manager waits for a probe response after sending a probe request.

**Preconditions**

MACInit must be called first.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_minChannelTime</td>
<td>Pointer where minimum time to wait for a probe response (in milliseconds) will be written.</td>
</tr>
</tbody>
</table>

**Returns**

None

**Remarks**

None
WF_CAGetProbeDelay Function

C

```c
void WF_CAGetProbeDelay(
    UINT16 * p_probeDelay
);
```

Description

The number of microseconds to delay before transmitting a probe request following the channel change event.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_probeDelay</td>
<td>Pointer to where probe delay is written</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

Default is 20uS
WF_CAGetRssi Function

C

```c
void WF_CAGetRssi(
    UINT8 * p_rssi
);
```

Description

See WF_CASetRssi. 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.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_rssi</td>
<td>Pointer to where RSSI value is written</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

Default is 255
WF_CAGetScanCount Function

```c
void WF_CAGetScanCount(
    UINT8 * p_scanCount
);
```

Description

The number of times the Connection Manager will scan a channel while attempting to find a particular WiFi network.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_scanCount</td>
<td>Pointer to where scan count is written</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

Default is 1
WF_CAGetScanType Function

```c
void WF_CAGetScanType(
    UINT8 * p_scanType
);
```

**Description**

Reads the current Connection Algorithm scan type. In active scan, STA will transmit probe request frames and AP/routers will respond by transmitting probe response frames. For AP/Router with hidden SSID, active scan is used. In passive scan, AP/router will continuously transmit beacon frames for every beacon interval (BI), any STAs may/will receive these beacons and know of existence of this AP/router.

**Preconditions**

MACInit must be called first.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_scanType</td>
<td>Pointer where Connection Algorithm scan type is written. Either WF_ACTIVE_SCAN or WF_PASSIVE_SCAN.</td>
</tr>
</tbody>
</table>

**Returns**

None

**Remarks**

Active scanning causes the MRF24W to send probe requests. Passive scanning implies the MRF24W only listens for beacons. Default is WF_ACTIVE_SCAN.
WF_CAGetScanType Function

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
WF_CASetBeaconTimeout Function

C

```c
void WF_CASetBeaconTimeout(
    UINT8 beaconTimeout
);
```

Description

Sets the Beacon Timeout used by the Connection Algorithm.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No monitoring of the beacon timeout condition. The host will not be notified of this event.</td>
</tr>
<tr>
<td>1-255</td>
<td>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.</td>
</tr>
</tbody>
</table>

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>beaconTimeout</td>
<td>Number of beacons that can be missed before the action in beaconTimeoutAction is taken.</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks
WF_CASetBeaconTimeoutAction Function

C

```c
void WF_CASetBeaconTimeoutAction(
    UINT8  beaconTimeoutAction
);
```

Description

Sets the Beacon Timeout Action used by the Connection Algorithm.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>beaconTimeoutAction</td>
<td>Action to take if a connection is lost due to a beacon timeout. Choices are either</td>
</tr>
<tr>
<td></td>
<td>• WF_ATTEMPT_TO_RECONNECT</td>
</tr>
<tr>
<td></td>
<td>• WF_DO_NOT_ATTEMPT_TO_RECONNECT</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_CASetChannelList Function

```c
void WF_CASetChannelList(
    UINT8 * p_channelList,
    UINT8 numChannels
);
```

Description

Sets the Channel List used by the Connection Algorithm. MRF24W is programmed with channel 1 to 11 as default.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_channelList</td>
<td>Pointer to channel list.</td>
</tr>
<tr>
<td>numChannels</td>
<td>Number of channels in p_channelList. If set to 0, the MRF24W will use all valid channels for the current regional domain.</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_CASetConnectionProfileList Function

C

```c
void WF_CASetConnectionProfileList(
    UINT8 cpList[WF_CP_LIST_LENGTH]
);
```

Description

Not currently supported

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpList</td>
<td>Array of connection profile ID's used to create CP list</td>
</tr>
</tbody>
</table>

Returns

None

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.
WF_CASetDeauthAction Function

C

```c
void WF_CASetDeauthAction(
    UINT8 deauthAction
);
```

Description

Action to take if a connection is lost due to receiving a deauthentication message from an AP.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deauthAction</td>
<td>Action to take in the event of a deauthentication.</td>
</tr>
</tbody>
</table>

Allowable values are

- WF_ATTEMPT_TO_RECONNECT
- WF_DO_NOT_ATTEMPT_TO_RECONNECT

Returns

None.

Remarks

None.
WF_CASetElements Function

C

```c
void WF_CASetElements(
tWFCAElements * p_elements);
```

Description

Connection Algorithm Functions

Sends a message to the MRF24W which sets all the Connection Algorithm elements.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_elements</td>
<td>Pointer to the input structure (tWFCAElements) containing the connection algorithm elements.</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

None
WF_CASetEventNotificationAction Function

```c
void WF_CASetEventNotificationAction(
    UINT8  eventNotificationAction
);
```

**Description**

Sets the Event Notification Action used by the Connection Algorithm. The bit mask for the allowable entries is as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>WF_NOTIFY_CONNECTION_ATTEMPT_SUCCESSFUL</td>
</tr>
<tr>
<td>1</td>
<td>WF_NOTIFY_CONNECTION_ATTEMPT_FAILED</td>
</tr>
<tr>
<td>2</td>
<td>WF_NOTIFY_CONNECTION_TEMPORARILY_LOST</td>
</tr>
<tr>
<td>3</td>
<td>WF_NOTIFY_CONNECTION_PERMANENTLY_LOST</td>
</tr>
<tr>
<td>4</td>
<td>WF_NOTIFY_CONNECTION_REESTABLISHED</td>
</tr>
</tbody>
</table>

**Preconditions**

MACInit must be called first.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventNotificationAction</td>
<td>Bit mask indicating which events the host wants to be notified of.</td>
</tr>
</tbody>
</table>

**Returns**

None.
Remarks

None.

Wi-Fi API > Wi-Fi Connection Algorithm > Connection Algorithm Public Members > WF_CASetEventNotificationAction Function
WF_CASetListenInterval Function

```c
void WF_CASetListenInterval(
    UINT16 listenInterval
);
```

**Description**

Sets the `listen` interval used by the Connection Algorithm. This value is measured in 100ms intervals, the default beacon period of APs.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MRF24W wakes up every 100ms to receive buffered messages.</td>
</tr>
<tr>
<td>2</td>
<td>MRF24W wakes up every 200ms to receive buffered messages.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>65535</td>
<td>MRF24W wakes up every 6535.5 seconds (~109 minutes) to receive buffered messages.</td>
</tr>
</tbody>
</table>

**Preconditions**

MACInit must be called first. Only used when PS Poll mode is enabled.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listenInterval</td>
<td>Number of 100ms intervals between instances when the MRF24W wakes up to receive buffered messages from the network.</td>
</tr>
</tbody>
</table>

**Returns**

None.
Remarks

None.

Wi-Fi API > Wi-Fi Connection Algorithm > Connection Algorithm Public Members > WF_CASetListenInterval Function
WF_CASetListRetryCount Function

C

```c
void WF_CASetListRetryCount(
    UINT8 listRetryCount
);
```

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 with the current Connection Profile before giving up.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listRetryCount</td>
<td>0 to 254 or WF_RETRY_FOREVER (255)</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

None
WF_CASetMaxChannelTime Function

C

```c
void WF_CASetMaxChannelTime(
    UINT16 minChannelTime
);
```

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.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxChannelTime</td>
<td>Maximum time to wait for a probe response (in milliseconds)</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

Default is 400ms
WF_CASetMinChannelTime Function

C

```c
void WF_CASetMinChannelTime(
    UINT16  minChannelTime
);
```

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.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minChannelTime</td>
<td>Minimum time to wait for a probe response (in milliseconds)</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

Default is 200ms
WF_CASetProbeDelay Function

C

```c
void WF_CASetProbeDelay(
  UINT16 probeDelay
);
```

Description

The number of microseconds to delay before transmitting a probe request following the channel change event.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>probeDelay</td>
<td>Desired probe delay</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

Default is 20uS
WF_CASetRssi Function

C

void WF_CASetRssi(
    UINT8 rssi
);

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 (MRF24WB), 1 -
128 (MRF24WG): Only connect 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.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanCount</td>
<td>Desired scan count</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

Default is 255
WF_CASetScanCount Function

C

```c
void WF_CASetScanCount(
    UINT8 scanCount
);
```

**Description**

The number of times the Connection Manager will scan a channel while attempting to find a particular WiFi network.

**Preconditions**

MACInit must be called first.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanCount</td>
<td>Desired scan count</td>
</tr>
</tbody>
</table>

**Returns**

None

**Remarks**

Default is 1

[Wi-Fi API] > [Wi-Fi Connection Algorithm] > [Connection Algorithm Public Members] > [WF_CASetScanCount Function]
WF_CASetScanType Function

```c
void WF_CASetScanType(
    UINT8 scanType
);
```

Description

Configures the Connection Algorithm for the desired scan type. In active scan, STA will transmit probe request frames and AP/routers will respond by transmitting probe response frames. For AP/Router with hidden SSID, active scan is used. In passive scan, AP/router will continuously transmit beacon frames for every beacon interval (BI), any STAs may/will receive these beacons and know of existence of this AP/router.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanType</td>
<td>Desired scan type. Either WF_ACTIVE_SCAN or WF_PASSIVE_SCAN.</td>
</tr>
</tbody>
</table>

Returns

None

Remarks

Active scanning causes the MRF24W to send probe requests. Passive scanning implies the MRF24W only listens for beacons. Default is WF_ACTIVE_SCAN.
WFCAElementsStruct Structure

C

```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;
    UINT16 dtimInterval;
    UINT16 beaconPrd;
};
```

Description

Connection Algorithm Elements

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT16 listenInterval;</td>
<td>This parameter is only used when PS Poll mode is enabled. See WF_PsPollEnable(). Number of 100ms intervals between instances when the MRF24W wakes up to received buffered messages from the network. Range is from 1 (100ms) to 6553.5 sec (~109 min). Note that the 802.11 standard defines the listen interval in terms of Beacon Periods, which are typically 100ms. If the MRF24W 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</td>
</tr>
</tbody>
</table>
the actual Beacon Period as closely as possible. Important Note: If the listenInterval is modified while connected to a network the MRF24W will automatically reconnect to the network with the new Beacon Period value. This may cause a temporary loss of data packets.

<p>| UINT8 scanType; | WF_ACTIVE_SCAN (Probe Requests transmitted out) or WF_PASSIVE_SCAN (listen only for beacons received) Default: WF_ACTIVE_SCAN |
| UINT8 rssi; | Specifies RSSI restrictions when connecting. This field is only used if: |
| | 1. The Connection Profile has not defined a SSID or BSSID or |
| | 2. An SSID is defined in the Connection Profile and multiple AP's are discovered with the same SSID. |
| UINT8 connectionProfileList[WF_CP_LIST_LENGTH]; | Note: Connection Profile lists are not yet supported. This array should be set to all FF's. |
| UINT8 listRetryCount; | This field is used to specify the number of retries for the single connection profile before taking the connection lost action. Range 1 to 254 or WF_RETRY_FOREVER (255) Default is 3 |
| 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. |
| | 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 |</p>
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8 beaconTimeoutAction;</td>
<td>Specifies the number of beacons that can be missed before the action described in beaconTimeoutAction is taken.</td>
</tr>
<tr>
<td>UINT8 deauthAction;</td>
<td>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</td>
</tr>
<tr>
<td>UINT8 channelList[WF_CHANNEL_LIST_LENGTH];</td>
<td>List of one or more channels that the MRF24W 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 MRF24W (set at manufacturing).</td>
</tr>
<tr>
<td>UINT8 numChannelsInList;</td>
<td>Number of channels in channelList. If set to 0 then the MRF24W will populate the list with all valid channels for the regional domain. Default: The number of valid channels for the regional domain of the MRF24W (set at manufacturing).</td>
</tr>
<tr>
<td>UINT8 beaconTimeout;</td>
<td>Specifies the number of beacons that can be missed before the action described in beaconTimeoutAction is taken.</td>
</tr>
<tr>
<td>UINT8 scanCount;</td>
<td>The number of times to scan a channel while attempting to find a particular access point. Default: 1</td>
</tr>
<tr>
<td>UINT16 minChannelTime;</td>
<td>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</td>
</tr>
<tr>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UINT16 maxChannelTime;</td>
<td>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</td>
</tr>
<tr>
<td>UINT16 probeDelay;</td>
<td>The number of microseconds to delay before transmitting a probe request following the channel change event. Default: 20us</td>
</tr>
<tr>
<td>UINT16 dtimInterval;</td>
<td>Default : 4</td>
</tr>
<tr>
<td>UINT16 beaconPrd;</td>
<td>Default : 100 (ms)</td>
</tr>
</tbody>
</table>
WF_CAGetDtimInterval Function

C

```c
void WF_CAGetDtimInterval(
    UINT16 * p_dtimInterval
);
```

Description

Gets the DTIM Interval used by the Connection Algorithm for MRF24WG0MA/B.

Preconditions

MACInit must be called first. Only used when PS Poll mode is enabled.

Returns

None.

Remarks

None.

Wi-Fi API > Wi-Fi Connection Algorithm > Connection Algorithm Public Members > WF_CAGetDtimInterval Function
WF_CASetDtimInterval Function

C

```c
void WF_CASetDtimInterval(
    UINT16 dtimInterval
);
```

Description

Sets the dtim interval used by the Connection Algorithm for MRF24WG0MA/B.

Preconditions

MACInit must be called first. Only used when PS Poll mode is enabled.

Returns

None.

Remarks

None.
WF_CAGetBeaconPeriod Function

C

```c
void WF_CAGetBeaconPeriod(
    UINT16 * beaconPeriod
);
```

Description

Gets Beacon period in Adhoc start mode for MRF24WG0MA/B.

Preconditions

MACInit must be called first.

Returns

None.

Remarks

None.
WF_CASetBeaconPeriod Function

C

```c
void WF_CASetBeaconPeriod(
    UINT16 beaconPeriod
);
```

Description

Sets the beacon period used by the Connection Algorithm for MRF24WG0MA/B.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>beaconPeriod</td>
<td>beacon period in adhoc start mode by ms resolution</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
The following functions and variables are designated as internal to the module.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LowLevel_CAGetElement</td>
<td>Get an element of the connection algorithm on the MRF24W.</td>
</tr>
<tr>
<td>LowLevel_CASetElement</td>
<td>Set an element of the connection algorithm on the MRF24W.</td>
</tr>
<tr>
<td>SetEventNotificationMask</td>
<td>Sets the event notification mask.</td>
</tr>
</tbody>
</table>

### Module

**Wi-Fi Connection Algorithm**

[Wi-Fi API] > [Wi-Fi Connection Algorithm] > [Connection Algorithm Internal Members]
LowLevel_CAGetElement Function

C

```c
static void LowLevel_CAGetElement(
    UINT8  elementId,
    UINT8  * p_elementData,
    UINT8  elementDataLength,
    UINT8  dataReadAction
);
```

Description

Low-level function to send the appropriate management message to the MRF24W to get the Connection Algorithm element.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementId</td>
<td>Element that is being read</td>
</tr>
<tr>
<td>p_elementData</td>
<td>Pointer to where element data will be written</td>
</tr>
<tr>
<td>elementDataLength</td>
<td>Number of element data bytes that will be read</td>
</tr>
<tr>
<td>dataReadAction</td>
<td>If TRUE then read data per parameters 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.</td>
</tr>
</tbody>
</table>

Returns

None.

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.
LowLevel_CASetElement Function

C

```c
static void LowLevel_CASetElement(
    UINT8 elementId,
    UINT8 * p_elementData,
    UINT8 elementDataLength
);
```

Description

LOCAL FUNCTION PROTOTYPES

Low-level function to send the appropriate management message to the MRF24W to set the Connection Algorithm element.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementId</td>
<td>Element that is being set</td>
</tr>
<tr>
<td>p_elementData</td>
<td>Pointer to element data</td>
</tr>
<tr>
<td>elementDataLength</td>
<td>Number of bytes pointed to by p_elementData</td>
</tr>
</tbody>
</table>

Returns

None.

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.
SetEventNotificationMask Function

C

```c
static void SetEventNotificationMask(
    UINT8 eventNotificationBitMask
);
```

Description

Sets the event notification mask for the Connection Algorithm. Allowable values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>WF_NOTIFY_CONNECTION_ATTEMPT_SUCCESSFUL</td>
</tr>
<tr>
<td>0x02</td>
<td>WF_NOTIFY_CONNECTION_ATTEMPT_FAILED</td>
</tr>
<tr>
<td>0x04</td>
<td>WF_NOTIFY_CONNECTION_TEMPORARILY_LOST</td>
</tr>
<tr>
<td>0x08</td>
<td>WF_NOTIFY_CONNECTION_PERMANENTLY_LOST</td>
</tr>
<tr>
<td>0x10</td>
<td>WF_NOTIFY_CONNECTION_REESTABLISHED</td>
</tr>
<tr>
<td>0x1f</td>
<td>WF_NOTIFY_ALL_EVENTS</td>
</tr>
</tbody>
</table>

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventNotificationBitMask</td>
<td>Bit mask defining which events the host will be notified of.</td>
</tr>
</tbody>
</table>

Returns

None.
Remarks

None.
Wi-Fi Connection Manager

The connection manager uses the connection algorithm and one or more connection profiles to connect to a network.

2 options are offered

1. Connection Manager Handled in Host Stack Software

MRF24W FW has a built-in connection manager, and it is enabled by default. If the host stack software is developed to have its own independent connection manager, the MRF24W connection manager should be disabled to avoid some possible conflicts.

In WF_Config.h, enable definition #define DISABLE_MODULE_FW_CONNECT_MANAGER_IN_INFRASTRUCT

2 Wi-Fi APIs that are affected if MRF24W connection manager is not disabled

A) UINT16 WF_CMDisconnect(void)

B) UINT16 WF_Scan(UINT8 Cpld)

For MRF24WB0M with FW versions older than 0x120C, the potential conflict between the 2 connection managers in host stack software and MRF24W firmware can cause fatal FW crash in MRF24WB FW.

2. Connection Manager Handled Entirely by MRF24W FW

Utilizes MRF24W FW built-in connection manager. This is enabled by default.

In WF_Config.h, make sure to disable definition #define DISABLE_MODULE_FW_CONNECT_MANAGER_IN_INFRASTRUCT

Module
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection Manager Public Members</td>
<td>Functions and variables accessible by the stack application</td>
</tr>
</tbody>
</table>

Wi-Fi API > Wi-Fi Connection Manager
Connection Manager Public Members

The following functions and variables are available to the stack application.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_CMConnect</td>
<td>Commands the MRF24W to start a connection.</td>
</tr>
<tr>
<td>WF_CMDisconnect</td>
<td>Commands the MRF24W to close any open connections and/or to cease attempting to connect.</td>
</tr>
<tr>
<td>WF_CMGetConnectionState</td>
<td>Returns the current connection state. Caution when using WF_CMGetConnectionState, as it may cause redirects on iPhone iOS6.1, even though redirection on laptop is functional. Users are encouraged to use 1 profile ID for MRF24W based on v5 stack SW. This function is retained for backward compatibility. In v6 stack SW, we are keeping to 1 profile ID for MRF24W and changing stack SW to have capability to handle multiple profile IDs.</td>
</tr>
<tr>
<td>WF_CMGetConnectContext</td>
<td>Retrieves WF connection context for MRF24W</td>
</tr>
<tr>
<td>WF_CMCheckConnectionState</td>
<td>Returns the current connection state.</td>
</tr>
<tr>
<td>WF_DisableModuleConnectionManager</td>
<td>When compilation flag DISABLE_MODULE_FW_CONNECT_MANAGER is enabled, this will disable MRF24W connection manager.</td>
</tr>
</tbody>
</table>
WF_CMConnect Function

```c
void WF_CMConnect(
    UINT8 CpId
);
```

Description

Connection Manager Functions

Directs the Connection Manager to scan for and connect to a WiFi network. This function does not wait until the connection attempt is successful, but returns immediately. See WF_ProcessEvent 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.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CplId</td>
<td>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 WF_CM_CONNECT_USING_LIST then the connectionProfileList will be used to connect, starting with the first Connection Profile in the list.</td>
</tr>
</tbody>
</table>

Returns
None.

Remarks

None.

Wi-Fi API > Wi-Fi Connection Manager > Connection Manager Public Members > WF_CMConnect Function
WF_CMDisconnect Function

C

UINT16 WF_CMDisconnect();

Description

Directs the Connection Manager to close any open connection or connection attempt in progress. No further attempts to connect are taken until WF_CMConnect() is called.

Preconditions

MACInit must be called.

Returns

Operation results. Success or Failure

Remarks

Disconnection is allowed only in connected state. If MRF24W FW is in the midst of connection (or reconnection) process, then disconnect can hammer connection process, and furthermore it may cause fatal failure in MRF24W FW operation. To be safe to use disconnect, we strongly recommend the user to disable module FW connection manager by enabling #define DISABLE_MODULE_FW_CONNECT_MANAGER_IN_INFRASTRUCTURE in WF_Config.h
WF_CMGetConnectionState Function

C

```c
void WF_CMGetConnectionState(
    UINT8 * p_state,
    UINT8 * p_currentCpId
);
```

Description

Returns the current connection state.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_state</td>
<td>Pointer to location where connection state will be written</td>
</tr>
<tr>
<td>p_currentCpId</td>
<td>Pointer to location of current connection profile ID that is being queried.</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.

```c
extern UINT8 state_SavedBeforeScan, ID_SavedBeforeScan;
```
WF_CMGetConnectContext Function

C

```c
void WF_CMGetConnectContext(
    tWFConnectContext * p_ctx
);
```

Description

Retrieve connection context
Retrieves WF connection context for MRF24WG0MA/B

Preconditions

MACInit must be called first.

Returns

None.

Remarks

None.
WF_CMCheckConnectionState Function

```c
void WF_CMCheckConnectionState(
    UINT8 * p_state,
    UINT8 * p_currentCpId
);
```

Description

Returns the current connection state.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_state</td>
<td>Pointer to location where connection state will be written</td>
</tr>
<tr>
<td>p_currentCpId</td>
<td>Pointer to location of current connection profile ID that is being queried.</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_DisableModuleConnectionManager

Function

```c
void WF_DisableModuleConnectionManager();
```

Description

Disable MRF24W connection manager.

Preconditions

MACInit must be called first.

Returns

None.

Remarks

None.

Wi-Fi API > Wi-Fi Connection Manager > Connection Manager Public Members > WF_DisableModuleConnectionManager Function
Wi-Fi Scan

If the application already knows the network SSID that it wants to join, 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, adhoc or Wi-Fi Direct networks are in the area, and then decide which network to join. The scan API functions are used to gather this information.

There are 2 types of scan operations:

- **Active Scan**

  STA will transmit probe request frames and AP/routers will respond by transmitting probe response frames. For AP/Router with hidden SSID, active scan is used.

- **Passive Scan**

  AP/router will continuously transmit beacon frames for every beacon interval (BI), any STAs may/will receive these beacons and know of existence of this AP/router.

Scanning operation is tied to channel list (MY_DEFAULT_CHANNEL_LIST).

For example,

```
#define MY_DEFAULT_CHANNEL_LIST {1, 3, 6}
```

specifies scanning operation will be conducted in channels 1, 3, 6.

In FCC regional domain, there are channels 1 to 11. And among these channels, channel 1, 6 and 11 are defined as social channels.

Module

Wi-Fi API
### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scan Operation and Scan Results</strong></td>
<td>Scan operation and type of information contained in scan results</td>
</tr>
<tr>
<td><strong>Shorter Scan or Connection Duration</strong></td>
<td>Describes the factors influencing scan or connection duration</td>
</tr>
<tr>
<td><strong>Use of macro#define MY_DEFAULT_CHANNEL_LIST</strong></td>
<td>Describes the use of macro #define MY_DEFAULT_CHANNEL_LIST</td>
</tr>
<tr>
<td><strong>Maximum Scan Results</strong></td>
<td>Describes the maximum scan results supported by MRF24W RF modules</td>
</tr>
<tr>
<td><strong>Scan Public Members</strong></td>
<td>Functions and variables accessible by the stack application</td>
</tr>
</tbody>
</table>

**Wi-Fi API** > **Wi-Fi Scan**
Scan Operation and Scan Results

Scan operation and scan result

Refer to WiFi EZConfig demo for example on how scanning operation is initiated and how scan results are displayed.

Use function prototype `WF_Scan()` to invoke a scan operation.

When the scan results are ready, a `WF_EVENT_SCAN_RESULTS_READY` event will be generated.

Use function prototype `WF_ScanGetResult()` to retrieve scan result from the MRF24WB0M / MRF24WG0M.

Refer to data struct `tWFScanResult` (WFApi.h)

Each scan result returned by MRF24WB0M / MRF24WG0M RF module will contain

- **bssid** (Network BSSID value)
- **ssid** (Network SSID value)
- Access point configuration information such as security mode (WPA2, WPA, WEP, OPEN)

For WEP security mode, the scan results will NOT indicate whether it is `WF_SECURITY_WEP_SHAREDKEY` or `WF_SECURITY_WEP_OPENKEY`. Referring to 802.11 specifications, the beacon itself does not indicate this difference. The scan results reflect whatever information is contained in the beacons or probe responses.

- Access point beacon interval
- atimWindow (Applicable only for infrastructure network)
- List of network basic rates and number of valid rates in basic rates
- RSSI (Signal strength of received beacon or probe response frames)
- DtimPeriod
- bssType (WF_INFRASTRUCTURE, WF_ADHOC)
- channel (channel number in which beacon or probe response frame is received in)
- ssidLen (Length of SSID)

Remember scan results are retained on the MRF24W until:

1. Calling `WF_Scan()` again (after scan results returned from previous call).
2. Resetting MRF24W.

**Scan Results Showing Multiple Copies of the Same SSID**

If the same SSID is detected on different channels, the scan results may seem to show duplicate scan results. In reality, the scan results are showing the same SSID but on different channels. For example, SSID "MCHP_Network" is detected on channel 1, 6 and 11. Therefore the scan results will display among the scan results 3 copies of SSID "MCHP_Network".

**AP/Routers With Hidden SSID**

If an Access Point uses a hidden SSID, then an active scan must be used (see scanType field in the Connection Algorithm). In active scan, MRF24W will transmit a probe request frame and the AP with the hidden SSID will respond by transmitting a probe response frame.
Scan Cache

If necessary, a scan cache can be created such that scan results are retrieved and stored.

As an example, in WiFi EZConfig in SoftAP mode, upon power on reset, a scan operation is invoked and scan results are retrieved from MRF24WG0M and stored into global array (tWFScanResult preScanResult[50]).

After this, wifi startup in softAP mode is initiated via function prototype WF_Connect(). Within WF_Connect(), there are embedded scan operation, which will cause the scan results to be reset to default values.

Module

Wi-Fi Scan

Wi-Fi API > Wi-Fi Scan > Scan Operation and Scan Results
Shorter Scan or Connection Duration

Scan or Connection Duration

1. Channel selection

Within FCC domain, there are 11 channels.

Channel 1, 6, 11 overlaps and are designated social channels. The longer the scan channel list, the longer will be the scan and connection duration.

To scan all channel

- #define MY_DEFAULT_CHANNEL_LIST {}

To scan selected channels

- #define MY_DEFAULT_CHANNEL_LIST {1,6,11}

2. Delay timing

Within the stack software, there are numerous delays added in.

These selected delay time may be optimized or changed to use shorter delay time.

Example:

In WFEasyConfigProcess(), when softAP enters Hibernate mode, a delay is executed before exiting Hibernate mode.

DelayMs(50); // SOFTAP_ZEROCONF_SUPPORT. Timing reduced from 200 to 50.
Delay timing was originally set to 200msec and then optimized to 50msec.

The delay timing has not been tested on actual products and are just recommended values.

Module

Wi-Fi Scan

Wi-Fi API > Wi-Fi Scan > Shorter Scan or Connection Duration

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
Use of macro `#define MY_DEFAULT_CHANNEL_LIST`  

Macro `#define MY_DEFAULT_CHANNEL_LIST`

Scanning operation is tied to channel lists (for example, channel 1 to 11 for FCC regional domain).

For example, you can specify

- Scanning to be performed in selected channels such as channel 1, 3 and 5.

`#define MY_DEFAULT_CHANNEL_LIST {1, 3, 5}`

- Scanning to be performed in all channels.

`#define MY_DEFAULT_CHANNEL_LIST {}`

An exception to the use of `MY_DEFAULT_CHANNEL_LIST` is when MRF24W network type is `CFG_WF_SOFT_AP`.

`MY_DEFAULT_CHANNEL_LIST` will instead indicate the channel the MRF24W softAP will reside in.

For example, MRF24W softAP will reside in social channel 6.

`#define MY_DEFAULT_CHANNEL_LIST {6}`

To cater for MRF24W softAP network type, 2 more macros are defined

- `#define MY_DEFAULT_CHANNEL_LIST_PRESCAN`

Before MRF24W starts up as softAP, MRF24W will first perform a scanning operation. And `MY_DEFAULT_CHANNEL_LIST_PRESCAN` will indicate the channel list to be scanned before starting up as softAP.

- `#define MY_DEFAULT_CHANNEL_LIST_POSTSCAN`
When MRF24W softAP is redirected to an infrastructure or any non-softAP network types, MY_DEFAULT_CHANNEL_LIST_POSTSCAN is used to indicate the channel list to be scanned in these non-softAP network types.

Module

Wi-Fi Scan

Wi-Fi API > Wi-Fi Scan > Use of macro #define MY_DEFAULT_CHANNEL_LIST
Maximum Scan Results

Maximum Scan Results Supported

Both MRF24WB0M and MRF24WG0M support up to a maximum of 60 scan results.

Module

Wi-Fi Scan

Wi-Fi API > Wi-Fi Scan > Maximum Scan Results
Scan Public Members

The following functions and variables are available to the stack application.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_Scan</td>
<td>Commands the MRF24W to start a scan operation. This will generate the WF_EVENT_SCAN_RESULTS_READY event.</td>
</tr>
<tr>
<td>WF_ScanGetResult</td>
<td>Read scan results back from MRF24W.</td>
</tr>
</tbody>
</table>

Module

Wi-Fi Scan

Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tWFScanResult</td>
<td>Scan Results</td>
</tr>
</tbody>
</table>

Wi-Fi API > Wi-Fi Scan > Scan Public Members
WF_Scan Function

C

```c
UINT16 WF_Scan(
    UINT8 CpId
);
```

Description

Scan Functions

Directs the MRF24W 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().

Scan results are retained on the MRF24W until:

1. Calling WF_Scan() again (after scan results returned from previous call).
2. MRF24W reset.

MRF24WB0M & MRF24WG0M support up to max of 60 scan results (SSIDs).

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile to use. If the CpId is valid then the values from that Connection Profile will be used for filtering</td>
</tr>
</tbody>
</table>
scan results. If the Cpld is set to WF_SCAN_ALL (0xFF) then a default filter will be used.

Valid Cpld

- If CP has a defined SSID only scan results with that SSID are retained.
- If CP does not have a defined SSID then all scanned SSID's will be retained
- Only scan results from Infrastructure or AdHoc networks are retained, depending on the value of networkType in the Connection Profile
- The channel list that is scanned will be determined from channelList in the Connection Algorithm (which must be defined before calling this function).

Cpld is equal to WF_SCAN_ALL

- All scan results are retained (both Infrastructure and Ad Hoc networks).
- All channels within the MRF24W's regional domain will be scanned.
- No Connection Profiles need to be defined before calling this function.
- The Connection Algorithm does not need to be defined before calling this function.

Returns

Operation results. Success or Failure

Remarks

Host scan is allowed only in idle or connected state. If MRF24W FW is
in the midst of connection (or reconnection) process, then host scan can hammer connection process, and furthermore it may cause fatal failure in MRF24W FW operation. To use host scan, we strongly recommend the user to disable MRF24W FW connection manager by enabling #define DISABLE_MODULE_FW_CONNECT_MANAGER_IN_INFRASTRUCTURE in WF_Config.h
WF_ScanGetResult Function

C

```c
void WF_ScanGetResult(
    UINT8 listIndex,
    tWFScanResult * p_scanResult
);
```

Description

After a scan has completed this function is used to read one or more of the scan results from the MRF24W. The scan results will be written contiguously starting at p_scanResult (see `tWFScanResult` structure for format of scan result). MRF24WB0M & MRF24WG0M support up to max of 60 scan results (SSIDs).

Preconditions

MACInit must be called first. WF_EVENT_SCAN_RESULTS_READY event must have already occurred.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listIndex</td>
<td>Index (0-based list) of the scan entry to retrieve.</td>
</tr>
<tr>
<td>p_scanResult</td>
<td>Pointer to location to store the scan result structure</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

RSSI can only be obtained from the scan results p_scanResult->rssi. MRF24W checks out the signal strength from the preamble of the incoming packets. The higher the values, the stronger is the received
signal strength.

p_scanResult->rssi contains received signal strength indicator (RSSI).

- MRF24WB : RSSI_MAX (200) , RSSI_MIN (106).
- MRF24WG : RSSI_MAX (128) , RSSI_MIN (43).

The RSSI value is not directly translated to dbm because this is not calibrated number. However, as a guideline, MAX(200) corresponds to 0 dbm, MIN (106) corresponds to -94 dbm.
### tWFScanResult Structure

```c
typedef struct {
    UINT8 bssid[WF_BSSID_LENGTH];
    UINT8 ssid[WF_MAX_SSID_LENGTH];
    UINT8 apConfig;
    UINT8 reserved;
    UINT16 beaconPeriod;
    UINT16 atimWindow;
    UINT8 basicRateSet[WF_MAX_NUM_RATES];
    UINT8 rssi;
    UINT8 numRates;
    UINT8 DtimPeriod;
    UINT8 bssType;
    UINT8 channel;
    UINT8 ssidLen;
} tWFScanResult;
```

### Description

Scan Results

### Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8 bssid[WF_BSSID_LENGTH];</td>
<td>Network BSSID value</td>
</tr>
<tr>
<td>UINT8 ssid[WF_MAX_SSID_LENGTH];</td>
<td>Network SSID value</td>
</tr>
<tr>
<td>UINT8 apConfig;</td>
<td>Access point configuration</td>
</tr>
<tr>
<td>UINT16 beaconPeriod;</td>
<td>Network beacon interval</td>
</tr>
<tr>
<td>UINT16 atimWindow;</td>
<td>Only valid if bssType = WF_INFRASTRUCTURE</td>
</tr>
<tr>
<td></td>
<td>List of Network basic rates. Each rate has the following format:</td>
</tr>
<tr>
<td></td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td>• 0 – rate is not part of the basic rates set</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>UINT8 basicRateSet[W_F_MAX_NUM_RATES];</td>
<td>○ 1 – rate is part of the basic rates set</td>
</tr>
<tr>
<td>Bits 6:0</td>
<td>Multiple of 500kbps giving the supported rate. For example, a value of 2 (2 * 500kbps) indicates that 1mbps is a supported rate. A value of 4 in this field indicates a 2mbps rate (4 * 500kbps).</td>
</tr>
<tr>
<td>UINT8 rssi;</td>
<td>Signal strength of received frame beacon or probe response</td>
</tr>
<tr>
<td>UINT8 numRates;</td>
<td>Number of valid rates in basicRates</td>
</tr>
<tr>
<td>UINT8 DtimPeriod;</td>
<td>Part of TIM element</td>
</tr>
<tr>
<td>UINT8 bssType;</td>
<td>WF_INFRASTRUCTURE or WF_ADHOC</td>
</tr>
<tr>
<td>UINT8 channel;</td>
<td>Channel number</td>
</tr>
<tr>
<td>UINT8 ssidLen;</td>
<td>Number of valid characters in ssid</td>
</tr>
</tbody>
</table>

Wi-Fi API > Wi-Fi Scan > Scan Public Members > tWFScanResult Structure
Wi-Fi Security

This section elaborates on the various security modes supported by MRF24WB0M and MRF24WG0M.

Security modes supported in

- Ad-hoc
  OPEN, WEP
- SoftAP
  OPEN, WEP
- Infrastructure
  OPEN, WEP, WPA/WPA2, WPS
- Wi-Fi Direct
  WPS
Module

Wi-Fi API

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wired Equivalent Privacy (WEP)</td>
<td>Describes the Wi-Fi security modes Wired Equivalent Privacy (WEP)</td>
</tr>
<tr>
<td>Wi-Fi Protected Access (WPA/WPA2)</td>
<td>Describes the Wi-Fi security modes Wi-Fi Protected Access (WPA/WPA2)</td>
</tr>
<tr>
<td>Wi-Fi Protected Setup (WPS)</td>
<td>Describes the Wi-Fi security connection modes Wi-Fi Protected Setup (WPS). WPS security mode is only supported by MLA v5.42 July 2012 releases or later. Only MRF24WG0M supports WPS security mode.</td>
</tr>
<tr>
<td>WPA2 Enterprise</td>
<td>Describes the Wi-Fi security modes WPA2 Enterprise</td>
</tr>
</tbody>
</table>

---

**Wi-Fi API > Wi-Fi Security**

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
Wired Equivalent Privacy (WEP)

WIRED EQUIVALENT PRIVACY (WEP)

Security loop holes are present and this is not recommend!!!

WEP security modes supported are

- **WF_SECURITY_WEP_40**
  WEP Encryption using 40 bit keys. Also known as WEP 64 keys.
  This security method uses a 40 bit (10 Hex character) "secret key" and a 24 bit Initialization Vector (IV).

- **WF_SECURITY_WEP_104**
  WEP Encryption using 104 bit keys Also known as WEP 128 keys.
  This security method uses a 104 bit (26 Hex Character) "secret key", and a 24 bit Initialization Vector (IV).

MRF24W only accepts WEP hex keys
(MY_DEFAULT_WEP KEYS_40 or MY_DEFAULT_WEP KEYS_104) and NOT passphrase. Some Web sites offer this automatic WEP key generators, whereby WEP keys are generated from ordinary text called a passphrase.

WEP key types supported are

- **WF_SECURITY_WEP_OPENKEY**
  Default
  Both AP/router & client STA do not use the key during the connection process (authentication and association). Thus the connection process is exactly the same as open mode. Once the connection is established, then both AP/router and client STA can start to use the key to encrypt
the data packets. For the case when the key mismatches, connection can still be established. However, it will fail during operations such as DHCP, etc.

- **WF_SECURITY_WEP_SHAREDKEY**

Supported by MRF24WG0M.

Supported by MRF24WB0M (RF module FW version 0x1209 and later)

This involves the key during the authentication process. When the key mismatches, this will cause the connection process to fail.

According to 802.11 specifications, WEP can have a total of 4 keys (or 4 key indices). However, in commercial products, only 1 key index (0) is really used. As an example, for iOS, Android and even Windows, there is probably no option to choose WEP key index. That implies they are using only 1 WEP key index.

**Functions**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WF_CPGetWepKeyType</strong></td>
<td>Gets the WEP key type for the specified Connection Profile ID.</td>
</tr>
<tr>
<td><strong>WF_CPGetDefaultWepKeyIndex</strong></td>
<td>Returns the value of the active WEP keys to use. Only WEP key index 0 is used in RF module FW.</td>
</tr>
<tr>
<td><strong>WF_CPSetDefaultWepKeyIndex</strong></td>
<td>Set up the WEP key index. Only WEP key index 0 is used in RF module FW.</td>
</tr>
<tr>
<td><strong>WF_CPSetWepKeyType</strong></td>
<td>Sets the WEP key type for the specified Connection Profile ID.</td>
</tr>
</tbody>
</table>

Wi-Fi API > Wi-Fi Security > Wired Equivalent Privacy (WEP)
WF_CPGetWepKeyType Function

```c
void WF_CPGetWepKeyType(
    UINT8 CpId,
    UINT8 * p_wepKeyType
);
```

**Description**

Gets the WEP key type element for the specified Connection Profile. Allowable values are:

- WF_SECURITY_WEP_SHAREDKEY (0)
- WF_SECURITY_WEP_OPENKEY (1) - Default

** Preconditions **

MACInit must be called first.

** Returns **

None.

** Remarks **

Before MRF24W is connected to the AP/Router, calling this function will always return WF_SECURITY_WEP_OPENKEY (1) as the default.
WF_CPGetDefaultWepKeyIndex Function

C

```c
void WF_CPGetDefaultWepKeyIndex(
    UINT8 CpId,
    UINT8 * p_defaultWepKeyIndex
);
```

Description

Only applicable if the Connection Profile security type is either WF_SECURITY_WEP_40 or WF_SECURITY_WEP_104. Since only WEP key index 0 is supported, this function is no longer needed.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>p_defaultWepKeyIndex</td>
<td>Pointer to index of WEP key to use.</td>
</tr>
<tr>
<td>No longer supporting 4 WEP key indexes (0-3).</td>
<td></td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

Note that only WEP key index 0 amongst AP manufacturers is typically used. Using any of the other three keys may be unpredictable from brand to brand.
WF_CPSetDefaultWepKeyIndex Function

C

```c
void WF_CPSetDefaultWepKeyIndex(
    UINT8 CpId,
    UINT8 defaultWepKeyIndex
);
```

Description

Only applicable if the Connection Profile security type is either WF_SECURITY_WEP_40 or WF_SECURITY_WEP_104. Since only WEP key index 0 is supported, this function is no longer needed.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>defaultWepKeyIndex</td>
<td>Use WEP key index 0. No longer supporting 4 WEP key indexes (0 - 3).</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

Note that only WEP key index 0 amongst AP manufacturers is typically used. Using any of the other three keys may be unpredictable from brand to brand.
WF_CPSetWepKeyType Function

```c
void WF_CPSetWepKeyType(
    UINT8 CpId,
    UINT8 wKeyTyp
);
```

Description

Sets the WEP key type (WF_SECURITY_WEP_SHAREDKEY or WF_SECURITY_WEP_OPENKEY) for the specified Connection Profile ID. Default WEP key type is WF_SECURITY_WEP_OPENKEY.

Preconditions

MACInit must be called first.

Returns

None.

Remarks

None.

Wi-Fi API > Wi-Fi Security > Wired Equivalent Privacy (WEP) > WF_CPSetWepKeyType Function
Wi-Fi Protected Access (WPA/WPA2)

Wi-Fi PROTECTED ACCESS (WPA/WPA2)

Refer to the Wi-Fi Protected Access (WPA) Enhanced Security Implementation Based on IEEE P802.11i standard.

Other equivalent WPA/WPA2 terminologies used are WPA - 4 way handshake or EAPOL - 4 way handshake or 802.1X authentication.

Upon initial connection, after authentication followed by association, WPA/WPA2 EAPOL 4-way handshaking will takes place.

WPA/WPA2 security modes supported are

- **WF_SECURITY_WPA_WITH_KEY**
  WPA-PSK Personal where binary key is given to MRF24W

- **WF_SECURITY_WPA_WITH_PASS_PHRASE**
  WPA-PSK Personal where passphrase is given to MRF24W and it calculates the binary key

- **WF_SECURITY_WPA2_WITH_KEY**
  WPA2-PSK Personal where binary key is given to MRF24W

- **WF_SECURITY_WPA2_WITH_PASS_PHRASE**
  WPA2-PSK Personal where passphrase is given to MRF24W and it calculates the binary key

- **WF_SECURITY_WPA_AUTO_WITH_KEY**
  WPA-PSK Personal or WPA2-PSK Personal where binary key is given and MRF24W will connect at highest level AP supports (WPA or WPA2)

- **WF_SECURITY_WPA_AUTO_WITH_PASS_PHRASE**
WPA-PSK Personal or WPA2-PSK Personal where passphrase is given to MRF24W and it calculates the binary key and connects at highest level AP supports (WPA or WPA2).

MRF24W can be configured to accept either binary key or passphrase, through MY_DEFAULT_WIFI_SECURITY_MODE compile-time definition. The WPA/WPA2 authentication process involves key derivation from the given passphrase.

Since this key derivation process is computational intensive and requires memory space, options are given

- **Host to derive the key from the passphrase**

DERIVE_KEY_FROM_PASSPHRASE_IN_HOST needs to be enabled. Refer to function prototype `pbkdf2_sha1()` in WF_pbkdf2.c for the algorithm.

In this scenario, MRF24W will generate an event (WF_EVENT_KEY_CALCULATION_REQUEST) to host (eg PIC32) and set g_WpsPassphrase.valid to TRUE. Upon receipt of this event, the host will execute the function `WF_ConvPassphrase2Key()` to convert the passphrase to key. Upon completion of this conversion, the host will call `WF_SetPSK()` to pass the converted key to MRF24W.

- **MRF24W will handle the entire WPA/WPA2 passphase and then key derivation**

MRF24WB0M consumes about ~32 seconds whereas MRF24WG0M will consume about ~25 seconds.

**Functions**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_CPUUpdatePMK</td>
<td>Updates the PMK element in the connection profile ID retrieved out of 802.1x authentication process</td>
</tr>
<tr>
<td>WF_ConvPassphrase2Key</td>
<td>Allow host to convert passphrase to key</td>
</tr>
</tbody>
</table>
This is function pbkdf2_sha1.
WF_CPUpdatePMK Function

```c
void WF_CPUpdatePMK(  
    UINT8 CpId,  
    UINT8 * pmk 
);
```

Description

Applicable for MRF24WG0M only. Updates the PMK element in the connection profile ID retrieved out of 802.1x authentication process. This function is not used in the TCPIP demo apps.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>pmk</td>
<td>pairwise master key</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_ConvPassphrase2Key Function

C

```c
void WF_ConvPassphrase2Key(
    UINT8  key_len,
    UINT8 * key,
    UINT8 ssid_len,
    UINT8 * ssid
);
```

Description

DERIVE_KEY_FROM_PASSPHRASE_IN_HOST and __C32__ must be enabled. This function is called in WPS or WPA or WPA2 security modes. Convert WPS/WPA/WPA2 passphrase to key. Allows host (eg PIC32) to perform conversion of the passphrase to the key by itself instead of relying on RF module FW. This is recommended only for PIC microprocessors that have high computational bandwidth and sufficient memory space. As a benchmark, MRF24WB0M will take 32 sec and MRF24WG0M will take 25 sec for this computation.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_len</td>
<td>key length</td>
</tr>
<tr>
<td>key</td>
<td>passphrase as an input. key as an output</td>
</tr>
<tr>
<td>ssid_len</td>
<td>ssid length</td>
</tr>
<tr>
<td>ssid</td>
<td>ssid</td>
</tr>
</tbody>
</table>

Returns
Remarks

MRF24W will generate an event (WF EVENT KEY CALCULATION REQUEST) to PIC32 and set g_WpsPassphrase.valid to TRUE. Upon receipt of this event, PIC32 will invoke this function WF_ConvPassphrase2Key to convert the passphrase to key. Once this conversion is completed, PIC32 will call WF_SetPSK to pass the converted key to MRF24W.
pbkdf2_sha1 Function

```c
void pbkdf2_sha1(
    const char * passphrase,
    const char * ssid,
    UINT16 ssid_len,
    UINT16 iterations,
    UINT8 * buf,
    UINT16 buflen
);
```

Description

This is function pbkdf2_sha1.
Wi-Fi Protected Setup (WPS)

Wi-Fi PROTECTED SETUP (WPS)

Refer to the Wi-Fi Protected Setup Specification Version 1.0h standard.

WiFi Protected Setup (WPS) allows users to set up and expand the WiFi networks with security enabled, even if they are not familiar with the underlying technologies or processes involved. For example, users no longer have to know that SSID refers to the network name or WPA2 refers to the security mechanism.

WPS does not support ad hoc networks.

WPS will configure the network name SSID and security key for the AP and WPS client devices on a network. It supports the WEP / WPA / WPA2 security methods.

2 methods are supported

- **WPS-PBC (Push Button Configuration)**

  Users can connect the device (MRF24WG0M) to the network and enable data encryption by pushing the buttons on the AP and client device. Users do NOT need to know the SSID of the AP, however the users are required to be within close proximity of the AP to press the push button on the AP.

- **WPS-PIN (Personal Information Number)**

  PIN is provided for each device which joins the network. Enter this PIN on the AP/Router (Registrar) and activate AP (Registrar) first before MRF24WG0M attempts to connect. The last digit of the PIN is the checksum of the first 7 digits of the PIN. This checksum must be correct, otherwise MRF24WG0M module will reject the PIN code.

To set up WPS-PBC, define
• MY_DEFAULT_NETWORK_TYPE as CFG_WF_INFRASTRUCTURE
• MY_DEFAULT_SSID_NAME as ""
• MY_DEFAULT_WIFI_SECURITY_MODE as WF_SECURITY_WPS_PUSH_BUTTON

To set up WPS-PIN, define

• MY_DEFAULT_NETWORK_TYPE as CFG_WF_INFRASTRUCTURE
• MY_DEFAULT_SSID_NAME to be the same SSSID as the AP/router
• MY_DEFAULT_WIFI_SECURITY_MODE as WF_SECURITY_WPS_PIN
• MY_DEFAULT_WPS_PIN to be the same as the AP/router PIN.

WPS protocol can be viewed as a security connection method, built upon the existing security modes WPA/WPA2. The protocol encompasses a M1-M8 message exchange process. Therefore additional time is consumed. The WPS specification specifies below:

1. Retransmission timeout = 5 sec
2. Individual message processing timeout = 15 sec
3. Overall timeout for entire protocol to complete = 2 min

To address this lengthy time required every single time the MRF24W is restarted, the function prototype WF_SaveWPSCredentials() can be invoked. Basically what this function does is to retrieve WPS credentials from MRF24W and store these into global variable AppConfig. In this way, it makes the WPS credentials re-useable and shorten subsequent reconnection time. For an actual product, it is advised to add in a timeout such that the WPS credentials will need to be refreshed or updated periodically.
WPS protocol

The WPS protocol, which encompass WPA/WPA2 authentication process, involves key derivation from the given passphrase. Since this key derivation process is computational intensive and requires memory space, options are given

- **Host to derive the key from the passphrase**

DERIVE_KEY_FROM_PASSPHRASE_IN_HOST needs to be enabled. Refer to function prototype `pbkdf2_sha1()` in WF_pbkdf2.c for the algorithm.

In this scenario, `WF_YieldPassphrase2Host()` will inform MRF24W that host wants to do conversion. MRF24W will generate an event (WF_EVENT_KEY_CALCULATION_REQUEST) to host (eg PIC32) and set g_WpsPassphrase.valid to TRUE. Upon receipt of this event, the host will execute the function `WF_ConvPassphrase2Key()` to convert the passphrase to key. Upon completion of this conversion, the host will call `WF_SetPSK()` to pass the converted key to MRF24W.

- **MRF24W will handle the entire WPA/WPA2 passphrase and then key derivation**

MRF24WG0M will consume about ~25 seconds.

Upon initial connection (after authentication followed by association), WPS will take place and is performed over 2 phases.

**Phase A: WPS Frame Exchanges (EAP authentication protocol)**

Refer to Wi-Fi Simple Configuration Specification Version 2.0.2 or Wi-Fi Protected Setup Specification Version 1.0h “Figure 4: In-band Setup Using a Standalone AP/Registrar”.

Information and network credentials, such as SSID, security mode, security keys, security passphrase, etc, are securely exchanged over the air using the Extensible Authentication Protocol (EAP). From these WPS frame exchanges, WPS will automatically configure the network
connection, without having the user to know the SSID and security keys or passphrases, etc.

The enrollee is defined as a device seeking to join a wireless network and is represented by MRF24WG0M. In the infrastructure network, the enrollee and wireless client are synonymous.

WSC IE (Wi-Fi Simple Configuration Information Element) will be present in the beacons, probe request/responses frames and association request/response frames. Refer to Wi-Fi Simple Configuration Specification for details on the WSC IE contents.
**Phase B: EAPOL 4-way Handshake or 802.1X-authentication**

Refer to specifications IEEE 802.11i-2004: Amendment 6: Medium Access Control (MAC) Security Enhancements.

After a successful EAP authentication, the EAPOL 4-way handshake begins.

The supplicant, STA and wireless client are synonymous.

Likewise, the authenticator and AP/router are synonymous.

The 4-way handshake shares unique random information between the supplicant/client and the authenticator /AP to derive the PTK key.

Below is a brief description of the EAPOL 4-way handshake

**4-way handshake message 1**

The AP/router sends the STA a nonce (ANonce). Along with this ANonce, the frame includes the AP/router MAC address. At this point the STA has all the information needed to create the PTK; Anonce, AP/router MAC address, its own Snonce and MAC address.

**4-way handshake message 2**

The STA creates its nonce (SNonce). The STA can now calculate the PTK because it has all the information from the first handshake. In this second message, the STA sends the SNonce to the AP/router. The STA also sends the security parameters (RSN) information. The entire message gets an authentication check using the (MIC) from the pairwise key hierarchy. The AP/router can then verify that the information, including the security parameters sent at association is valid.

**4-way handshake message 3**

In this third message, the AP/router derives the GTK key from the GMK
key. The AP/router derives an ANonce, RSN information element info and a MIC and sends these information to the STA in an EAPOL-Key frame. This is kept secret from sniffing since it is encrypted within the PTK.

4-way handshake message 4

The fourth message acts as a confirmation. It indicates that the temporal keys are installed.

Below shows the WPS protocol
Integration of WPS into 802.11 joining operation

Below lists the overall sequences

- Scanning, **Authentication**, Association
- WPS Frame Exchanges (EAP protocol)
- Deauthentication (Refer to above figure) or Disassociation

Some APs are found to transmit disassociation instead of deauthentication frame.

Provision needs to be made to handle receipt of disassociation frame.

- **Authentication**, Association
- EAPOL 4-way handshake or 802.1X-authentication

**Functions**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_CPGetWPSCredentials</td>
<td>Gets the WPS credentials for the specified Connection Profile ID for MRF24WG0MA/B.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WF_YieldPassphrase2Host</td>
<td>Allows host to handle WPS WPA/WPA2-PSK passphrase</td>
</tr>
<tr>
<td>WF_SetPSK</td>
<td>Sends PSK to module FW in WPS mode.</td>
</tr>
<tr>
<td>WF_SaveWPSCredentials</td>
<td>Stores WPS credentials into global variable AppConfig</td>
</tr>
<tr>
<td>ConfigWep</td>
<td>Configures WEP security mode from WPS credentials data</td>
</tr>
</tbody>
</table>

**Structures**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tWFpsCred</td>
<td>This is type tWFpsCred.</td>
</tr>
</tbody>
</table>

*Wi-Fi API > Wi-Fi Security > Wi-Fi Protected Setup (WPS)*

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
WF_CPGetWPSCredentials Function

C

```c
void WF_CPGetWPSCredentials(
    UINT8 CpId,
    tWFwpsCred * p_cred
);
```

Description

Applicable for MRF24WG0M only. Gets the WPS credentials after WPS completed for MRF24WG0MA/B.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
<tr>
<td>p_cred</td>
<td>Pointer to the credentials</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_YieldPassphrase2Host Function

C

```c
void WF_YieldPassphrase2Host();
```

Description

WPS supporting Functions

DERIVE_KEY_FROM_PASSPHRASE_IN_HOST must be enabled. Applicable for MRF24WG0M only. This function is used only for WF_SECURITY_WPS_PUSH_BUTTON and WF_SECURITY_WPS_PIN security mode. Allows host (eg PIC32) to convert pass phrase to key in WPS WPA/WPA2-PSK. The SW process flow is like this: WF_YieldPassphrase2Host() will inform MRF24W that host wants to do conversion. MRF24W will then generate an event (WF_EVENT_KEY_CALCULATION_REQUEST) to host and set g_WpsPassphrase.valid to TRUE. Upon receipt of this event, the host will execute the function `WF_ConvPassphrase2Key()` to convert the passphrase to key. Upon completion of this conversion, the host will call `WF_SetPSK()` to pass the converted key to MRF24W.

Preconditions

MACInit must be called first.

Returns

None.

Remarks

None.
WF_SetPSK Function

C

```c
void WF_SetPSK(
    UINT8 * psk
);
```

Description

DERIVE_KEY_FROM_PASSPHRASE_IN_HOST must be enabled. Applicable for MRF24WG0M only. Sends PSK to MRF24W FW in WPS mode.

Preconditions

MACInit must be called first.

Returns

None.

Remarks

This function is used in conjunction with WF_ConvPassphrase2Key(). MRF24W will generate an event (WF_EVENT_KEY_CALCULATION_REQUEST) to PIC32 and set g_WpsPassphrase.valid to TRUE. Upon receipt of this event, PIC32 will invoke this function WF_ConvPassphrase2Key to convert the passphrase to key. Once this conversion is completed, PIC32 will call WF_SetPSK to pass the converted key to MRF24W.
WF_SaveWPSCredentials Function

C

```
static void WF_SaveWPSCredentials(
    UINT8 CpId
);
```

Description

Reads back WPS credentials from MRF24W and stores these into global variable AppConfig. WPS protocol can easily take up to 2 mins (refer to WPS specifications) to complete. To address this lengthy time required every single time the MRF24W is restarted, this function offers you to retrieve and store WPS credentials, so that the WPS credentials can be re-used upon re-starting.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpId</td>
<td>Connection Profile ID</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
### ConfigWep Function

```c
static void ConfigWep(
    twFwpwspCred * cred,
    UINT8 * secType,
    union sec_key * key
);
```

#### Description

Configures WEP security mode from WPS credentials data. Based on key length, determine whether it is WF_SECURITY_WEP_40 or WF_SECURITY_WEP_104. Perform a key conversion to hex key values.

#### Preconditions

MACInit must be called first.

#### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*cred</td>
<td>pointer to WPS credentials</td>
</tr>
<tr>
<td>*secType</td>
<td>pointer to security mode type</td>
</tr>
<tr>
<td>*key</td>
<td>pointer to security key</td>
</tr>
</tbody>
</table>

#### Returns

None.

#### Remarks

None.
C
t

typedef struct {
    UINT8 ssid[32];
    UINT8 netKey[64];
    UINT16 authType;
    UINT16 encType;
    UINT8 netIdx;
    UINT8 ssidLen;
    UINT8 keyIdx;
    UINT8 keyLen;
    UINT8 bssid[6];
} tWFWpsCred;

Description

This is type tWFWpsCred.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8 ssid[32];</td>
<td>SSID</td>
</tr>
<tr>
<td>UINT8 netKey[64];</td>
<td>Net Key PSK</td>
</tr>
<tr>
<td>UINT16 authType;</td>
<td>Authentication Type: AUTH_OPEN / AUTH_WPA_PSK / AUTH_SHARED / AUTH_WPA / AUTH_WPA2 / AUTH_WPA2_PSK</td>
</tr>
<tr>
<td>UINT16 encType;</td>
<td>Encoding Type: ENC_NONE / ENC_WEP / ENC_TKIP / ENC_AES</td>
</tr>
<tr>
<td>UINT8 netIdx;</td>
<td>Net ID</td>
</tr>
<tr>
<td>UINT8 ssidLen;</td>
<td>SSID length</td>
</tr>
<tr>
<td>UINT8 keyIdx;</td>
<td>Key ID</td>
</tr>
<tr>
<td>UINT8 keyLen;</td>
<td>WPA/WPA2-PSK key length</td>
</tr>
<tr>
<td>UINT8 bssid[6];</td>
<td>BSSID</td>
</tr>
</tbody>
</table>
WPA2 Enterprise

Supported by MRF24WG0M only.

Needs a special MRF24WG0M FW version. Contact your local Microchip sales office.

Requires MLA v5.42.06 March 2013 release and future releases.

This security mode WPS Enterprise is used in WiFi Console. The security types supported are

- EAP-PEAP/MSCHAPv2
- EAP-TTLS/MSCHAPv2
Wi-Fi Tx Power Control

The API functions in this section are used to configure the MRF24WB0M / MRF24WG0M transmit (Tx) power control settings.

MRF24WB0M transmit power settings are from -10dBm to +10dBm.
MRF24WG0M transmit power settings are from 0dBm to +18dBm.

Module

Wi-Fi API

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tx Power Control Public Members</td>
<td>Functions and variables accessible by the stack application</td>
</tr>
</tbody>
</table>

Wi-Fi API > Wi-Fi Tx Power Control
Tx Power Control Public Members

The following functions and variables are available to the stack application.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_TxPowerGetMinMax</td>
<td>Gets the Tx min and max power on the MRF24WB0MA/B.</td>
</tr>
<tr>
<td>WF_TxPowerSetMinMax</td>
<td>Sets the Tx min and max power on the MRF24WB0MA/B.</td>
</tr>
<tr>
<td>WF_TxPowerGetFactoryMax</td>
<td>Retrieves the factory-set max Tx power from the MRF24WB0MA/B and MRF24WG0MA/B.</td>
</tr>
<tr>
<td>WF_TxPowerGetMax</td>
<td>Gets the Tx max power on the MRF24WG0MA/B.</td>
</tr>
<tr>
<td>WF_TxPowerSetMax</td>
<td>Sets the Tx max power on the MRF24WG0MA/B.</td>
</tr>
<tr>
<td>WF_FixTxRateWithMaxPower</td>
<td>Fix transmission rate with maximum power for MRF24WB0MA/B.</td>
</tr>
</tbody>
</table>

Module

**Wi-Fi Tx Power Control**

Wi-Fi API > Wi-Fi Tx Power Control > Tx Power Control Public Members
WF_TxPowerGetMinMax Function

C

```c
void WF_TxPowerGetMinMax(
    INT8  * p_minTxPower,
    INT8  * p_maxTxPower
);
```

Description

After initialization the MRF24WB0MA/B 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()`.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_minTxPower</td>
<td>Pointer to location to write the minTxPower</td>
</tr>
<tr>
<td>p_maxTxPower</td>
<td>Pointer to location to write the maxTxPower</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

No conversion of units needed, input to MRF24WB0MA/B is in dB.
WF_TxPowerSetMinMax Function

C

```c
void WF_TxPowerSetMinMax(
    INT8 minTxPower,
    INT8 maxTxPower
);
```

Description

After initialization the MRF24WB0MA/B 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()`.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minTxPower</td>
<td>Desired minTxPower (-10 to 10dB)</td>
</tr>
<tr>
<td>maxTxPower</td>
<td>Desired maxTxPower (-10 to 10dB)</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

No conversion of units needed, input to MRF24WB0MA/B is in dB.
WF_TxPowerGetFactoryMax Function

C

```c
void WF_TxPowerGetFactoryMax(
    INT8 * p_factoryMaxTxPower
);
```

Description

For MRF24WB, desired maxTxPower (-10 to 10 dBm), in 1dB steps
For MRF24WG, desired maxTxPower (0 to 18 dBm), in 1dB steps

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_factoryMaxTxPower</td>
<td>for MRF24WB, desired maxTxPower (-10 to 10 dBm), in 1dB steps for MRF24WG, desired maxTxPower (0 to 18 dBm), in 1dB steps</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.

Wi-Fi API > Wi-Fi Tx Power Control > Tx Power Control Public Members > WF_TxPowerGetFactoryMax Function
WF_TxPowerGetMax Function

C

```c
void WF_TxPowerGetMax(
    INT8 * p_maxTxPower
);
```

Description

After initialization the MRF24WG0MA/B max Tx power is determined by a factory-set value. This function can set a different 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()`.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_maxTxPower</td>
<td>Pointer to location to write the maxTxPower</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

No conversion of units needed, input to MRF24WG0MA/B is in dBm.
WF.TxPowerSetMax Function

C

```c
void WF_TxPowerSetMax(
    INT8 maxTxPower
);
```

Description

Tx Power Control Functions

After initialization the MRF24WG0MA/B max Tx power is determined by a factory-set value. This function can set a different 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().

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxTxPower</td>
<td>valid range (0 to 18 dBm)</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

No conversion of units needed, input to MRF24WG0MA/B is in dBm.
WF_FixTxRateWithMaxPower Function

C

```c
void WF_FixTxRateWithMaxPower(
    BOOL oneMegaBps
);
```

Description

Fix transmission rate with maximum power for MRF24WB0MA/B.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oneMegaBps</td>
<td>When true, that is 1 mbps. Otherwise 2 mbps</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

This function can be used by MRF24WG0M to fix transmission rates too. The input parameter is 2 * rate. Eg for 11Mbps, set input parameter as 22.

Wi-Fi API > Wi-Fi Tx Power Control > Tx Power Control Public Members > WF_FixTxRateWithMaxPower Function
Wi-Fi Power Save

The MRF24WB0M / MRF24WG0M supports two power-saving modes – sleep and hibernate.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleep</td>
<td>This mode is used when in PS Poll mode where the MRF24WB0M / MRF24WG0M 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 / MRF24WG0M 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.</td>
</tr>
<tr>
<td>Hibernate</td>
<td>This mode effectively turns off the LDO of the MRF24WB0M / MRF24WG0M for maximum power savings. MRF24WB0M / MRF24WG0M state is not retained, and when the MRF24WB0M / MRF24WG0M is taken out of the Hibernate state it performs a reboot. Hibernate mode is controlled by toggling the HIBERNATE pin on the MRF24WB0M / MRF24WG0M module (high to enter hibernate, low to exit). This mode should be used when the application allows for the MRF24WB0M / MRF24WG0M module to be off for extended periods of time.</td>
</tr>
</tbody>
</table>

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).

Module

Wi-Fi API

Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Save Public Members</td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td>Power Save Internal Members</td>
<td>Functions and variables internal to the module</td>
</tr>
<tr>
<td>Functions</td>
<td>The following table lists functions in this documentation.</td>
</tr>
<tr>
<td>Types</td>
<td>The following table lists types in this documentation.</td>
</tr>
<tr>
<td>Variables</td>
<td>The following table lists variables in this documentation.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetAppPowerSaveMode</td>
<td>Returns state of power save mode</td>
</tr>
<tr>
<td>SetAppPowerSaveMode</td>
<td>Enable or disable power save mode</td>
</tr>
</tbody>
</table>

Types
### Name | Description
--- | ---
| tWFPsPwrMode | Enumeration of valid values for WFSetPowerSaveMode() |
| tWFPwrModeReq | Power Save Mode Request Structure |

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_AppPowerSaveModeEnabled</td>
<td>Enable or Disable Power Save Mode</td>
</tr>
<tr>
<td>g_powerSaveState</td>
<td>Power save states</td>
</tr>
<tr>
<td>g_psPollActive</td>
<td>Status of PS-Poll</td>
</tr>
<tr>
<td>g_sleepNeeded</td>
<td>TRUE if need to put device back into PS-Poll sleep mode. else FALSE</td>
</tr>
</tbody>
</table>

**Wi-Fi API > Wi-Fi Power Save**
Power Save Public Members

The following functions and variables are available to the stack application.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_GetPowerSaveState</td>
<td>Returns current power-save state.</td>
</tr>
<tr>
<td>WF_HibernateEnable</td>
<td>Puts the MRF24W into hibernate mode by setting HIBERNATE pin to HIGH.</td>
</tr>
<tr>
<td>WF_PsPollDisable</td>
<td>Disables PS-Poll mode.</td>
</tr>
<tr>
<td>WF_PsPollEnable</td>
<td>Enables PS Poll mode.</td>
</tr>
<tr>
<td>CheckHibernate</td>
<td>Enters or exits from MRF24W hibernate mode.</td>
</tr>
</tbody>
</table>

### Module

**Wi-Fi Power Save**

### Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFHibernate</td>
<td>This is record WFHibernate.</td>
</tr>
</tbody>
</table>

[Wi-Fi API] > [Wi-Fi Power Save] > Power Save Public Members
WF_GetPowerSaveState Function

C

```c
void WF_GetPowerSaveState(
    UINT8 * p_powerSaveState
);
```

**Description**

Returns the current MRF24W power save state.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_PS_HIBERNATE</td>
<td>MRF24W in hibernate state</td>
</tr>
<tr>
<td>WF_PS_PS_POLL_DTIM_ENABLED</td>
<td>MRF24W in PS-Poll mode with DTIM enabled</td>
</tr>
<tr>
<td>WF_PS_PS_POLL_DTIM_DISABLED</td>
<td>MRF24W in PS-Poll mode with DTIM disabled</td>
</tr>
<tr>
<td>WF_PS_POLL_OFF</td>
<td>MRF24W is not in any power-save state</td>
</tr>
</tbody>
</table>

**Preconditions**

MACInit must be called first.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_powerSaveState</td>
<td>Pointer to where power state is written</td>
</tr>
</tbody>
</table>

**Returns**

None.

**Remarks**

None.
WF_HibernateEnable Function

C

```c
void WF_HibernateEnable();
```

Description

Enables Hibernate mode on the MRF24W, which effectively turns off the device for maximum power savings. HIBERNATE pin on MRF24W is set to HIGH.

MRF24W state is not maintained when it transitions to hibernate mode. To remove the MRF24W from hibernate mode call WF_Init().

Preconditions

MACInit must be called first.

Returns

None.

Remarks

Note that because the MRF24W 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 MRF24W 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 MRF24W without needing a host reset.
WF_PsPollDisable Function

```c
void WF_PsPollDisable();
```

**Description**

Power Management Functions

Disables PS Poll mode. The MRF24W will stay active and not go sleep.

**Preconditions**

MACInit must be called first.

**Returns**

None.

**Remarks**

None.
WF_PsPollEnable Function

```c
void WF_PsPollEnable(
    BOOL rxDtim
);
```

Description

Enables PS Poll mode. PS-Poll (Power-Save Poll) is a mode allowing for longer battery life. The MRF24W 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 MRF24W to wake up each time the Host sends Tx data or a control message to the MRF24W. When the Host message transaction is complete the MRF24W 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.

Preconditions

MACInit must be called first.

Parameters
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rxDtim</td>
<td>TRUE listens at the DTIM interval and FALSE listens at the CASetListenInterval</td>
</tr>
</tbody>
</table>

**Returns**

None.

**Remarks**

None.
CheckHibernate Function

```c
static void CheckHibernate();
```

**Description**

Achieves maximum power savings. WF_USE_POWER_SAVE_FUNCTIONS must be enabled. In hibernate mode, it will turn off LDO of the MRF24W module, which is turning off the power completely. It has the same effect of resetting the module.

MRF24W state is not maintained when it transitions to hibernate mode. To remove the MRF24W from hibernate mode call WF_Init().

**Preconditions**

MACInit must be called first.

**Returns**

None.

**Remarks**

Note that because the MRF24W 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 MRF24W and TCP/IP stack.

Refer to WFEasyConfigProcess() for a working example of softAP using hibernate mode.

Future versions of the stack might have the ability to save stack context as well, ensuring a clean wake up for the MRF24W without needing a host reset.
WFHibernate Structure

```c
struct WFHibernate {
    UINT8 state;
    UINT8 wakeup_notice;
};
```

Description

This is record WFHibernate.
Power Save Internal Members

The following functions and variables are designated as internal to the module.

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendPowerModeMsg</td>
<td>Send power mode management message to the MRF24W.</td>
</tr>
<tr>
<td>SetPowerSaveState</td>
<td>Sets the desired power save state of the MRF24W.</td>
</tr>
</tbody>
</table>

Module

Wi-Fi Power Save

Wi-Fi API > Wi-Fi Power Save > Power Save Internal Members
SendPowerModeMsg Function

```c
static void SendPowerModeMsg(
    tWFPwrModeReg * p_powerMode
);
```

Description

LOCAL FUNCTION PROTOTYPES

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_powerMode</td>
<td>Pointer to tWFPwrModeReg structure to send to MRF24W.</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
SetPowerSaveState Function

```
C

void SetPowerSaveState(
    UINT8 powerSaveState
);
```

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>powerSaveState</td>
<td>Value of the power save state desired.</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.

Wi-Fi API > Wi-Fi Power Save > Power Save Internal Members > SetPowerSaveState Function
## Functions

### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GetAppPowerSaveMode</code></td>
<td>Returns state of power save mode</td>
</tr>
<tr>
<td><code>SetAppPowerSaveMode</code></td>
<td>Enable or disable power save mode</td>
</tr>
</tbody>
</table>

## Module

**Wi-Fi Power Save**

[Wi-Fi API](#) > [Wi-Fi Power Save](#) > [Functions](#)
GetAppPowerSaveMode Function

```c
BOOL GetAppPowerSaveMode();
```

**Description**

Returns state of global variable `g_AppPowerSaveModeEnabled`

**Returns**

None.

**Remarks**

None.
SetAppPowerSaveMode Function

C

void SetAppPowerSaveMode(
    BOOL state
);

Description

Set or reset global variable `g_AppPowerSaveModeEnabled` to enable or disable power save mode

Returns

None.

Remarks

None.

Wi-Fi API > Wi-Fi Power Save > Functions > SetAppPowerSaveMode Function
### Module

**Wi-Fi Power Save**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tWFPsPwrMode</td>
<td>Enumeration of valid values for WFSetPowerSaveMode()</td>
</tr>
<tr>
<td>tWFPwrModeReg</td>
<td>Power Save Mode Request Structure</td>
</tr>
</tbody>
</table>

[Wi-Fi API] > [Wi-Fi Power Save] > [Types]
**tWFPsPwrMode Type**

C

```c
typedef enum tWFPsPwrMode tWFPsPwrMode;
```

**Description**

Enumeration of valid values for WFSetPowerSaveMode()
tWFPwrModeReq Type

typedef struct pwrModeRequestStruct tWFPwrModeReq;

Description

Power Save Mode Request Structure
## Variables

### Module

**Wi-Fi Power Save**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>g_AppPowerSaveModeEnabled</code></td>
<td>Enable or Disable Power Save Mode</td>
</tr>
<tr>
<td><code>g_powerSaveState</code></td>
<td>Power save states</td>
</tr>
<tr>
<td><code>g_psPollActive</code></td>
<td>Status of PS-Poll</td>
</tr>
<tr>
<td><code>g_sleepNeeded</code></td>
<td>TRUE if need to put device back into PS-Poll sleep mode. else FALSE</td>
</tr>
</tbody>
</table>

[Wi-Fi API] > [Wi-Fi Power Save] > Variables
g_AppPowerSaveModeEnabled Variable

C

BOOL g_AppPowerSaveModeEnabled = FALSE;

Description

Enable or Disable Power Save Mode
g_powerSaveState Variable

C

UINT8 g_powerSaveState = WF_PS_OFF;

Description

Power save states

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_PS_HIBERNATE</td>
<td>1</td>
</tr>
<tr>
<td>WF_PS_PS_POLL_DTIM_ENABLED</td>
<td>2</td>
</tr>
<tr>
<td>WF_PS_PS_POLL_DTIM_DISABLED</td>
<td>3</td>
</tr>
<tr>
<td>WF_PS_OFF</td>
<td>4</td>
</tr>
</tbody>
</table>

Wi-Fi API > Wi-Fi Power Save > Variables > g_powerSaveState Variable
g_psPollActive Variable

C

BOOL g_psPollActive = FALSE;

Description

Status of PS-Poll
g_sleepNeeded Variable

```c
BOOL g_sleepNeeded = FALSE;
```

Description

TRUE if need to put device back into PS-Poll sleep mode. else FALSE
## Wi-Fi Miscellaneous Module

[Wi-Fi API](#)

### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wi-Fi Miscellaneous Public Members</strong></td>
<td>Functions and variables accessible by the stack application</td>
</tr>
</tbody>
</table>

[Wi-Fi API](#) > Wi-Fi Miscellaneous

Microchip TCP/IP Stack 5.42.08 - June 15, 2013  
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
The following functions and variables are available to the stack application.

## Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_GetDeviceInfo</td>
<td>Retrieves WF device information (MRF24WB0M_DEVICE/MRF24WG0MDEVICE, romVersion and patchVersion).</td>
</tr>
<tr>
<td>WF_GetMacAddress</td>
<td>Retrieves the MRF24W MAC address</td>
</tr>
<tr>
<td>WF_GetMacStats</td>
<td>Gets MAC statistics.</td>
</tr>
<tr>
<td>WF_GetMultiCastFilter</td>
<td>Gets a multicast address filter from one of the two multicast filters.</td>
</tr>
<tr>
<td>WF_GetRegionalDomain</td>
<td>Retrieves the MRF24W Regional domain.</td>
</tr>
<tr>
<td>WF_GetRtsThreshold</td>
<td>Gets the RTS Threshold</td>
</tr>
<tr>
<td>WF_SetMacAddress</td>
<td>Uses a different MAC address for the MRF24W</td>
</tr>
<tr>
<td>WF_SetMultiCastFilter</td>
<td>Sets a multicast address filter using one of the two multicast filters.</td>
</tr>
<tr>
<td>WF_SetRegionalDomain</td>
<td>Enables or disables the MRF24W Regional Domain. This function is NOT supported due to FCC requirements, which does not allow programming of the regional domain.</td>
</tr>
<tr>
<td>WF_SetRtsThreshold</td>
<td>Sets the RTS Threshold</td>
</tr>
<tr>
<td>WF_EnableSWMultiCastFilter</td>
<td>Forces the module FW to use software filter instead of hardware filter</td>
</tr>
<tr>
<td>WF_MulticastSetConfig</td>
<td>Sets a multicast address filter using one of the two multicast filters.</td>
</tr>
<tr>
<td>WF_SetLinkDownThreshold</td>
<td>Can be called to set link down threshold in softAP network type.</td>
</tr>
</tbody>
</table>
Module

**Wi-Fi Miscellaneous**

**Structures**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFMacStatsStruct</td>
<td>Used in <strong>WF_GetMacStats</strong>.</td>
</tr>
<tr>
<td>WFMulticastConfigStruct</td>
<td>Used in <strong>WF_MulticastSetConfig</strong>, <strong>WF_MulticastGetConfig</strong>.</td>
</tr>
<tr>
<td>tWFDeviceInfoStruct</td>
<td>used in <strong>WF_GetDeviceInfo</strong>.</td>
</tr>
</tbody>
</table>

**Wi-Fi API** > **Wi-Fi Miscellaneous** > **Wi-Fi Miscellaneous Public Members**
WF_GetDeviceInfo Function

```c
void WF_GetDeviceInfo(tWFDeviceInfo * p_deviceInfo);
```

Description

Version functions

Retrieves RF module information.

- MRF24WB will have romVersion = 0x12.
- MRF24WG will have romVersion = 0x30 or 0x31.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_deviceInfo</td>
<td>Pointer where device info will be written</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_GetMacAddress Function

C

```c
void WF_GetMacAddress(
    UINT8 * p_mac
);
```

Description

Retrieves the MRF24W MAC address

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_mac</td>
<td>Pointer where mac will be written (must point to a 6-byte buffer)</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.

Wi-Fi API > Wi-Fi Miscellaneous > Wi-Fi Miscellaneous Public Members > WF_GetMacAddress Function
WF_GetMacStats Function

C

```c
void WF_GetMacStats(
    tWFMacStats * p_macStats
);
```

Description

MAC Stats

Returns MAC statistics on number of frames received or transmitted for defined situations such as number of frames transmitted with multicast bit set in destination MAC address. Refer to WFApi.h for data struct `WFMacStatsStruct` / tWFMacStats.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_macStats</td>
<td>Pointer to where MAC statistics are written</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_GetMultiCastFilter Function

C

```c
void WF_GetMultiCastFilter(
    UINT8 multicastFilterId,
    UINT8 multicastAddress[6]
);
```

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 0 address
[12] -- Compare Address 0 group
[13] -- Compare Address 0 type
[14-19] -- Compare Address 1 address
[20] -- Compare Address 1 group
[21] -- Compare Address 1 type
[22-27] -- Compare Address 2 address
[28] -- Compare Address 2 group
```
Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>multicastFilterId</td>
<td>WF_MULTICAST_FILTER_1 or WF_MULTICAST_FILTER_2</td>
</tr>
<tr>
<td>multicastAddress</td>
<td>6-byte address</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_GetRegionalDomain Function

C

```c
void WF_GetRegionalDomain(
    UINT8 * p_regionalDomain
);
```

Description

see tWFRegDomain enumerated types

Gets the regional domain on the MRF24W. MRF24W is programmed with FCC regional domain as default.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_regionalDomain</td>
<td>Pointer where the regional domain value will be written</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.

Wi-Fi API > Wi-Fi Miscellaneous > Wi-Fi Miscellaneous Public Members > WF_GetRegionalDomain Function
WF_GetRtsThreshold Function

C

```
void WF_GetRtsThreshold(
    UINT16 * p_rtsThreshold
);
```

Description

Gets the RTS/CTS packet size threshold.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_rtsThreshold</td>
<td>Pointer to where RTS threshold is written</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_SetMacAddress Function

C

```c
void WF_SetMacAddress(
    UINT8 * p_mac
);
```

Description

MAC Address Functions

Directs the MRF24W 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 MRF24W to use a different MAC.

Preconditions

MACInit must be called first. Cannot be called when the MRF24W is in a connected state.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_mac</td>
<td>Pointer to 6-byte MAC that will be sent to MRF24W</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_SetMultiCastFilter Function

C

```c
void WF_SetMultiCastFilter(
    UINT8 multicastFilterId,
    UINT8 multicastAddress[6]
);
```

Description

Multicast Functions

This function allows the application to configure up to two Multicast Address 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.

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>multicastFilterId</td>
<td>WF_MULTICAST_FILTER_1 or WF_MULTICAST_FILTER_2</td>
</tr>
<tr>
<td>multicastAddress</td>
<td>6-byte address (all 0xFF will inactivate the filter)</td>
</tr>
</tbody>
</table>

Returns
Remarks

Definition WF_USE_MULTICAST_FUNCTIONS needs to be enabled.
WF_SetRegionalDomain Function

```c
void WF_SetRegionalDomain(
    UINT8 regionalDomain
);
```

**Description**

see tWFRegDomain enumerated types

MRF24W is programmed with FCC regional domain as default. To cater for other regional domains, use `WF_CASetChannelList()` to set up specific channels.

**Preconditions**

MACInit must be called first. This function must not be called while in a connected state.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regionalDomain</td>
<td>Value to set the regional domain to</td>
</tr>
</tbody>
</table>

**Returns**

None.

**Remarks**

None.
WF_SetRtsThreshold Function

```c
void WF_SetRtsThreshold(
    UINT16 rtsThreshold
);
```

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).

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtsThreshold</td>
<td>Value of the packet size threshold</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WF_EnableSWMultiCastFilter Function

```c
void WF_EnableSWMultiCastFilter();
```

**Description**

This function allows the application to configure up to max 16 software-based Multicast Address Filters on the MRF24WG0MA/B.

**Preconditions**

MACInit must be called first.

**Returns**

None.

**Remarks**

Definition ENABLE_SOFTWARE_MULTICAST_FILTER needs to be enabled.
WF_MulticastSetConfig Function

C

```c
void WF_MulticastSetConfig(
    tWFMultiCastConfig * p_config
);
```

Description

This function allows the application to configure up to two Multicast Address Filters on the MRF24W. If two active multicast filters are set up they are OR’d together – the MRF24W will receive and pass to the Host CPU received packets from either multicast address. The allowable values in p_config (tWFMultiCastConfig / WFMulticastConfigStruct) are:

- **filterId** -- WF_MULTICAST_FILTER_1 or WF_MULTICAST_FILTER_2
- **action** -- WF_MULTICAST_DISABLE_ALL (default) The Multicast Filter discards all received multicast messages – they will not be forwarded to the Host PIC. The remaining fields in this structure are ignored.

WF_MULTICAST_ENABLE_ALL The Multicast Filter forwards all received multicast messages to the Host PIC. The remaining fields in this structure are ignored.

WF_MULTICAST_USE_FILTERS The MAC filter will be used and the remaining fields in this structure configure which Multicast messages are forwarded to the Host PIC.
• macBytes -- Array containing the MAC address to filter on (using the destination address of each incoming 802.11 frame). Specific bytes with the MAC address can be designated as ‘don't care' bytes. See macBitMask. This field in only used if action = WF_MULTICAST_USE_FILTERS.

• macBitMask -- A byte where bits 5:0 correspond to macBytes[5:0]. If the bit is zero then the corresponding MAC byte must be an exact match for the frame to be forwarded to the Host PIC. If the bit is one then the corresponding MAC byte is a ‘don't care’ and not used in the Multicast filtering process. This field in only used if action = WF_MULTICAST_USE_FILTERS.

By default, both Multicast Filters are inactive.

Preconditions

MACInit must be called first.

Returns

None.

Remarks

Definition WF_USE_MULTICAST_FUNCTIONS needs to be enabled.

Example

- Filter on Multicast Address of 01:00:5e:xx:xx:xx where xx are don't care bytes.

    p_config->filterId = WF_MULTICAST_FILTER_1

    [0] [1] [2] [3] [4] [5]
p_config->macBytes[] = 01, 00, 5e, ff, ff, ff (0xff are the don't care bytes)

p_config->macBitMask = 0x38 --> bits 5:3 = 1 (don't care on bytes 3,4,5)

--> bits 2:0 = 0 (exact match required on bytes 0,1,2)
WF_SetLinkDownThreshold Function

C

```c
void WF_SetLinkDownThreshold(
    UINT8 threshold
);
```

**Description**

When MRF24W is configured as a softAP, it will ping the devices to determine whether devices are alive or dead by transmitting consecutive NULL DATA packets. If device is alive, it will transmit ACK back to softAP. If device is dead, softAP will not receive any packets from device. After PARAM_LINK_DOWN_THRESHOLD is reached, softAP considers the device to be dead. To enable this feature, enable SOFTAP_CHECK_LINK_STATUS. This function is only valid with MRF24WG RF module FW version 0x3107 or the later.

**Preconditions**

MACInit must be called first. SOFTAP_CHECK_LINK_STATUS must be enabled.

**Returns**

None.

**Remarks**

None.
WF_GetTxMode Function

C

```c
void WF_GetTxMode(
    UINT8 * mode
);
```

Description

Retrieves transmission (tx) mode that indicates transmission rates (802.11b or 802.11g or 802.11 legacy rates)

Preconditions

MACInit must be called first.

Returns

None.

Remarks

None.
WF_SetTxMode Function

C

```c
void WF_SetTxMode(
    UINT8 mode
);
```

Description

Configures 802.11 transmission (Tx) rates for 802.11b or 802.11g or 802.11 legacy rates (1-2Mbps)

Preconditions

MACInit must be called first.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>Tx rate</td>
</tr>
<tr>
<td></td>
<td>- WF_TXMODE_G_RATES (default) -- will use all 802.11g rates</td>
</tr>
<tr>
<td></td>
<td>- WF_TXMODE_B_RATES -- will only use 802.11b rates</td>
</tr>
<tr>
<td></td>
<td>- WF_TXMODE_LEGACY_RATES -- will only use 1 and 2 mbps rates</td>
</tr>
</tbody>
</table>

Returns

None.

Remarks

None.
WFMacStatsStruct Structure

```
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;
};
```

Description

Used in **WF_GetMacStats**.

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT32 MibWEPExcludeCtr;</td>
<td>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.</td>
</tr>
<tr>
<td>UINT32 MibTxBytesCtr;</td>
<td>Total number of Tx bytes that have been transmitted</td>
</tr>
<tr>
<td>UINT32 MibTxMulticastCtr;</td>
<td>Number of frames successfully transmitted that had the multicast bit set in the destination MAC address.</td>
</tr>
<tr>
<td>UINT32 MibTxFailedCtr;</td>
<td>Number of Tx frames that failed due to the number of transmits exceeding the retry count.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UINT32 MibTxRtryCtr;</td>
<td>Number of times a transmitted frame needed to be retried</td>
</tr>
<tr>
<td>UINT32 MibTxMultRtryCtr;</td>
<td>Number of times a frame was successfully transmitted after more than one retransmission.</td>
</tr>
<tr>
<td>UINT32 MibTxSuccessCtr;</td>
<td>Number of Tx frames successfully transmitted.</td>
</tr>
<tr>
<td>UINT32 MibRxDupCtr;</td>
<td>Number of frames received where the Sequence Control field indicates a duplicate.</td>
</tr>
<tr>
<td>UINT32 MibRxCtsSuccCtr;</td>
<td>Number of CTS frames received in response to an RTS frame.</td>
</tr>
<tr>
<td>UINT32 MibRxCtsFailCtr;</td>
<td>Number of times an RTS frame was not received in response to a CTS frame.</td>
</tr>
<tr>
<td>UINT32 MibRxAckFailCtr;</td>
<td>Number of times an Ack was not received in response to a Tx frame.</td>
</tr>
<tr>
<td>UINT32 MibRxBytesCtr;</td>
<td>Total number of Rx bytes received.</td>
</tr>
<tr>
<td>UINT32 MibRxFragCtr;</td>
<td>Number of successful received frames (management or data)</td>
</tr>
<tr>
<td>UINT32 MibRxMultCtr;</td>
<td>Number of frames received with the multicast bit set in the destination MAC address.</td>
</tr>
<tr>
<td>UINT32 MibRxFCSErrCtr;</td>
<td>Number of frames received with an invalid Frame Checksum (FCS).</td>
</tr>
<tr>
<td>UINT32 MibRxWEPUndecryptCtr;</td>
<td>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.</td>
</tr>
<tr>
<td>UINT32 MibRxFragAgedCtr;</td>
<td>Number of times that fragments ‘aged out’, or were not received in the allowable time.</td>
</tr>
<tr>
<td>UINT32 MibRxMICFailureCtr;</td>
<td>Number of MIC failures that have occurred.</td>
</tr>
</tbody>
</table>
WFMulticastConfigStruct Structure

C

```c
struct WFMulticastConfigStruct {
    UINT8 filterId;
    UINT8 action;
    UINT8 macBytes[6];
    UINT8 macBitMask;
};
```

Description

Used in WF_MulticastSetConfig, WF_MulticastGetConfig.
tWFDeviceInfoStruct Structure

C

```c
struct tWFDeviceInfoStruct {
    UINT8 deviceType;
    UINT8 romVersion;
    UINT8 patchVersion;
};
```

Description

used in `WF_GetDeviceInfo`

Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8 deviceType;</td>
<td>MRF24WB0M_DEVICE or MRF24WG0M_DEVICE</td>
</tr>
<tr>
<td>UINT8 romVersion;</td>
<td>ROM version number</td>
</tr>
<tr>
<td>UINT8 patchVersion;</td>
<td>Patch version number</td>
</tr>
</tbody>
</table>

Wi-Fi API > Wi-Fi Miscellaneous > Wi-Fi Miscellaneous Public Members > tWFDeviceInfoStruct Structure
WF_ProcessEvent

There are several events that can occur on the MRF24WB0M / MRF24WG0M that the host CPU may want to know about. All MRF24WB0M / MRF24WG0M 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_Config.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 WF_ProcessEvent(). 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:

```c
void WF_ProcessEvent(UINT8 event, UINT16 eventInfo, UINT8 *extraInfo)
```

There are 3 inputs to the function:

<table>
<thead>
<tr>
<th>event</th>
<th>The event that occurred.</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventInfo</td>
<td>Additional information about the event. Not all events have associated info, in which case this value will be set to WF_NO_ADDITIONAL_INFO (0xff)</td>
</tr>
<tr>
<td>*extraInfo</td>
<td>Additional information about the event. When DERIVE_KEY_FROM_PASSPHRASE_IN_HOST is enabled, where host will compute the key from the passphrase, this field contains the WPA Passphrase that will be sent to the host for the computation.</td>
</tr>
</tbody>
</table>

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), or
2. Function **WF_CASetEventNotificationAction()**.

<table>
<thead>
<tr>
<th>event</th>
<th>eventInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WF_EVENT_CONNECTION_SUCCESSFUL</strong></td>
<td>The connection attempt was successful.</td>
</tr>
<tr>
<td></td>
<td>eventInfo:</td>
</tr>
<tr>
<td></td>
<td>• Always <strong>WF_NO_ADDITIONAL_INFO</strong> (Optional event)</td>
</tr>
<tr>
<td><strong>WF_EVENT_CONNECTION_FAILED</strong></td>
<td>The connection attempt failed</td>
</tr>
<tr>
<td></td>
<td>eventInfo:</td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_JOIN_FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_AUTHENTICATION_FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_ASSOCIATION_FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_WEP_HANDSHAKE_FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_PSK_CALCULATION_FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_PSK_HANDSHAKE_FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_ADHOC_JOIN_FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_SECURITY_MISMATCH_FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_NO_SUITABLE_AP_FOUND_FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_RETRY_FOREVER_NOT_SUPPORTED_FAILURE</strong></td>
</tr>
<tr>
<td></td>
<td>(Optional event)</td>
</tr>
<tr>
<td><strong>WF_EVENT_CONNECTION_TEMPORARILY_LOST</strong></td>
<td>An established connection was temporarily lost – the connection</td>
</tr>
<tr>
<td></td>
<td>algorithm is attempting to reconnect. The eventInfo field indicates why</td>
</tr>
<tr>
<td></td>
<td>the connection was lost.</td>
</tr>
<tr>
<td></td>
<td>eventInfo:</td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_BEACON_TIMEOUT</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_DEAUTH_RECEIVED</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_DISASSOCIATE_RECEIVED</strong></td>
</tr>
<tr>
<td></td>
<td>(Optional event)</td>
</tr>
<tr>
<td><strong>WF_EVENT_CONNECTION_PERMANENTLY_LOST</strong></td>
<td>An established connection was permanently lost – the connection</td>
</tr>
<tr>
<td></td>
<td>algorithm is attempting to reconnect. The eventInfo field indicates why</td>
</tr>
<tr>
<td></td>
<td>the connection was lost.</td>
</tr>
<tr>
<td></td>
<td>eventInfo:</td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_BEACON_TIMEOUT</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_DEAUTH_RECEIVED</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>WF_DISASSOCIATE_RECEIVED</strong></td>
</tr>
<tr>
<td></td>
<td>(Optional event)</td>
</tr>
<tr>
<td>Event ID</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| WF_EVENT_CONNECTION_PERMANENTLY_LOST | connection algorithm either ran out of retries or was configured not to retry. The eventInfo field indicates why the connection was lost. EventInfo:  
- WF_BEACON_TIMEOUT  
- WF_DEAUTH_RECEIVED  
- WF_DISASSOCIATE_RECEIVED  

This event can also be generated when `WF_CMDDisconnect()` 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()` completed and results can be read from MRF24WB0M / MRF24WG0M. EventInfo: Number of scan results |
| WF_EVENT_SOFT_AP_EVENT | Available only for MRF24WG0M (i) FW version 0x3108 and later and (ii) MLA v5.42.06 release or later. Indication of client's connection status, when a client has connected or disconnected or not powered on/active or received deauthentication. |
| WF_EVENT_KEY_CALCULATION_REQUEST | This event is generated when `DERIVE_KEY_FROM_PASSPHRASE_IN_HOST` is enabled, MRF24WG0M will transmit the passphrase to the host via the field *extraInfo. Where the host will then compute the passphrase from the key. Refer to function prototype `WF_ConvPassphrase2Key()` for more information. |

### 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.

```c
void WF_ProcessEvent(UINT8 event, UINT16 eventInfo)
```
```c
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_CONNECTION_REESTABLISHED:
        /* Application code here */
        break;

    default:
        WF_ASSERT(FALSE);
        break;
}
```
Access Point Compatibility

Introduction
The MRF24WB0M / MRF24WG0M 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 section 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
- Apple Airport Express
- Apple Airport Extreme
- Apple Time Capsule
- Asus RT-N16
- Asus WL530g
- AirLink AR690W
- Belkin N1
- Belkin F5D7231-4
- Belkin F5D8231-4
- Belkin F7D1301 v1
- Belkin F7D3302 v1
- Belkin F7D5301 v1
- Belkin Surf N300
- Buffalo WHR-G125
- Buffalo WHR-HP-G54
- Buffalo WHR-HP-GN
- Cisco E1000
- Cisco E3000
- Cisco E4200
- Cisco M20
- Cisco Vallet M10
- Corega CG-WLAPGMN
- Corega CG-WLBARGO
- D-Link DI-524
- D-Link DIR-615
- D-Link DIR-655
- D-Link DIR-665
- D-Link DIR-825
- D-Link DIR-855
- D-Link WBR-1310
- D-Link WBR-2310
- Dynex DX-WGRTR G
- Dynex DX-WGRTR v1000
- Level1 WBR-3408
- Linksys WRT150N v1.1
- Linksys WRT310N
- Linksys WRT54G2
- Microsoft MN-700
- Netgear WG103
- Netgear WGR614v9
- Netgear WGT624v2
- Netgear WN2000RPT
- Netgear WN802T v2
- Netgear WNDR3300
- Netgear WNDR3700
- Netgear WNR1000 v2
- Netgear WNR1000 v3
- Netgear WNR200 v3
- Netgear WPN824v2
- Netgear WNR854T
- PCI MZK-W04NU
- Proxim AP-700
- SMC Networks SMCWBR14S-N4
- SMC Networks SMCWBR14T-G
- TP-Link TL-WR340G
- TP-Link TL-WR541G
- TP-Link TL-WR740N
- TP-Link TL-WR741ND
- TP-Link TL-WR841ND
- TP-Link TL-WR941N/D
- Westell B90-327W15-06
- ZyXel P-330W
- ZyXel X550N

*Note Tests Performed:
- Basic association with the AP (no security)
- Association with WEP security
- Association with WPA/WPA2-PSK security
- Ping test validation.

**Module**

**Wi-Fi API**

Wi-Fi API > Access Point Compatibility

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
Tips for Setting up Routers for 802.11b/g Use

The purpose of this section is to describe the settings for the most typical AP configurable parameters to enable compatibility with the Microchip MRF24WB0M / MRF24WG0M 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 / MRF24WG0M, 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.

**Module**

**Wi-Fi API**

**Wi-Fi API** > 802.11 AP/Router Configuration Settings

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
WiFi Troubleshooting Tips

The following clarifications are to be noted for use of the MRF24W with Microchip TCPIP Stack versions unless otherwise noted.

For topics not addressed in this documentation, search Microchip Technical Support Knowledgebase.

## Module

### Wi-Fi API

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null String ESSID</td>
<td>Describes how to handle null string ESSID</td>
</tr>
<tr>
<td>Read back RF module Firmware version</td>
<td>Describes how to read back RF module firmware version</td>
</tr>
<tr>
<td>RF Module Firmware Update</td>
<td>Describes how to update MRF24WB0M/MRF24WG0M RF module firmware</td>
</tr>
<tr>
<td>Wi-Fi Protected Setup (WPS) Issues</td>
<td>Describes MRF24WG0M WPS Issues</td>
</tr>
<tr>
<td>Network Switch or Change</td>
<td>Describes how to perform a network switch or change</td>
</tr>
<tr>
<td>Hibernate Mode</td>
<td>Describes the MRF24W RF module Hibernate Mode software flow</td>
</tr>
<tr>
<td>Management Scan Message Conflict</td>
<td>Describes how to handle management scan message conflict</td>
</tr>
<tr>
<td>Handling of maximum length SSID</td>
<td>Describes how to handle maximum length SSID</td>
</tr>
<tr>
<td>Multicast Filters : Hardware vs Software</td>
<td>Describes considerations for hardware and software Multicast filters.</td>
</tr>
<tr>
<td>MRF24WB0M assert failures when using &lt;iwconfig scan&gt; command</td>
<td>In WiFi console demo, using MRF24WB0M and certain sequences (wrong pass-phrase), &lt;iwconfig scan&gt; command can produce assert failures.</td>
</tr>
<tr>
<td>Topic</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MRF24WB0M advertised supported rates of 1, 2, 5.5 and 11 Mbps</td>
<td>Describes why MRF24WB0M shows advertised supported rates of 1, 2, 5.5 and 11 Mbps</td>
</tr>
<tr>
<td>MRF24WB0M Compatibility with AP/Routers</td>
<td>Describes MRF24WB0M compatibility with AP/Routers</td>
</tr>
<tr>
<td>Encounter issues after upgrading MRF24WB0M RF module Firmware version 0x1207</td>
<td>Describes new &quot;gRFModuleVer1209orLater&quot; APIs</td>
</tr>
<tr>
<td>How to fix MRF24WB0M / MRF24WG0M transmission rates</td>
<td>Describes how to fix the transmission rates of MRF24W</td>
</tr>
<tr>
<td>How to determine new IP address assigned</td>
<td>Describes how to determine new IP address being assigned to MRF24W by a DHCP server</td>
</tr>
<tr>
<td>How to increase TCP throughput</td>
<td>Describes the various methods to increase TCP throughput</td>
</tr>
<tr>
<td>Missing DHCP Client Name</td>
<td>Describes the software fixes to display DHCP client name</td>
</tr>
<tr>
<td>Error Scenario And Possible Causes</td>
<td>List of error messages and the possible causes</td>
</tr>
</tbody>
</table>

Wi-Fi API > WiFi Troubleshooting Tips
Null String ESSID

It is possible to call `WF_CMConnect(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` to a wrong AP or not `connect` at all. The only valid values that can be used for connection profile references are 1 and 2 (assuming that the `WF_CPCreate(&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(cpld)`, ensure that only a valid profile number previously returned from `WF_CPCreate(&cpld)` is used.
Read back RF module Firmware version

RF module FW version

There are 2 methods available.

1. Run WiFi Console standalone CLI command

   Type in command getwfver.

   The following will be displayed; MRF24W firmware version and Host Driver version.

2. Invoking function prototype `WF_GetDeviceInfo(tWFDeviceInfo *p_deviceInfo)`

   As part of initialization, WF_Init() will call `WF_GetDeviceInfo(tWFDeviceInfo *p_deviceInfo)`.

3 parameters will be returned

   • Type (1 for MRF24WB0M_DEVICE and 2 for MRF24WG0MDEVICE)
   • Rom Version (0x12 for MRF24WB0M and 0x30 / 0x31 for MRF24WG0M)
   • Patch version

MRF24WG0M RF module FW version

From RF module FW version 0x3107 onwards, FW release will follow this roll-out order

Even Numbered (eg 0x3108, 0x310a, etc)

   • Multi-DHCP
   • WPA-EAP
• No Wi-Fi Direct

• All other features, including SoftAP supporting max 4 clients

Odd Numbered (eg 3109, 0x310b)

• No multi-DHCP

• No WPA-EAP

• Wi-Fi Direct

• All other features, including SoftAP supporting only 1 client
RF Module Firmware Update

Flash Update Project

This is applicable for both MRF24WB0M and MRF24WG0M.

Go to website

ftp://mrfupdates@ftp.microchip.com

where username is mrfupdates and password is mchp1234.

For MRF24WB0M, the flash update project is
MRF24WB_Exp16FlashUpdater-120c-Rev1-windows-installer.exe

For MRF24WG0M, the flash update project is
MRF24WG_FlashUpdater-3107-Rev1-windows-installer.exe

Over-The-Air (OTA) MRF24WG0M RF Module Firmware Update

This is applicable only for MRF24WG0M.

To use OTA RF module firmware update, MRF24WG0M needs at least
RF module FW version 0x3107 and MLA v5.42.04 Oct 2012 release or
later.

This method is located in \tcpip\wifi console project.

In TCPIP MRF24W.h, enable
STACK_USE_AUTOUPDATE_TCPCLIENT.

In AutoUpdate_TCPClient_24G.c, configuration is defined in this file.
static BYTE ServerName[] = "www.microchip.com";
static BYTE PatchName[]="/mrfupdates/A2Patch_3107.bin";

//Username is mrfupdates , password is mchp1234
static BYTE Key_authorization[]="bXJmdXBkYXRlczptY2hwMTIzNA==";

static WORD ServerPort = 80; // Defines the port to be accessed for this application

Update PatchName[] with the required MRF24W firmware version file name. PatchName is case-sensitive.

In WF_Config.h, configure Wi-Fi parameters according to selected AP/router.

• Define infrastructure network type
#define MY_DEFAULT_NETWORK_TYPE
CFG_WF_INFRASTRUCTURE

• Define SSID and Wi-Fi security mode as that used by the AP/router.
#define MY_DEFAULT_WIFI_SECURITY_MODE
#define MY_DEFAULT_SSID_NAME

• For faster connection, you may define the channel AP/router is in.

e.g. #define MY_DEFAULT_CHANNEL_LIST {3}
Wi-Fi Protected Setup (WPS) Issues

Wi-Fi Protected Setup (WPS) security connection mode is only supported by MLA v5.42 July 2012 releases or later.

- Why is MRF24WG0M reporting error message "Event: Connection Failed : WF_RECV_DISASSOC : WF_UNSPECIFIED" when trying to connect to certain AP/routers in WPS-PBC security connection modes?

When running WiFi TCP/IP Demo, the output display shows

*** WiFi TCP/IP Demo ***

Start WiFi Connect

Domain: FCC
MAC: 00 1E C0 08 F1 04
SSID: (none)
Network Type: Infrastructure (using WPS Push Button)
Scan Type: Active Scan
Channel List: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
Retry Count: Retry Forever
Beacon Timeout: 40
Security: WPS push button method
Power Save: Disabled

New IP Address: 169.254.1.1
Event: Connection Failed : WF_RECV_DISASSOC : WF_UNSPECIFIED

A wireless capture revealed that some AP/routers is transmitting disassociation frames instead of deauthentication frames during WPS process. This behavior is not according to 802.11 specifications and thereby causing WPS to dysfunction. RF module FW (from 0x3108 onwards) was modified to handle this erroneous case.

- Why is the AP/router, in WPS mode, taking a longer time to connect, as compared to when AP/router is in WPA/WPA2 security mode?

Refer to Wi-Fi Protected Setup Specification Version 1.0h standard.

WPS protocol is built on top of the WPA/WPA2 EAPOL 4-way handshaking process and there are additional process for WPS. Time allowed for entire WPS protocol is 2 min.

The specifications listed the below parameters

1. Retransmission timeout = 5 sec
2. Individual message processing timeout = 15 sec
3. Overall timeout for entire protocol to complete = 2 min

- What is needed to see the WPS frame exchanges in a wireless sniffer capture?

Either click on <Protocol> field and scroll down until this starts with the alphabet "E..." or in the <Filter: > field, type in "eapol".
What is special about the PIN selected for WPS-PIN? Can this PIN be randomly selected by the user?

The 8 digit PIN is NOT randomly generated. The last digit is the checksum of first 7 digits of the PIN. If this checksum is wrong, MRF24WG0M module will reject this PIN code.

I just bought a brand new AP and have problems using WPS to connect to MRF24WG0M.

It has come to our attention that there are some AP/routers that do not work out of the box using WPS feature. These AP/routers may still require some minimum set up, such as setting up security to WPA2-Personal, etc. Please refer to their instruction manuals for more details in WPS setup.
Network Switch or Change

WiFi EZConfig is a good demo reference for handling network switch.

In WiFi EZConfig, the function prototype WFEasyConfigProcess() is called to execute a network change.

MRF24W is put into Hibernate mode, which implies a reset operation, whereby the LDO is turned off.

Old connection profile is deleted and new connection profile is created. New parameters, linked to selected network, are configured into new profile. Then exit out of Hibernate mode.

The recommend sequences are

1. Disconnect from current network by invoking WF_CMDisconnect()
2. Delete the profile by invoking WF_CPDelete()
3. Create new profile by invoking WF_CPCreate()
4. Set up parameters for this new profile such as setting SSID, Wi-Fi security, network type.
5. Enter Hibernate mode by setting WF_hibernate.state = WF_HB_ENTER_SLEEP and WF_hibernate.wakeup_notice = FALSE
6. Have a short time delay such as DelayMs(50)
7. Exit from Hibernate mode by setting WF_hibernate.wakeup_notice = TRUE
Hibernate Mode

Hibernate Mode

Hibernate mode is used during network switch or change.

Ensure definition WF_USE_POWER_SAVE_FUNCTIONS is enabled.

Within the main() loop, StackTask() will call MACProcess(). It will then invoke CheckHibernate(), which executes/handles hibernate mode based on WF_hibernate.state and WF_hibernate.wakeup_notice.

To enter into Hibernate mode, the following settings are required

- WF_hibernate.state = WF_HB_ENTER_SLEEP
- WF_hibernate.wakeup_notice = FALSE

To exit from Hibernate mode, the following settings are required

- WF_hibernate.wakeup_notice = TRUE
Management Scan Message Conflict

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, or scan command is sent.

Work around:

If you are controlling connect/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(MY_DEFAULT_LIST_RETRY_COUNT); should be 0

b. To disable De-authorization action

WF_CASetDeauthAction(WF_DO_NOT_ATTEMPT_TO_RECONNECT);

c. To disable De-authentication action

WF_CASetBeaconTimeoutAction(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()`

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 0.4 seconds 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 4 seconds unless the management timeout has been increased).

b. If you are only using the firmware retry and not doing ANY connection management (scan, `connect`, 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`” 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.
Handling of maximum length SSID

Maximum length SSID

An issue has been found with MLA v5.42.06 Feb 2013 and prior releases, when SSID of the selected AP is up to the maximum length (length of 32). If SSID of AP is 32 characters, then the type of security is always seen as OPEN. However, if SSID is less than 32 characters, then host-scan result returns the correct security type.

The root cause has been found to occur during the process of converting the SSID to ASCII string in order to print this out through console, whereby the last character was set to 0 to indicate end of string.

The workarounds needed are as follows.

1. WFEasyConfig.c

In WFRetrieveScanResult(), update the code as

```c
UINT16 WFRetrieveScanResult(UINT8 Idx, tWFScanResult *p_ScanResult)
{
    if (Idx >= SCANCXT.numScanResults)
        return WF_ERROR_INVALID_PARAM;
    WF_ScanGetResult(Idx, p_ScanResult);
    return WF_SUCCESS;
}
```

In WFDisplayScanMgr(), update the code as
void WFDisplayScanMgr()
{
    tWFScanResult bssDesc;
    char ssid[WF_MAX_SSID_LENGTH+1];
    char rssiChan[48];
    int i;

    ......................
    /* Display SSID */
    for(i=0;i<WF_MAX_SSID_LENGTH;i++) ssid[i] = bssDesc.ssid[i];
    ssid[WF_MAX_SSID_LENGTH] = 0;
    putsUART(ssid);
    putsUART("\n\n");
    ..........................
}

2. CustomHTTPApp.c
Modify HTTPPrint_name() as follows.
void HTTPPrint_name(void)
{
    if (bssDescIsValid)
    {
        if(strlen((const char*)bssDesc.ssid)<WF_MAX_SSID_LENGTH)
TCPPutString(sktHTTP, bssDesc.ssid);

else
{
    unsigned char buf_tmp[WF_MAX_SSID_LENGTH + 1];
    int i;
    for(i=0;i<WF_MAX_SSID_LENGTH;i++) buf_tmp[i] = bssDesc.ssid[i];
    buf_tmp[WF_MAX_SSID_LENGTH] = 0;
    TCPPutString(sktHTTP, buf_tmp);
}

else
{
    TCPPutROMString(sktHTTP, (ROM BYTE *)"0");
}

Wi-Fi API > WiFi Troubleshooting Tips > Handling of maximum length SSID

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
Multicast Filters

MRF24W has 2 hardware multicast filters.

If your design requires more than 2 multicast filters, there is an option to extend the multicast filters to a maximum of 16 software-based multicast filters.

To use this maximum of 16 multicast filters, the macro

#define ENABLE_SOFTWARE_MULTICAST_FILTER

needs to be enabled. The MRF24W FW will be notified to use software to run the filters instead of hardware. The downside of this software-based multicast filter option is the performance could possibly be degraded in the case scenario when there are so many multicast packets on the air.
MRF24WB0M assert failures when using <iwconfig scan> command

Host scan <iwconfig scan> command ASSERT failures

Applicable for MRF24WB0M only.

Using both MLA v2012-04-03 and v2013-02-15, if all connection parameters are correct, with only the pass-phrase set wrongly, it causes assert failure

WF ASSERTION at WFMgmtMsg.c Line Number = 248.

However, if the wrong ssid is used, <iwconfig scan> is OK.

- Case [1] wrong ssid only. < iwconfig scan > is OK
  > iwconfig ssid MCHP_test
  > iwpriv enc wpa-phrase
  > iwpriv phrase 12345434
  > iwconfig mode managed
  > Event: Connection Failed – eventInfo = 10, WF_NO_SUI TABLE_AP_FOUND_FAILURE
  > iwconfig scan <---- iwconfig scan is OK
  Scanning...
  Scan completed.
  > Event: Scan Results Ready, 21 results

- Case [2] wrong pass-phrase only. < iwconfig scan > causes assert failure
iwconfig ssid MCHP_test
iwpriv enc wpa-phrase
iwpriv phrase 12341234
iwconfig mode managed
Event: Connection Failed – eventInfo = 9, WF_SECURITY_MISMATCH_FAILURE
iwconfig scan
Scanning...
WF ASSERTION at WFMgmtMsg.c Line Number = 248

Recommended workarounds

1. WFScan.c
Replace WF_Scan() with the following

UINT16 WF_Scan(UINT8 Cpld)
{
UINT8 hdr[4];
ifndef MRF24WG
UINT8 connectionState;
UINT8 dummy;
endif

if (!WF_CMIsHostScanAllowed())
return WF_ERROR_OPERATION_CANCELLED;

#ifndef MRF24WG

WF_CMGetConnectionState(&connectionState, &dummy);
if (connectionState == WF_CSTATE_NOT_CONNECTED)

WF_CMConnect(0xff); /* MRF24WB 0x120c host scan bug workaround */
#endif

hdr[0] = WF_MGMT_REQUEST_TYPE;
hdr[1] = WF_SCAN_START_SUBTYPE;
hdr[2] = CplId; /* Connection Profile ID */
hdr[3] = 0; /* not used */

SendMgmtMsg(hdr, /* header */ sizeof(hdr), /* size of header */ NULL, /* no data */ 0); /* no data */

/* wait for mgmt response, free it after it comes in (no data needed) */
WaitForMgmtResponse(WF_SCAN_START_SUBTYPE, FREE_MGMT_BUFFER);
return WF_SUCCESS;

2. WFMgmtMsg.c
In WaitForMgmtResponse()
replace
} else {
} else {
WF_ASSERT(hdr.result == WF_SUCCESS);
}

} else {
if (!(hdr.result == WF_ERROR_CP_INVALID_PROFILE_ID
&& hdr.subtype == WF_CM_CONNECT_SUBYTYPE)) {
WF_ASSERT(hdr.result == WF_SUCCESS);

Wi-Fi API > WiFi Troubleshooting Tips > MRF24WB0M assert failures whe using <iwconfig scan>
command
**MRF24WB0M advertised supported rates of 1, 2, 5.5 and 11 Mbps**

MRF24WB0M, on the product level, only supports legacy 802.11 data rates of 1Mbps and 2Mbps.

However when MRF24WB0M transmits an association request to an AP, MRF24WB0M will advertise in the supported rates information element (IE) rates of 1, 2, 5.5 and 11 Mbps.

This is not a bug and is intentional, leveraging from our past experiences.

If MRF24WB0M module just advertises 1,2 Mbps, most APs out in the field have a tendency to drop MRF24WB0M module and not permit connection to the MRF24WB0M module. Therefore for APs compatibility reasons, MRF24WB0M will advertise supported rates reflecting that of the APs. For example, the beacon from the AP will advertise supported rates of 1, 2, 5.5, 11, 18, 24, 36 and 54 Mbps. MRF24WB0M will duplicate the 802.11b rates in this information and advertise supported rates of 1, 2, 5.5 and 11 Mbps in the association request frame supported rates IE field.

However, the tradeoff is that APs, thinking that MRF24WB0M module supports 1,2,5.5, 11 Mbps, starts transmitting 11 Mbps and this will cause eventually wireless connectivity to fail as MRF24WB0M module will not be able to receive the AP’s transmissions. However, bear in mind, according to 802.11 specifications, broadcast frames are to be transmitted at base rates of 1 and 2 Mbps. At the same time, this trend may start changing as the push for higher rates persist.

Bear in mind that MRF24WB0M module is meant for a captive environment, whereby the 802.11 network is known.
MRF24WB0M advertised supported rates of 1, 2, 5.5 and 11 Mbps
MRF24WB0M Compatibility with AP/Routers

MRF24WB0M Compatibility With AP/Routers

MRF24WB0M supports only legacy 802.11b rates of 1Mbps and 2Mbps.

To ensure compatibility with MRF24WB0M,

- iOS and Android FW releases need to support 1 and 2 Mbps
- APs/ routers need to support 1 and 2 Mbps

Recommended configurations on APs/Routers settings

- Set AP/router basic rate to 1-2 Mbps
- Set AP/router channel settings to defined social channels such as channel 1, 6 or 11.

If iOS, Android FW releases and AP/routers do not support 1 and 2 Mbps, customers are encouraged to feedback to the manufacturers or developers.

MRF24WB0M is best suited for captive networks, whereby the surrounding wireless networks are under control by the users.

In applications where the surrounding wireless networks are unknown and constantly changing, the MRF24WG0M is highly recommended over the MRF24WB0M.
Encounter issues after upgrading MRF24WB0M RF module Firmware version 0x1207

Current MRF24WB0M RF module Firmware version is 0x120C.

Starting with MRF24WB0M RF module Firmware version 0x1209, a new compilation parameter "gRFModuleVer1209orLater" is introduced into MLA TCPIP source codes. Ensure these supporting source codes are incorporated.

"gRFModuleVer1209orLater" changes are

1. New APIs:
These new APIs are not backward compatible with MRF24WB0M RF module Firmware version 0x1207.

- `WF_CPSetWepKeyType()` // Shared or Open Key
- `WF_CMGetConnectContext()` // Retrieve AP channel & bssid
- `WFEnableBroadcastProbeResponse()` // Send Probe Response with broadcast address in destination address.
- `WFEnableAggressivePowerSave()` // Turn off RF quicker in PS mode.
- `WF_CPSetSsidType()` // Allows connection to hidden SSID in Adhoc network.
- `WFEnableDeferredPowerSave()` // To enable compatibility with 1207. Allows FW to wait till DHCP done before going to PS mode.
- `WF_FixTxRateWithMaxPower()` // Set TX rate at 1Mbps with Max power
How to fix MRF24WB0M / MRF24WG0M transmission rates

**MRF24W Transmission Rates**

The function prototype `WF_FixTxRateWithMaxPower()` can be used to fix the transmission rates, with maximum power, for the MRF24W.

Even though in the MLA SW this function is compiled only under MRF24WB configuration, this function is applicable also for MRF24WG0M.

For MRF24WB0M, only 1 or 2 Mbps transmission rates can be fixed.

For MRF24WG0M, this function can be used to fix the transmission rates beyond 1 or 2 Mbps. The input parameter to be used is

* #define kOneMbps (2)
* #define kTwoMbps (4)
* #define kFiveMbps (11)
* #define kElevenMbps (22)
* #define kSixMbps (12)
* #define kNineMbps (18)
* #define kTwelveMbps (24)
* #define kEighteenMbps (36)
* #define kTwentyFourMbps (48)
* #define kThirtySixMbps (72)
* #define kFortyEightMbps (96)
* #define kFiftyFourMbps (108)
Bear in mind, this fixed transmission rate only applies to data packets. All management packets still use 1 Mbps.
How to determine new IP address assigned

Determine New IP Address Assigned to MRF24W

Check the global variable g_DhcpSuccessful.

When g_DhcpSuccessful becomes TRUE, AppConfig.MyIPAddr will reflect the new IP address that is assigned to MRF24W.

Below are some suggestions to determine the new IP address that has been assigned.

a) Enable Zero Configuration / mDNS

This feature enables users not to have any knowledge of the IP Address. Instead what is needed is the URL address such as http://mchpboard.local.

After typing this URL address, the MRF24W web page could be displayed, showing the new IP address.

Example
b) AP DHCP Client Table

Refer to your AP’s documentation for the necessary steps required to obtain the new IP address. As an example, for certain APs, by selecting <Status> and then <Local Network>, an option <DHCP Client Table> is offered. This DHCP Client Table will list the IP address that is assigned to the MRF24W.
c) WiFi TCPIP Demo will display the new IP address through a designated console terminal.
Wi-Fi API > WiFi Troubleshooting Tips > How to determine new IP address assigned

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home
How to increase TCP throughput

Methods to increase TCP throughput

1. Enlarge generic TCP rx/tx buffer size. Refer to TCPIP MRF24W.h. In TCPIP MRF24W.h modify the buffer size accordingly.

For example,

{TCP_PURPOSE_GENERIC_TCP_CLIENT, TCP_ETH_RAM, 1024, 100},

{TCP_PURPOSE_GENERIC_TCP_SERVER, TCP_ETH_RAM, 20, 4096},

2. In tcp.c, increase max TCP segment size.

For example, change #define TCP_MAX_SEG_SIZE_RX from (536u) to (1460u).

3. Disable TCP rx checksum check. This is not necessary because packet integrity is guaranteed by the MAC layer CRC32 check.

BOOL TCPProcess(NODE_INFO* remote, IP_ADDR* localIP, WORD len) {

    TCP_HEADER TCPHeader;
    PSEUDO_HEADER pseudoHeader;
    WORD_VAL checksum1;
    WORD_VAL checksum2;
    BYTE optionsSize;
}
#if 0 // disable TCP RX checksum check

// Calculate IP pseudoheader checksum.
pseudoHeader.SourceAddress = remote->IPAddr;
pseudoHeader.DestAddress = *localIP;
pseudoHeader.Zero = 0x0;
pseudoHeader.Protocol = IP_PROT_TCP;
pseudoHeader.Length = len;
SwapPseudoHeader(pseudoHeader);
checksum1.Val = ~CalcIPChecksum((BYTE*)&pseudoHeader,
sizeof(pseudoHeader));

// Now calculate TCP packet checksum in NIC RAM - should match
// pesudo header checksum
checksum2.Val = CalcIPBufferChecksum(len);

// Compare checksums.
if(checksum1.Val != checksum2.Val)
{
    MACDiscardRx();
    return TRUE;
}
#endif

4. Use PIC ram as TCB buffer instead of MRF24W scratch memory.
In TCPIP MRF24W.h

for example

Change

\{TCP_PURPOSE_FTP_DATA, TCP_ETH_RAM, 0, 128\},

to

\{TCP_PURPOSE_FTP_DATA, TCP_PIC_RAM, 0, 4096\},
Missing DHCP Client Name

Display DHCP client name

When MRF24W is a client in the infrastructure network, MRF24W client name appears as empty field in the AP/router's DHCP client table. See figure below.

To have MRF24W display a DHCP client name, incorporate the following changes into DHCP.c

Add new code below.

#define DHCP_HOSTNAME_SIZE 18
char Dhcp_HostName[DHCP_HOSTNAME_SIZE + 1];
Function:
static void DHCPSetHostName(char *HostName)
Description:
Set the Host Name.
Precondition:
None.
Parameters:
HostName = Pointer to zero terminated host name
Returns:
None
***************************************************************************/
void DHCPSetHostName(char *HostName)
{
    strncpy(Dhcp_HostName, HostName, DHCP_HOSTNAME_SIZE);
    Dhcp_HostName[DHCP_HOSTNAME_SIZE] = 0;
}

Modify existing DHCPSend() as below.
static void _DHCPSend(BYTE messageType, BOOL bRenewing)
{
    ...................................................
    // Load our interested parameters
// This is hardcoded list. If any new parameters are desired,
// new lines must be added here.
UDPPut(DHCP_PARAM_REQUEST_LIST);
UDPPut(DHCP_PARAM_REQUEST_LIST_LEN - 1);
UDPPut(DHCP_SUBNET_MASK);
UDPPut(DHCP_ROUTER);
UDPPut(DHCP_DNS);

// Add requested IP address to DHCP Request Message
if( ((messageType == DHCP_REQUEST_MESSAGE) && !bRenewing) ||
    (messageType == DHCP_DISCOVER_MESSAGE) &&
    DHCPClient.tempIPAddress.Val))
{
    UDPPut(DHCP_PARAM_REQUEST_IP_ADDRESS);
    UDPPut(DHCP_PARAM_REQUEST_IP_ADDRESS_LEN);
    UDPPutArray((BYTE*)&DHCPClient.tempIPAddress,
                DHCP_PARAM_REQUEST_IP_ADDRESS_LEN);
}

// Add any new parameter request here.
UDPPut(DHCP_HOST_NAME);

 sprintf((char *)Dhcp_HostName,"%s_%02x%02x",
               MY_DEFAULT_HOST_NAME, AppConfig.MyMACAddr.v[4],
               AppConfig.MyMACAddr.v[5]);
```c
UDPPut(strlen(Dhcp_HostName));
UDPPutArray((BYTE*)Dhcp_HostName, strlen(Dhcp_HostName));
```

With these modifications, MRF24W has a valid client name now. See figure below.
Error Scenario And Possible Causes

- **During scanning, the web browser reported error message "Command failed, connection to development board was lost".**

  This is not a bug because when you click the "OK" button in the message box, it will continue to work correctly. Above timeout error message happens when you choose to scan ALL channels. If you elect to scan only 2-3 channels, the scan duration will be much faster and the web browser is unlikely to timeout.

  A possible solution is to increase the timeout setting in mchp.js

  Eg \EasyConfigWebPages\javascript\mchp.js

  var timeOutMS = 5000; //ms

- **Scan results seem to display 1 SSID less. For example, there are 10 scan results but only 9 scan results are displayed.**

  If the scan result yields 10 scan results, our MRF24W + PIC development board will send 10 scan results to the webpage. However, the webpage will display only 9 scan results and drop its own SSID (scan result). As an example, the development board and laptop are using an AP named "test", then the webpage will not display SSID "test" among the displayed scan results.

- **Why is my device not displaying MRF24WG0M softAP SSID in the scan results?**

  There are 2 possible reasons.

  (a) Hidden SSID

  (b) The device is only supporting active scan. MRF24WG0M RF module FW version 0x3107 only supports softAP with passive scan.
From MRF24WG0M RF module FW version 0x3108 and future releases, softAP supports both active and passive scan. Check that your MRF24WG0M RF module FW version is 0x3108 and later.

- **How can the received signal strength indicator (RSSI) be obtained from the MRF24W?**

  RSSI can only be obtained from the scan results p_scanResult->rssi. Refer to function prototype `WF_ScanGetResult()`. MRF24W checks out the signal strength from the preamble of the incoming packets. The higher the values, the stronger is the received signal strength.

  MRF24WB : RSSI_MAX (200) , RSSI_MIN (106).
  MRF24WG : RSSI_MAX (128) , RSSI_MIN (43).

  The RSSI value is not directly translated to dbm because this is not calibrated number. However, as a guideline, MAX(200) corresponds to 0 dbm, MIN (106) corresponds to -94 dbm.

- **Why is MRF24W failing after chip reset?**

  It is discovered that glitches on the MRF24W reset line, that is occurring after ChipReset(), could potentially cause MRF24W to fail.

  The root cause is traced to the macros `WF_SetCE_N` and `WF_SetRST_N`, where the pin is configured as output first and then the level is set. The correct sequences should be to set the level first and then configure the pin as output.

  ```
  #define WF_SetCE_N(level)
  
  /*_ set pin to desired level */
  
  WF_HIBERNATE_IO = level;
  
  /*_ configure I/O as ouput */
  
  WF_HIBERNATE_TRIS = 0
  ```
#define WF_SetRST_N(level)

/* set pin to desired level */
WF_RESET_IO = level;

/* configure the I/O as an output */
WF_RESET_TRIS = 0

The above changes apply to both MRF24WB0M and MRF24WG0M.

Refer to MLA v5.42.04 for the changes in the file WFDriverPrv_24G.h (MRF24WG0M)

Refer to MLA v5.42.08 for the changes in the file WFDriverPrv.h (MRF24WB0M)

- **Why is the software hanging in infinite loop within WaitForMgmtResponse() function?**

  For MLA v5.41 2012-02-15 and earlier versions, it is possible to be caught in an infinite loop. This could be caused by a race condition where a data message comes in before a mgmt response. Thus the workaround solution is to throw away a data message that comes in while waiting for a mgmt response.

  This workaround is implemented in MLA v5.41.02 2012-04-03 and future versions.

  The work-around is to change the while loop code to:

  while (gMgmtConfirmMsgReceived == FALSE) {
    WFProcess();
/* if received a data packet while waiting for mgmt packet */

if (g_HostRAWDataPacketReceived) {

    // throw away the data rx
    RawMountRxBuffer();
    DeallocateDataRxBuffer();
    g_HostRAWDataPacketReceived = FALSE;

    /* ensure interrupts enabled */
    WF_EintEnable(); }}
Wireless Packets Analysis

This section is applicable to any wireless 802.11 products and describes the client device in an infrastructure network type. The approach remains similar when analyzing other network types.

Hardware

Wireless sniffer hardware, such as AirPcap USB adaptor by Riverbed Technology, is needed to capture the wireless packets. Here it is assumed the user has knowledge of setting up the wireless sniffer hardware for packets capture.

Software

Software such as Wireshark is needed to analyse the wireless packets captured.

Even if you do not have the hardware and only have the wireshark capture file (with file extension *.pcapng), this software is still needed to view and analyze the wireless packets capture file.

To capture 802.11 wireless packets, a sniffer hardware such as the AirPcap is required. Certain configurations will not allow you to capture 802.11 wireless packets.

• A laptop/PC that has wireless network capabilities or wireless adaptor with an AP connected to this laptop/PC via wired Ethernet cable is not sufficient as this environment is likely to be in non-promiscuous mode. This means that many packets will be filtered and not shown on the wireshark trace. If using wired ethernet or the built-in PC NIC, generally only packets meant for the PC are displayed. The AirPcap allows selecting a promiscuous NIC that will display all traffic in the air.

• A laptop/PC that has wireless network capabilities or wireless adaptor and only plugged into wired Ethernet. This will only allow monitoring of Ethernet traffic and specifically traffic that is targeted to the PC, even though this laptop/PC is 802.11 wireless connected to some wireless...
devices. Also this method will prevent seeing packets that are attempting to make it to the PC or anywhere else but are not getting through due to any misconfigurations.

Below figure shows the recommended configuration for wireless packets capture.

**802.11 Protocols**

Below figure shows the 802.11 exchange protocol for a client device in an infrastructure network.

**Authentication** and association frame types are unidirectional. For example, the client device will be the one transmitting the association request frame and likewise the AP/router will be the one to respond back with the association response frame.
Display Filter

Use the Wireshark display filter to filter down to the traces of interest. Refer to Wireshark website for more information on display filter settings.

As an example, enter into the display filter

(wlan.bssid == Cisco-Li_27:5a:a0 && wlan.da == Broadcast) ||
wlan.addr == Microchi_02:75:7e

In our example, Cisco-Li_27:5a:a0 represents the AP/router and Microchi_02:75:7e represents the wireless client.

• More examples of display filter

Wlan.fc.type == 0 Management frames
Wlan.fc.type == 1 Control frames
Wlan.fc.type == 2 Data frames
Wlan.fc.type_subtype == 0 Association request
Wlan.fc.type_subtype == 1 Association response
Wlan.fc.type_subtype == 2 Reassociation request
Wlan.fc.type_subtype == 3 Reassociation response
Wlan.fc.type_subtype == 4 Probe request
Wlan.fc.type_subtype == 5 Probe response
Wlan.fc.type_subtype == 8 Beacon
Wlan.fc.type_subtype == 10 Disassociation
Wlan.fc.type_subtype == 11 Authentication
Wlan.fc.type_subtype == 12 Deauthentication

Information you are likely to need for the display filter

• BSSID of AP/router

BSSID is unique and identifies a specific AP/router eg Cisco-Li_27:5a:a0

If you can unable to determine the BSSID, an alternative is to review the list of beacons in the wireless packets capture and use this list to narrow down the BSSID.

This will translate to “wlan.bssid == Cisco-Li_27:5a:a0” in the display filter.

• MAC Address of client device eg MRF24W

This will identify the client device’s hardware address.

As an example, the console terminal when using the MRF24W will display a MAC address field.

See below figure where the MAC address is 00:1e:c0:02:75:7e.
“00:1e:c0:02:75:7e” is translated to “Microchi_02:75:7e”.

Enter “wlan.addr == Microchi_02:75:7e” in the display filter.

• Source **Address** or Destination address

This could be the following; BSSID field, address field or “Broadcast” or “Multicast”.

Beacons are broadcast frames and the destination address is set as “Broadcast”.

As an example, to display beacons transmitted by a particular AP/router, this will translate to “(wlan.bssid == Cisco-Li_27:5a:a0 && wlan.da == Broadcast) “ in the display filter.

**Procedures To Analyze 802.11 wireless packets traces**

**Step 1:** Open the wireless packets capture traces / file.
Step 2: Use the display filter to filter down to the traces of interest.

Step 3: Active or Passive Scanning

Before wireless connection is established, a client device needs to locate the specific AP/router that it wishes to join by either passive or active scanning.

Passive Scanning

Every “unhidden” AP/router will be transmitting beacons almost every beacon interval (BI). The client device is likely to receive multiple beacons from different AP/routers. A scan list result will be generated showing wireless devices or AP/routers that is present in the wireless network.

Each beacon frame will contain information about the infrastructure network such as

- Transmission rate of beacon (Based on 802.11-2007 specifications, transmitted at 1-2Mbps)
- BSSID, source address, destination address
- Destination address is often of Broadcast type.
- SSID of infrastructure network
- AP/Router supported rates
**Active Scanning**

This involves transmission of broadcast probe request frames by the wireless client device and the AP/routers responding back with a directed (unicast) probe response frames (destination is the same client device that transmits the probe request frames). Likewise, a scan list result will be generated from the probe response frames received.

“Hidden” AP/router
In the case of a “hidden” AP/router, only active scan is used and the wireless client knows the SSID of the wireless network it desires to connect to. The wireless client will transmit a directed probe request frame to this “hidden” AP/router and the “hidden” AP/router will respond back with the corresponding probe response frames.

Refer to the figure below.

Once the wireless client knows the particular infrastructure network it wishes to join, the authentication and association process will be initiated.
Step 4: **Authentication**

To initiate wireless connection to the AP/router, authentication process will have to first take place.

The wireless client device will transmit a directed authentication frame to the desired AP/router. The AP/Router will respond by transmitting a directed authentication to the same wireless client device to indicate the authentication status.

Within the authentication frame from the AP/router, it will contain

- BSSID, source address, destination address

Destination address will be the wireless device which has transmitted the authentication (directed).

- **Authentication** Algorithm

  This contained the authentication algorithm used, such as Open System.

- Status Code
Status Code will indicate whether authentication is a success or failure.

Step 5 : Association

When the authentication process is completed, this will be followed by the association process.

The wireless client device will transmit a directed association request frame to the desired AP/router. The AP/Router will respond by transmitting a directed association response frame to the same wireless device to indicate the association status.

Within the association response frame from the AP/router, it will contain

- BSSID, source address, destination address

Destination address will be the wireless device which has transmitted
the association request (directed).

- Status Code and Association ID (AID)

Status Code will indicate whether association is a success or failure. If association is successful, an unique AID will be assigned.

Integration of WPS into 802.11 joining operation

Below lists the overall sequences

- Scanning, Authentication, Association

- WPS Frame Exchanges (EAP protocol)

- Deauthentication or Disassociation

- Some APs are found to transmit disassociation instead of
deauthentication frame.

Provision needs to be made to handle receipt of disassociation frame.

- **Authentication**, Association
- EAPOL 4-way handshake or 802.1X-authentication

**Disconnection from a wireless network**

If the device is disconnected according to the 802.11 specifications, either of the following procedures will take place.

- Disassociation
- Deauthentication

Or

- Deauthentication

When an AP sends a deauthentication notice to an associated STA, the association shall also be terminated

---

**Module**

Wi-Fi API
Contents

This is the table of contents of this documentation.

Introduction
  Getting Help
  Directory Structure
SW License Agreement

Release Notes
  Stack Performance
  Memory Usage
  Peripheral Usage
Silicon Solutions

Software
  TCP/IP Configuration Wizard
  MPFS2 Utility
    Building MPFS2 Images
    Uploading Pre-built MPFS2 Images
    Advanced MPFS2 Settings
    MPFS2 Command Line Options
  Hash Table Filter Entry Calculator
  Microchip TCP/IP Discoverer

Getting Started
  Hardware Setup
    Daughter Boards
    PICDEM.net 2
    PIC18 Explorer
    Explorer 16 and PIC32 Starter Kit
    PIC24FJ256DA210 Dev Board
    Wi-Fi G Demo Board
  Programming and First Run
  Configure your WiFi Access Point
  Connecting to the Network
  Uploading Web Pages
  Accessing the Demo Application
  Configuring WiFi Security

Demo Information
  Demo Compatibility Table

Available Demos
  Demo App
    TCPIP Demo App Features by Hardware Platform
Demo Modules

Web Page Demos
Dynamic Variables
Authentication
Forms using GET
Forms using POST
Cookies

Functions
HTTPPostImage Function
HTTPPostConfig Function
HTTPPostSNMPCommunity Function
HTTPPostDDNSConfig Function
HTTPPostEmail Function
HTTPPostLCD Function
HTTPPostMD5 Function

Variables
Flag_ImageUpdate_running Variable
ImageUpdate_Addr Variable
DDNSData Variable
ImageUpdate_Checksum Variable
ImageUpdate_Size Variable
lastFailure Variable
lastSuccess Variable

E-mail (SMTP) Demo
SMTPDemo Function

Generic TCP Client
GenericTCPClient Function

Variables
RemoteURL Variable
ServerName Variable
ServerPort Variable

Generic TCP Server
GenericTCPServer Function

Macros
SERVER_PORT Macro

Ping (ICMP) Demo
PingDemo Function

Macros
HOST_TO_PING Macro

SNMP Server (Agent)
MIB files
MIB Browsers
SNMP Operations
SNMP Traps
HTTP Configuration

Functions
SendNotification Function
SNMPGetTimeStamp Function

Variables
gSendTrapSMstate Variable
gSnmpNonMibRecInfo Variable
gSnmpv3UserSecurityName Variable
gtrapSMStateUpdate Variable

Macros
MAX_TRY_TO_SEND_TRAP Macro
SNMP_MAX_NON_REC_ID_OID Macro
STACK_USE_SMIV2 Macro

UART-to-TCP Bridge
Zero Configuration (ZeroConf)

Internet Bootloader
Bootloader Design
Using the Bootloader
WebVend
Internet Radio

WiFi Console
Standalone Commands
iwconfig Commands
ifconfig Commands
iwpriv Commands
iperf Example
WiFi EZConfig
Demo App MDD
Google PowerMeter
Energy Monitoring
WiFi G Demo

Using the Stack
Stack Architecture

How the Stack Works
Required Files
APP_CONFIG Structure

Main File
Initialization
Main Loop
Cooperative Multitasking
RTOS
Configuring the Stack
Hardware Configuration
  Clock Frequency
  External Storage
  ENC28J60 Config
  ENCX24J600 Config
  PIC18F97J60 Config
  PIC32MX7XX Config
Address
  MAC Address
  IP Address
Protocol Configuration
  Protocol Macros and Files
  Additional Features
Sockets
  Memory Allocation
  Socket Types
  Initialization Structure
  UDP Sockets
  BSD Sockets
Stack API
Announce
  Stack Members
    AnnounceIP Function
    DiscoveryTask Function
ARP
Public Members
  ARPResolve Function
  ARPIsResolved Function
  ARPDeRegisterCallbacks Function
  ARPRegisterCallbacks Function
  ARPSendPkt Function
  arp_app_callbacks Structure
  ARP_REQ Macro
  ARP_RESP Macro
  MAX_REG_APPS Macro
Stack Members
  ARPInt Function
  ARPProcess Function
Internal Members

ARPPut Function
SwapARPPacket Function
ARP_OPERATION_REQ Macro
ARP_OPERATION_RESP Macro
HWEtherNET Macro
ARP_IP Macro
Cache Variable
reg_apps Variable

BSD Sockets

Public Members

accept Function
AF_INET Macro
bind Function
BSDSocket Structure
closesocket Function
connect Function
gethostname Function
in_addr Structure
INADDR_ANY Macro
INVALID_TCP_PORT Macro
IP_ADDR_ANY Macro
IPPROTO_IP Macro
IPPROTO_TCP Macro
IPPROTO_UDP Macro
listen Function
recv Function
recvfrom Function
send Function
sendto Function
SOCK_DGRAM Macro
SOCK_STREAM Macro
sockaddr Structure
SOCKADDR Type
sockaddr_in Structure
SOCKADDR_IN Type
socket Function
SOCKET Type
SOCKET_CNXN_IN_PROGRESS Macro
SOCKET_DISCONNECTED Macro
SOCKET_ERROR Macro

Stack Members
BerkeleySocketInit Function

**Internal Members**
- BSD_SCK_STATE Enumeration
- BSDSocketArray Variable
- gAutoPortNumber Variable
- HandlePossibleTCPDisconnection Function

**DNS**

**Public Members**
- DNSBeginUsage Function
- DNSEndUsage Function
- DNSResolve Function
- DNSResolveROM Function
- DNSIsResolved Function
- DNS_TYPE_A Macro
- DNS_TYPE_MX Macro

**Internal Members**
- DNSPutString Function
- DNSPutROMString Function
- DNS_PORT Macro
- DNS_TIMEOUT Macro
- DNSHostName Variable
- DNSHostNameROM Variable
- Flags Variable
- RecordType Variable
- ResolvedInfo Variable
- smDNS Variable
- DNS_HEADER Structure
- DNSDiscardName Function

**Dynamic DNS Client**

**Public Members**
- DDNS_POINTERS Structure
- DDNS_SERVICES Enumeration
- DDNS_STATUS Enumeration
- DDNSClient Variable
- DDNSForceUpdate Function
- DDNSGetLastIP Function
- DDNSGetLastStatus Function
- DDNSSetService Function

**Stack Members**
- DDNSInit Function
- DDNSTask Function

**Internal Members**
bForceUpdate Variable
dDNSServiceHosts Variable
dDNSServicePorts Variable
dwUpdateAt Variable
lastKnownIP Variable
lastStatus Variable
_checkIpSrvrResponse Variable
_updateIpSrvrResponse Variable
DDNS_CHECKIP_SERVER Macro
DDNS_DEFAULT_PORT Macro

Hashes

Public Members
HashAddData Function
HashAddROMData Function
MD5Calculate Function
MD5Initialize Function
SHA1Calculate Function
SHA1Initialize Function
HASH_SUM Structure

Stack Members
MD5AddROMData Function
SHA1AddROMData Function
SHA1AddData Function
MD5AddData Function

Internal Members
_MD5_k Variable
_MD5_r Variable
lastBlock Variable
HASH_TYPE Enumeration
SHA1HashBlock Function
MD5HashBlock Function

Helpers

Public Members
Base64Decode Function
Base64Encode Function
btohexa_high Function
btohexa_low Function
CalcIPChecksum Function
ExtractURLFields Function
FormatNetBIOSName Function
GenerateRandomDWORD Function
hexatob Function
leftRotateDWORD Function
leftRotateDWORD Macro
Replace Function
ROMStringToIPAddress Function
ROMStringToIPAddress Macro
stricmpgm2ram Function
StringToIPAddress Function
strupr Function
strnchr Function
swapl Function
swaps Function
ultoa Function
uitoa Function
UnencodeURL Function

Functions
LFSRRand Function
LFSRSeedRand Function
strncpy_m Function

Variables
dwLFSRRandSeed Variable

HTTP2 Server

Features
Dynamic Variables
Form Processing
Authentication
Cookies
Compression

Public Members
curHTTP Variable
HTTP_CONN Structure
HTTP_IO_RESULT Enumeration
HTTP_READ_STATUS Enumeration
HTTPPrint_varname Function
HTTPExecuteGet Function
HTTPExecutePost Function
HTTPGetArg Function
HTTPGetROMArg Function
HTTPNeedsAuth Function
HTTPPrint_varname Function
HTTPReadPostName Function
HTTPReadPostPair Macro
HTTPReadPostValue Function
HTTPURLDecode Function
sktHTTP Macro

Stack Members
HTTPInit Function
HTTPServer Function

Internal Members
curHTTPID Variable
HTTP_CACHE_LEN Macro
HTTP_FILE_TYPE Enumeration
HTTP_MAX_DATA_LEN Macro
HTTP_MAX_HEADER_LEN Macro
HTTP_MIN_CALLBACK_FREE Macro
HTTP_PORT Macro
HTTP_STATUS Enumeration
HTTP_STUB Structure
HTTP_TIMEOUT Macro
httpContentTypes Variable
httpFileExtensions Variable
HTTPHeaderParseAuthorization Function
HTTPHeaderParseContentLength Function
HTTPHeaderParseCookie Function
HTTPHeaderParseLookup Function
HTTPIncFile Function
HTTPLoadConn Function
HTTMPMFUpload Function
HTTPProcess Function
HTTPReadTo Function
HTTPRequestHeaders Variable
HTTPResponseHeaders Variable
HTTPS_PORT Macro
HTTPSendFile Function
httpStubs Variable
SM_HTTP2 Enumeration
smHTTP Macro
RESERVED_HTTP_MEMORY Macro

ICMP

Public Members
ICMPBeginUsage Function
ICMPSendPing Function
ICMPSendPingToHost Function
ICMPSendPingToHostROM Function
ICMPGetReply Function
ICMPEndUsage Function
ICMPSendPingToHostROM Macro

Internal Members
ICMPProcess Function
ICMPFlags Variable
ICMP_PACKET Structure
ICMPState Variable
ICMP_TIMEOUT Macro
ICMPTimer Variable
StaticVars Variable
wICMPSequenceNumber Variable

MPFS2

Public Members
MPFS_HANDLE Type
MPFS_INVALID Macro
MPFS_INVALID_HANDLE Macro
MPFS_SEEK_MODE Enumeration
MPFSClose Function
MPFSFormat Function
MPFSGet Function
MPFSGetArray Function
MPFSGetBytesRem Function
MPFSGetEndAddr Function
MPFSGetFilename Function
MPFSGetFlags Function
MPFSGetID Function
MPFSGetLong Function
MPFSGetMicrotime Function
MPFSGetPosition Function
MPFSGetSize Function
MPFSGetStartAddr Function
MPFSGetTimestamp Function
MPFSOpen Function
MPFSOpenID Function
MPFSOpenROM Function
MPFSPutArray Function
MPFSSeek Function
MPFSPutEnd Function

Stack Members
MPFSInit Function

Internal Members
isMPFSLocked Variable
lastRead Variable
MAX_FILE_NAME_LEN Macro
MPFS_PTR Type
MPFS_STUB Structure
MPFS_WRITE_PAGE_SIZE Macro
MPFS2_FLAG_HASINDEX Macro
MPFS2_FLAG_ISZIPPED Macro
MPFSStubs Variable
MPFSTell Macro
ReadProgramMemory Function
_LoadFATRecord Function
_VALIDATE Function
MPFS_FAT_RECORD Structure
fatCache Variable
fatCacheID Variable
numFiles Variable
MPFS_INVALID_FAT Macro

NBNS
Stack Members
NBNSGetName Function
NBNSPutName Function
NBNSTask Function
NBNS_HEADER Structure
NBNS_PORT Macro

Performance Tests
Stack Members
TCPPerformanceTask Function
UDPPerformanceTask Function

Internal Members
TCPRXPerformanceTask Function
TCPTXPerformanceTask Function
PERFORMANCE_PORT Macro
RX_PERFORMANCE_PORT Macro
TX_PERFORMANCE_PORT Macro

SMTP Client
Examples
Short Message
Long Message
Public Members
SMTP_CONNECT_ERROR Macro
SMTP_POINTERS Structure
SMTP_RESOLVE_ERROR Macro
SMTP_SUCCESS Macro
SMTPBeginUsage Function
SMTPClient Variable
SMTPEndUsage Function
SMTPFlush Function
SMTPIsBusy Function
SMTPIsPutReady Function
SMTPPut Function
SMTPPutArray Function
SMTPPutDone Function
SMTPPutROMArray Function
SMTPPutROMString Function
SMTPPutString Function
SMTPSendMail Function

Stack Members
SMPTPTask Function

Internal Members
CRPeriod Variable
FindEmailAddress Function
FindROMEmailAddress Function
MySocket Variable
PutHeadersState Variable
ResponseCode Variable
RXParserState Variable
SMTP_PORT Macro
SMTP_SERVER_REPLY_TIMEOUT Macro
SMTPFlags Variable
SMTPServer Variable
SMTPState Variable
TransportState Variable

Reboot

Stack Members
RebootTask Function
REBOOT_PORT Macro
REBOOTSAME_SUBNET_ONLY Macro

SNMP

Public Members
GENERIC_TRAP_NOTIFICATION_TYPE Enumeration
VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE Enumeration
SNMP_ACTION Enumeration
COMMUNITY_TYPE Enumeration
SNMP_VAL Union
TRAP_INFO Structure
gSendTrapFlag Variable
gSetTrapSendFlag Variable
gGenericTrapNotification Variable
gSpecificTrapNotification Variable
gOIDCorrespondingSnmpMibID Variable
SNMPSendTrap Function
SNMPNotify Function
SNMPSetVar Function
SNMPGetVar Function
SNMPIsNotifyReady Function
SNMPNotifyPrepare Function
SNMPGetNextIndex Function
SNMPValidateCommunity Function
SNMP_ID Type
SNMP_INDEX Type
SNMP_COMMUNITY_MAX_LEN Macro
OID_MAX_LEN Macro
SNMP_START_OF_VAR Macro
SNMP_END_OF_VAR Macro
SNMP_INDEX_INVALID Macro
TRAP_TABLE_SIZE Macro
TRAP_COMMUNITY_MAX_LEN Macro
NOTIFY_COMMUNITY_LEN Macro

Internal Members
_SNMPDuplexInit Function
_SNMPGet Function
_SNMPGetTxOffset Macro
_SNMPPut Function
_SNMPSetTxOffset Macro
AGENT_NOTIFY_PORT Macro
appendZeroToOID Variable
ASN_INT Macro
ASN_NULL Macro
ASN_OID Macro
DATA_TYPE Enumeration
DATA_TYPE_INFO Structure
DATA_TYPE_TABLE_SIZE Macro
dataTypeTable Variable
FindObjectInRequest Function
GET_BULK_REQUEST Macro
GET_NEXT_REQUEST Macro
GET_REQUEST Macro
GET_RESPONSE Macro
hMPFS Variable
INDEX_INFO Union
IS_AGENT_PDU Macro
IS_ASN_INT Macro
IS_ASN_NULL Macro
IS_GET_NEXT_REQUEST Macro
IS_GET_REQUEST Macro
IS_GET_RESPONSE Macro
IS_OCTET_STRING Macro
IS_OID Macro
GetDataTypeInfo Function
IS_SET_REQUEST Macro
IS_STRUCTURE Macro
IS_TRAP Macro
IsASNNull Function
MIB_INFO Union
OCTET_STRING Macro
OID_INFO Structure
PDU_INFO Structure
reqVarErrStatus Structure
SET_REQUEST Macro
SetErrorStatus Function
SNMP_AGENT_PORT Macro
SNMP_BIB_FILE_NAME Macro
SNMP_COUNTER32 Macro
SNMP_ERR_STATUS Enumeration
SNMP_GAUGE32 Macro
SNMP_IP_ADDR Macro
SNMP_NMS_PORT Macro
SNMP_NOTIFY_INFO Structure
SNMP_NSAP_ADDR Macro
IsValidLength Function
SNMP_OPAQUE Macro
SNMP_STATUS Union
SNMP_TIME_TICKS Macro
SNMP_V1 Macro
SNMP_V2C Macro
SNMPAgentSocket Variable
SNMPNotifyInfo Variable
snmpReqVarErrStatus Variable
SNMPRxOffset Variable
SNMPStatus Variable
SNMPTxOffset Variable
STRUCTURE Macro
TRAP Macro
trapInfo Variable
GetNextLeaf Function
GetOIDStringByAddr Function
GetOIDStringByID Function
IsValidCommunity Function
IsValidInt Function
IsValidOID Function
IsValidPDU Function
IsValidStructure Function
OIDLookup Function
ProcessGetSetHeader Function
ProcessHeader Function
ProcessSetVar Function
ProcessVariables Function
ReadMIBRecord Function
SNMPCheckIfPvtMibObjRequested Function

**Stack Members**
- SNMPInit Function
- SNMPTask Function

**Functions**
- _IsSNMPv3ValidStructure Function
- _Snmpv3IsValidAuthStructure Function
- _Snmpv3IsValidInt Function
- FindOIDsFromSnmpV3Request Function
- getSnmpV2GenTrapOid Function
- IsSnmpV3ASNNull Function
- IsSnmpv3ValidOID Function
- IsSNMPv3ValidStructure Function
- ProcessGetBulkVar Function
- ProcessGetNextVar Function
- ProcessGetVar Function
- ProcessSnmpv3MsgData Function
- SNMPGetExactIndex Function
- SNMPGetTrapTime Function
- SNMPIdRecrdValidation Function
- SNMPIsValidSetLen Function
- Snmpv3AESDecryptRxedScopedPdu Function
Snmpv3AESEncryptResponseScopedPdu Function
Snmpv3AuthenticateRxedPduForDataIntegrity Function
Snmpv3AuthenticateTxPduForDataIntegrity Function
Snmpv3AuthKeyZeroing2HmacBufLen64 Function
Snmpv3BufferPut Function
Snmpv3CmprTrapSecNameAndSecLvWithUSMDb Function
Snmpv3ComputeHMACIpadOpadForAuthLoclzedKey Function
Snmpv3ComputeHmacMD5Digest Function
Snmpv3ComputeHmacShaDigest Function
Snmpv3ComputeMd5HmacCode Function
Snmpv3ComputeShaHmacCode Function
Snmpv3FormulateEngineID Function
Snmpv3FreeDynAllocMem Function
Snmpv3GetAuthEngineTime Function
Snmpv3GetBufferData Function
Snmpv3GetSecurityLevel Function
Snmpv3GetTrapSecurityLevel Function
Snmpv3Init Function
Snmpv3InitializeUserDataBase Function
Snmpv3IsValidAuthStructure Function
Snmpv3IsValidInt Function
Snmpv3MsgProcessingModelProcessPDU Function
Snmpv3Notify Function
Snmpv3Pswd2LocalizedAuthKeyMD5Hashing Function
Snmpv3Pswd2LocalizedAuthKeyShaHashing Function
Snmpv3ReportPdu Function
Snmpv3ScopedPduProcessing Function
Snmpv3SetErrorStatus Function
Snmpv3TrapScopedpdu Function
Snmpv3UserSecurityModelProcessPDU Function
Snmpv3UsmAesEncryptDecryptInitVector Function
Snmpv3UsmOutMsgAuthenticationParam Function
Snmpv3USMOutMsgPrivParam Function
Snmpv3UsmSnmpEngnAuthPrivPswdLocalization Function
Snmpv3ValidateEngineId Function
Snmpv3ValidateSecNameAndSecLv Function
Snmpv3ValidateSecurityName Function

**Structs, Records, Enums**

AccessCtrlSubSysIsAccessAllowed Structure
dispatcherProcessPdu Structure
dispatcherStatusInfo Structure
dispatchererrReturnResponsePdu Structure
MsgProcModPrepareDataElements Structure
MsgProcModPrepareOutgoingMessage Structure
MsgProcModPrepareResponseMessage Structure
processResponsePdu Structure
SecuritySysGenerateRequestMsg Structure
SecuritySysGenerateResponseMsg Structure
StateRelease Structure
unregisterContextEngineID Structure

**Types**

INOUT_SNMP_PDU Enumeration
REPORT_FLAG_AND_SECURITY_LEVEL_FLAGS Enumeration
SecuritySysProcessIncomingMsg Structure
SNMP_ENGNID_OCTET_IDENTIFIER_VAL Enumeration
SNMPNONMIBRECDINFO Structure
SNMPV3_HMAC_HASH_TYPE Enumeration
SNMPV3_MSG_AUTH_SEC_PARAM_RESULT Enumeration
SNMPV3_MSG_PRIV_SEC_PARAM_RESULT Enumeration
SNMPV3_PRIV_PROT_TYPE Enumeration
SNMPV3_REQUEST_WHOLEMSG Structure
SNMPV3_RESPONSE_WHOLEMSG Structure
snmpV3EngnUserDataBase Structure
SNMPV3MSGDATA Structure
snmpV3TrapConfigDataBase Structure
statusInformation Structure
STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL Enumeration
STD_BASED_SNMP_SECURITY_MODEL Enumeration
STD_BASED_SNMPV3_SECURITY_LEVEL Enumeration
USM_SECURITY_LEVEL Enumeration

**Variables**

authKey_iPad Variable
authKey_oPad Variable
authoritativeSnmpEngineBoots Variable
authoritativeSnmpEngineTime Variable
cipher_text Variable
deciphered_text Variable
getZeroInstance Variable
gSnmpV3InPduWholeMsgBuf Variable
gSnmpV3OUTPduWholeMsgBuf Variable
gSNMPv3PduHeaderBuf Variable
gSNMPV3ScopedPduDataPos Variable
gSNMPv3ScopedPduRequestBuf Variable
gSNMPv3ScopedPduResponseBuf Variable
gSnmpv3TrapConfigData Variable
gSNMPv3TrapMsgHeaderBuf Variable
gSnmpV3TrapOUTPduWholeMsgBuf Variable
gSNMPv3TrapScopedPduResponseBuf Variable
gSNMPV3TrapSecurityLevel Variable
gSnmpv3UserDBIndex Variable
gUsmStatsEngineID Variable
hmacAuthKeyBuf Variable
HmacMd5Digest Variable
HmacSHADigest Variable
incomingPdu Variable
incomingSnmpPDUmsgID Variable
ivEncrptKeyOut Variable
md5LocalizedAuthKey Variable
msgSecrtyParamLenOffset Variable
securityPrimitivesOfIncomingPdu Variable
session_key Variable
sha1LocalizedAuthKey Variable
snmpEngineBoots Variable
snmpEngineID Variable
snmpEngineMaxMessageSize Variable
snmpEngineMsgProcessModel Variable
snmpEngineSecurityModel Variable
snmpEngineTime Variable
snmpEngineTimeOffset Variable
snmpEngn1DLength Variable
snmplnMsgAuthParamLen Variable
snmplnMsgAuthParamString Variable
snmplnMsgPrivParamLen Variable
snmplnMsgPrvParamString Variable
snmpMsgBufSeekPos Variable
snmpOutMsgAuthParamLen Variable
snmpOutMsgAuthParamString Variable
snmpOutMsgPrivParamLen Variable
snmpOutMsgPrvParamString Variable
snmpResponseSecurityFlag Variable
snmpSecurityLevel Variable
snmpTrapTimer Variable
snmpV3AesDecryptInitVector Variable
snmpV3AesEncryptInitVector Variable
snmpV3UserDataBase Variable

Macros
AUTH_LOCALIZED_PASSWORD_KEY_LEN Macro
INVALID_INDEX Macro
IS_SNMPV3_AUTH_STRUCTURE Macro
MSG_AUTHORITATIVE_HEADER_LEN Macro
MSGGLOBAL_HEADER_LEN Macro
PRIV_LOCALIZED_PASSWORD_KEY_LEN Macro
REPORT_RESPONSE Macro
SNMP_ENGINE_MAX_MSG_SIZE Macro
SNMP_MAX_MSG_SIZE Macro
SNMP_MAX_OID_LEN_MEM_USE Macro
SNMP_trap_community_MAX_LEN_MEM_USE Macro
SNMP_V3 Macro
SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE Macro
SNMPV3_H Macro
SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE Macro
SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE Macro
SNMPV3_USM_MAX_USER Macro
SNMPV3MSG_AUTHENTICATION_FAIL Macro
SNMPV3MSG_AUTHENTICATION_SUCCESS Macro
USER_SECURITY_NAME_LEN Macro

Files
SNMPv3.c
SNMPv3.h
SNMPv3USM.c
SNMP.c
SNMP.h

SNTP Client

Public Members
SNTPGetUTCSeconds Function

Stack Members
SNTPClient Function

Internal Members
NTP_PACKET Structure
dwLastUpdateTick Variable
dwSNTPSeconds Variable
NTP_EPOCH Macro
NTP_FAST_QUERY_INTERVAL Macro
NTP_QUERY_INTERVAL Macro
NTP_REPLY_TIMEOUT Macro
NTP_SERVER Macro
NTP_SERVER_PORT Macro

SSL
Generating Server Certificates

**Public Members**
- SSL_INVALID_ID Macro
- TCPAddSSLListener Function
- TCPSSLIsHandshaking Function
- TCPStartSSLClient Function
- TCPIsSSL Function
- SSLStartSession Function
- SSL_SUPPLEMENTARY_DATA_TYPES Enumeration
- SSL_PKEY_INFO Structure
- SSL_RSA_KEY_SIZE Macro
- SSL_RSA_CLIENT_SIZE Macro

**Stack Members**
- SSL_STATE Enumeration
- SSLInit Function
- SSLPeriodic Function
- TCPRequestSSLMessage Function
- TCPSSLGetPendingTxSize Function
- TCPSSLHandleIncoming Function
- TCPSSLHandshakeComplete Function
- TCPSSLInPlaceMACEncrypt Function
- TCPSSLPutRecordHeader Function
- TCPStartSSLServer Function
- SSL_MIN_SESSION_LIFETIME Macro
- SSL_RSA_LIFETIME_EXTENSION Macro

**Internal Members**
- CalculateFinishedHash Function
- GenerateHashRounds Function
- GenerateSessionKeys Function
- HSEnd Function
- HSGet Function
- HSGetArray Function
- HSGetWord Function
- HSPut Function
- HSPutArray Function
- HSPutROMArray Function
- HSPutWord Function
- HSStart Function
- isBufferUsed Variable
- isHashUsed Variable
- isStubUsed Variable
- masks Variable
ptrHS Variable
RESERVED_SSL_MEMORY Macro
LoadOffChip Function
SaveOffChip Function
SM_SSL_RX_SERVER_HELLO Enumeration
SSL_ALERT Macro
SSL_ALERT_LEVEL Enumeration
SSL_APPLICATION Macro
SSL_BASE_BUFFER_ADDR Macro
SSL_BASE_HASH_ADDR Macro
SSL_BASE_KEYS_ADDR Macro
SSL_BASE_SESSION_ADDR Macro
SSL_BASE_STUB_ADDR Macro
SSL_BUFFER Union
SSL_BUFFER_SIZE Macro
SSL_BUFFER_SPACE Macro
SSL_CERT Variable
SSL_CERT_LEN Variable
SSL_CHANGE_CIPHER_SPEC Macro
SSL_HANDSHAKE Macro
SSL_HASH_SIZE Macro
SSL_HASH_SPACE Macro
SSL_KEYS Structure
SSL_KEYS_SIZE Macro
SSL_KEYS_SPACE Macro
SSL_MESSAGES Enumeration
SSL_RSA_EXPORT_WITH_ARCFOUR_40_MD5 Macro
SSL_RSA_WITH_ARCFOUR_128_MD5 Macro
SSL_SESSION Structure
SSL_SESSION_SIZE Macro
SSL_SESSION_SPACE Macro
SSL_SESSION_STUB Structure
SSL_SESSION_TYPE Enumeration
SSL_STUB Structure
SSL_STUB_SIZE Macro
SSL_STUB_SPACE Macro
SSL_VERSION Macro
SSL_VERSION_HI Macro
SSL_VERSION_LO Macro
SSLBufferAlloc Function
SSLBufferFree Function
sslBufferID Variable
SSLBufferSync Function
SSLFinishPartialRecord Macro
SSLFlushPartialRecord Macro
sslHash Variable
SSLHashAlloc Function
SSLHashFree Function
sslHashID Variable
SSLHashSync Function
sslKeys Variable
sslKeysID Variable
SSLKeysSync Function
SSLMACAdd Function
SSLMACBegin Function
SSLMACCalc Function
SSLRSAOperation Function
sslRSAStubID Variable
SSLrxAlert Function
SSLrxAntiqueClientHello Function
SSLrxCCS Function
SSLrxClientHello Function
SSLrxClientKeyExchange Function
SSLrxFinished Function
SSLrxHandshake Function
SSLrxRecord Function
SSLrxServerCertificate Function
SSLrxServerHello Function
sslSession Variable
sslSessionID Variable
SSLSessionMatchID Function
SSLSessionMatchIP Function
SSLSessionNew Function
sslSessionStubs Variable
SSLSessionSync Function
SSLSessionUpdated Macro
sslSessionUpdated Variable
SSLStartPartialRecord Function
sslStub Variable
SSLStubAlloc Function
SSLStubFree Function
sslStubID Variable
SSLStubSync Function
SSLTerminate Function
SSLtxCCSFin Function
SSLtxClientHello Function
SSLtxClientKeyExchange Function
SSLtxMessage Function
SSLtxRecord Function
SSLtxServerCertificate Function
SSLtxServerHello Function
SSLtxServerHelloDone Function

Files
SSLClientSize.h

TCP

Public Members
INVALID_SOCKET Macro
UNKNOWN_SOCKET Macro
TCP_ADJUST_GIVE_REST_TO_RX Macro
TCP_ADJUST_GIVE_REST_TO_TX Macro
TCP_ADJUST_PRESERVE_RX Macro
TCP_ADJUST_PRESERVE_TX Macro
TCP_OPEN_IP_ADDRESS Macro
TCP_OPEN_NODE_INFO Macro
TCP_OPEN_RAM_HOST Macro
TCP_OPEN_ROM_HOST Macro
TCP_OPEN_SERVER Macro
TCPAdjustFIFOSize Function
TCPConnect Macro
TCPClose Function
TCPDiscard Function
TCPDisconnect Function
TCPFind Macro
TCPFindArray Macro
TCPFindArrayEx Function
TCPFindEx Function
TCPFindROMArray Macro
TCPFindROMArrayEx Function
TCPFlush Function
TCPGet Function
TCPGetArray Function
TCPGetRemoteInfo Function
TCPGetRxFIFOFree Function
TCPGetRxFIFOFull Macro
TCPGetTxFIFOFree Macro
TCPGetTxFIFOFull Function
TCPIsConnected Function
TCPIsGetReady Function
TCPIsPutReady Function
TCPListen Macro
TCPOpen Function
TCPPeek Function
TCPPeekArray Function
TCPPut Function
TCPPutArray Function
TCPPutROMArray Function
TCPPutROMString Function
TCPPutString Function
TCPRAMCopy Function
TCPRAMCopyROM Function
TCPWasReset Function

Stack Members
SOCKET_INFO Structure
TCB Structure
TCB_STUB Structure
TCP_SOCKET Type
TCP_STATE Enumeration
TCPInit Function
TCPProcess Function
TCPTick Function
TCPSSLDecryptMAC Function
TCPStartSSLClientEx Function

Internal Members
ACK Macro
CloseSocket Function
FIN Macro
FindMatchingSocket Function
HandleTCPSeg Function
hCurrentTCP Variable
LOCAL_PORT_END_NUMBER Macro
LOCAL_PORT_START_NUMBER Macro
MyTCB Variable
MyTCBStub Variable
PSH Macro
RST Macro
SendTCP Function
SENDTCP_KEEP_ALIVE Macro
SENDTCP_RESET_TIMERS Macro
SwapTCPHeader Function
SYN Macro
SyncTCB Function
SyncTCBStub Macro
SYNQueue Variable
TCBStubs Variable
TCP_AUTO_TRANSMIT_TIMEOUT_VAL Macro
TCP_WINDOW_UPDATE_TIMEOUT_VAL Macro
TCP_CLOSE_WAIT_TIMEOUT Macro
TCP_DELAYED_ACK_TIMEOUT Macro
TCP_FIN_WAIT_2_TIMEOUT Macro
TCP_HEADER Structure
TCP_KEEP_ALIVE_TIMEOUT Macro
TCP_MAX_RETRIES Macro
TCP_MAX_SEG_SIZE_RX Macro
TCP_MAX_SEG_SIZE_TX Macro
TCP_MAX_SYN_RETRIES Macro
TCP_MAX_UNACKED_KEEP_ALIVES Macro
TCP_OPTIMIZE_FOR_SIZE Macro
TCP_OPTIONS Structure
TCP_OPTIONS_END_OF_LIST Macro
TCP_OPTIONS_MAX_SEG_SIZE Macro
TCP_OPTIONS_NO_OP Macro
TCP_SOCKET_COUNT Macro
TCP_START_TIMEOUT_VAL Macro
TCP_SYN_QUEUE Structure
TCP_SYN_QUEUE_MAX_ENTRIES Macro
TCP_SYN_QUEUE_TIMEOUT Macro
URG Macro

Functions
WFGetTCBSize Function

TFTP

Public Members
TFTPClose Macro
TFTPCloseFile Function
TFTPGet Function
TFTPGetError Macro
TFTPIsFileClosed Function
TFTPIsFileOpened Function
TFTPIsFileOpenReady Macro
TFTPIsGetReady Function
TFTPOpened Function
TFTPIsPutReady Function
TFTPOpen Function
TFTPOpenFile Function
TFTPOpenROMFile Function
TFTPPut Function
TFTP_ACCESS_ERROR Enumeration
TFTP_FILE_MODE Enumeration
TFTP_RESULT Enumeration
TFTPGetUploadStatus Function
TFTPUploadFragmentedRAMFileToHost Function
TFTPUploadRAMFileToHost Function
TFTP_CHUNK_DESCRIPTOR Structure
TFTP_UPLOAD_COMPLETE Macro
TFTP_UPLOAD_CONNECT Macro
TFTP_UPLOAD_CONNECT_TIMEOUT Macro
TFTP_UPLOAD_GET_DNS Macro
TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT Macro
TFTP_UPLOAD_RESOLVE_HOST Macro
TFTP_UPLOAD_SEND_DATA Macro
TFTP_UPLOAD_SEND_FILENAME Macro
TFTP_UPLOAD_SERVER_ERROR Macro
TFTP_UPLOAD_WAIT_FOR_CLOSURE Macro

Stack Members
  TFTP_ARP_TIMEOUT_VAL Macro
  TFTP_GET_TIMEOUT_VAL Macro
  TFTP_MAX_RETRIES Macro

Internal Members
  MutExVar Variable
  TFTP_BLOCK_SIZE Macro
  TFTP_BLOCK_SIZE_MSB Macro
  TFTP_CLIENT_PORT Macro
  TFTP_OPCODE Enumeration
  TFTP_SERVER_PORT Macro
  TFTP_STATE Enumeration
  _tftpError Variable
  _tftpFlags Variable
  _tftpRetries Variable
  _TFTPSendAck Function
  _TFTPSendFileName Function
  _TFTPSendROMFileName Function
  _tftpSocket Variable
  _tftpStartTick Variable
_tftpState Variable
smUpload Variable
uploadChunkDescriptor Variable
uploadChunkDescriptorForRetransmit Variable
vUploadFilename Variable
vUploadRemoteHost Variable
wUploadChunkOffset Variable
wUploadChunkOffsetForRetransmit Variable

Tick

Public Members
TICK Variable
TICK_HOUR Macro
TICK_MINUTE Macro
TICK_SECOND Macro
TickConvertToMilliseconds Function
TickGet Function
TickGetDiv256 Function
TickGetDiv64K Function

Stack Functions
TickInit Function
TickUpdate Function

Internal Members
dwInternalTicks Variable
GetTickCopy Function
TICKS_PER_SECOND Macro

Functions
__attribute__ Function

UDP

Public Members
INVALID_UDP_PORT Macro
INVALID_UDP_SOCKET Macro
UDP_SOCKET Type
UDPOpenEx Function
UDPOpen Macro
UDPClose Function
UDPDiscard Function
UDPFlush Function
UDPGet Function
UDPGetArray Function
UDPIsGetReady Function
UDPIsPutReady Function
UDPPut Function
UDPPutArray Function
UDPPutROMArray Function
UDPPutROMString Function
UDPPutString Function
UDPSetRxBuffer Function
UDPSetTxBuffer Function
UDPIsOpened Function
UDP_OPEN_IP_ADDRESS Macro
UDP_OPEN_NODE_INFO Macro
UDP_OPEN_RAM_HOST Macro
UDP_OPEN_ROM_HOST Macro
UDP_OPEN_SERVER Macro

Stack Members
UDPInit Function
UDPProcess Function
UDPTask Function

Internal Members
activeUDPSocket Variable
FindMatchingSocket Function
LastPutSocket Variable
LOCAL_UDP_PORT_END_NUMBER Macro
LOCAL_UDP_PORT_START_NUMBER Macro
SocketWithRxData Variable
UDP_HEADER Structure
UDP_PORT Type
UDP_SOCKET_INFO Structure
UDPRxCount Variable
UDPSocketInfo Variable
UDPTxCount Variable
wGetOffset Variable
wPutOffset Variable

Types
UDP_STATE Enumeration

Wi-Fi API
Wi-Fi Compilation Options

Wi-Fi Network Topologies
Infrastructure Network
Ad-hoc Network
SoftAP Network
Wi-Fi Direct Network

Wi-Fi Connection Profile
Connection Profile Public Members
WF_CPCreate Function
WF_CPDelete Function
WF_CPGetAdHocBehavior Function
WF_CPGetBssid Function
WF_CPGetElements Function
WF_CPGetIds Function
WF_CPGetNetworkType Function
WF_CPGetSecurity Function
WF_CPGetSsid Function
WF_CPSetAdHocBehavior Function
WF_CPSetBssid Function
WF_CPSetElements Function
WF_CPSetNetworkType Function
WF_CPSetSecurity Function
WF_CPSetSsid Function
WFCPElementsStruct Structure
WF_CPGetSsidType Function
WF_CPSetSsidType Function

**Connection Profile Internal Members**

LowLevel_CPGetElement Function
LowLevel_CPSetElement Function

**Wi-Fi Connection Algorithm**

**Connection Algorithm Public Members**

WF_CAGetBeaconTimeout Function
WF_CAGetBeaconTimeoutAction Function
WF_CAGetChannelList Function
WF_CAGetConnectionProfileList Function
WF_CAGetDeauthAction Function
WF_CAGetElements Function
WF_CAGetEventNotificationAction Function
WF_CAGetListenInterval Function
WF_CAGetListRetryCount Function
WF_CAGetMaxChannelTime Function
WF_CAGetMinChannelTime Function
WF_CAGetProbeDelay Function
WF_CAGetRssi Function
WF_CAGetScanCount Function
WF_CAGetScanType Function
WF_CASetBeaconTimeout Function
WF_CASetBeaconTimeoutAction Function
WF_CASetChannelList Function
WF_CASetConnectionProfileList Function
WF_CASetDeauthAction Function
WF_CASetElements Function
WF_CASetEventNotificationAction Function
WF_CASetListenInterval Function
WF_CASetListRetryCount Function
WF_CASetMaxChannelTime Function
WF_CASetMinChannelTime Function
WF_CASetProbeDelay Function
WF_CASetRssi Function
WF_CASetScanCount Function
WF_CASetScanType Function
WFCAElementsStruct Structure
WF_CAGetDtimInterval Function
WF_CASetDtimInterval Function
WF_CAGetBeaconPeriod Function
WF_CASetBeaconPeriod Function
WF_CASetBeaconPeriod Function

Connection Algorithm Internal Members
LowLevel_CAGetElement Function
LowLevel_CASetElement Function
SetEventNotificationMask Function

Wi-Fi Connection Manager

Connection Manager Public Members
WF_CMConnect Function
WF_CMDisconnect Function
WF_CMGetConnectionState Function
WF_CMGetConnectContext Function
WF_CMCheckConnectionState Function
WF_DisableModuleConnectionManager Function

Wi-Fi Scan
Scan Operation and Scan Results
Shorter Scan or Connection Duration
Use of macro #define MY_DEFAULT_CHANNEL_LIST
Maximum Scan Results

Scan Public Members
WF_Scan Function
WF_ScanGetResult Function
tWFScanResult Structure

Wi-Fi Security

Wired Equivalent Privacy (WEP)
WF_CPGetWepKeyType Function
WF_CPGetDefaultWepKeyIndex Function
WF_CPSetDefaultWepKeyIndex Function
WF_CPSetWepKeyType Function

**Wi-Fi Protected Access (WPA/WPA2)**
WF_CPUpdatePMK Function
WF_ConvPassphrase2Key Function
pbkdf2_sha1 Function

**Wi-Fi Protected Setup (WPS)**
WF_CPGetWPSCredentials Function
WF_YieldPassphrase2Host Function
WF_SetPSK Function
WF_SaveWPSCredentials Function
ConfigWep Function
tWFWpsCred Structure

WPA2 Enterprise

**Wi-Fi Tx Power Control**

**Tx Power Control Public Members**
WF_TxPowerGetMinMax Function
WF_TxPowerSetMinMax Function
WF_TxPowerGetFactoryMax Function
WF_TxPowerGetMax Function
WF_TxPowerSetMax Function
WF_FixTxRateWithMaxPower Function

**Wi-Fi Power Save**

**Power Save Public Members**
WF_GetPowerSaveState Function
WF_HibernateEnable Function
WF_PsPollDisable Function
WF_PsPollEnable Function
CheckHibernate Function
WFHibernate Structure

**Power Save Internal Members**
SendPowerModeMsg Function
SetPowerSaveState Function

**Functions**
GetAppPowerSaveMode Function
SetAppPowerSaveMode Function

**Types**
tWFPsPwrMode Type
tWFPwrModeReq Type

**Variables**
g_AppPowerSaveModeEnabled Variable
g_powerSaveState Variable
g_psPollActive Variable
g_sleepNeeded Variable

Wi-Fi Miscellaneous

Wi-Fi Miscellaneous Public Members
- WF_GetDeviceInfo Function
- WF_GetMacAddress Function
- WF_GetMacStats Function
- WF_GetMultiCastFilter Function
- WF_GetRegionalDomain Function
- WF_GetRtsThreshold Function
- WF_SetMacAddress Function
- WF_SetMultiCastFilter Function
- WF_SetRegionalDomain Function
- WF_SetRtsThreshold Function
- WF_EnableSWMultiCastFilter Function
- WF_MulticastSetConfig Function
- WF_SetLinkDownThreshold Function
- WF_GetTxMode Function
- WF_SetTxMode Function
- WFMacStatsStruct Structure
- WFMulticastConfigStruct Structure
- tWFDeviceInfoStruct Structure

WF_ProcessEvent

Access Point Compatibility

802.11 AP/Router Configuration Settings

WiFi Troubleshooting Tips
- Null String ESSID
- Read back RF module Firmware version
- RF Module Firmware Update
- Wi-Fi Protected Setup (WPS) Issues
- Network Switch or Change
- Hibernate Mode
- Management Scan Message Conflict
- Handling of maximum length SSID
- Multicast Filters : Hardware vs Software
- MRF24WB0M assert failures when using <iwconfig scan> command
- MRF24WB0M advertised supported rates of 1, 2, 5.5 and 11 Mbps
- MRF24WB0M Compatibility with AP/Routers
- Encounter issues after upgrading MRF24WB0M RF module Firmware version 0x1207
- How to fix MRF24WB0M / MRF24WG0M transmission rates
- How to determine new IP address assigned
- How to increase TCP throughput
- Missing DHCP Client Name
Error Scenario And Possible Causes

Wireless Packets Analysis
## Index

These are all topics and symbols available in this documentation.

### Alphabetical Index

- _attribute_ function
- _checkIpSrvrResponse_ variable
- _IsSNMPv3ValidStructure_ function
- _LoadFATRecord_ function
- _MD5_k_ variable
- _MD5_r_ variable
- _NBNS_HEADER_ structure
- _SNMPDuplexInit_ function
- _SNMPGet_ function
- _SNMPGetTxOffset_ macro
- _SNMPPut_ function
- _SNMPSetTxOffset_ macro
- _Snmpv3IsValidAuthStructure_ function
- _Snmpv3IsValidInt_ function
- _TFTP_ACCESS_ERROR_ enumeration
- _TFTP_FILE_MODE_ enumeration
- _TFTP_RESULT_ enumeration
- _tftpError_ variable
- _tftpFlags_ variable
- _tftpRetries_ variable
- _TFTPSendAck_ function
- _TFTPSendTimeName function
- _TFTPSendROMFileName function
- _tftpSocket_ variable
- _tftpStartTick_ variable
- _tftpState_ variable
- _updateIpSrvrResponse_ variable

### Other Topics

- SNMP Server (Agent)
- Variables
- SNMP Stack Members
- SNMP Traps
- SNMP.c
- SNMP.h
- SNMP_ACTION
- SNMP_AGENT
- SNMP_BIB_FILE
- SNMP_COMMUNITY
- SNMP_COUNT
- SNMP_END_OF
- SNMP_ENGINE
Validate function

802.11 AP/Router Configuration Settings

A

accept function
Access Point Compatibility
AccessCtrlSubSysIsAccessAllowed structure
Accessing the Demo Application
ACK macro
activeUDPSocket variable
Additional Features
Address
Ad-hoc Network
Advanced MPFS2 Settings
AF_INET macro
AGENT_NOTIFY_PORT macro
Announce
Announce Stack Members
AnnounceIP function
APP_CONFIG Structure
appendZeroToOID variable
ARP
ARP Internal Members
ARP Public Members
ARP Stack Members
arp_app_callbacks structure
ARP_IP macro
ARP_OPERATION_REQ macro
ARP_OPERATION_RESP macro
ARP_REQ macro
ARP_RESP macro
ARPDeRegisterCallbacks function
ARPInit function
ARPIsResolved function
ARPProcess function
ARPPut function
ARPRegisterCallbacks function
ARPResolve function
ARPSendPkt function
ASN_INT macro
ASN_NULL macro
ASN_OID macro
AUTH_LOCALIZED_PASSWORD_KEY_LEN macro
Authentication
authKey_iPad variable
authKey_oPad variable
authoritativeSnmpEngineBoots variable
authoritativeSnmpEngineTime variable
Available Demos

Base64Decode function
Base64Encode function
Berkeley (BSD) Sockets
BerkeleySocketInit function
bForceUpdate variable
bind function
Bootloader Design
BSD Sockets
BSD Wrapper Internal Members
BSD Wrapper Public Members
BSD Wrapper Stack Members
BSD_SCK_STATE enumeration
BSDSocket structure
BSDSocketArray variable

SNMPGetTimeSt
SNMPGetTrapTir
SNMPGetVar fun
SNMPIdRecrdrVa
SNMPIsNotifyRe
SNMPIsValidSetLen
SNMPNONMIBR
SNMPNotify func
SNMPNotifyInfo
SNMPNotifyPrep
SNMPTrapTimer v
SNMPv3.c
SNMPv3.h

snmpInMsgAuthF
snmpInMsgAuthF
snmpInMsgPrivP
snmpInMsgPrivP
snmpOutMsgAuthl
snmpOutMsgAuthl
snmpOutMsgPriv
snmpOutMsgPrivl
snmpResponseS
snmpResponseS
snmpSecurityLev
snmpSecurityLev
SNMPSendTrap
SNMPSendTrap
SNMPStatus vari
SNMPTask functi
snmpTrapTimer v
snmpTrapTimer v
SNMPTxOffset v;
btohexa_high function
btohexa_low function
Building MPFS2 Images

C
Cache variable
CalcIPChecksum function
CalculateFinishedHash function
CheckHibernate function
cipher_text variable
Clock Frequency
closesocket function
CloseSocket function
COMMUNITY_TYPE enumeration
Configure your WiFi Access Point
Configuring the Stack
Configuring WiFi Security
ConfigWep function
connect function
Connecting to the Network
Connection Algorithm Internal Members
Connection Algorithm Public Members
Connection Manager Public Members
Connection Profile Internal Members
Connection Profile Public Members
Cookies
Cooperative Multitasking
CRPeriod variable
curHTTP variable
curHTTPID variable

D
DATA_TYPE enumeration
DATA_TYPE_INFO structure

SNMPV3_H macro
SNMPV3_HMAC
SNMPV3_MSG_*
SNMPV3_MSG_*
SNMPV3_PRIV_*
macro
SNMPV3_PRIV_*
SNMPV3_REQ_*
SNMPV3_RESP_*
SNMPV3_USM_*

snmpV3AesDecryptInitVector
Snmpv3AESDecrypt
Snmpv3AesEncryptInitVector
Snmpv3AesEncrypt
Snmpv3AuthenticateRxedPduForDataIntegrity
Snmpv3AuthenticateTxPduForDataIntegrity
Snmpv3AuthKeyZeroing2HmacBufLen64
Snmpv3BufferPut
Snmpv3CmprTrapSecNameAndSecLvlWithUSMDb
Snmpv3ComputeHMACIpadOpadForAuthLoclzedKey
Snmpv3ComputeHmacMD5Digest
Snmpv3ComputeHmacShaDigest
Snmpv3ComputeMd5HmacCode
Snmpv3ComputeShaHmacCode
Snmpv3EngnUserDataBase
Snmpv3FreeDynMem
Snmpv3GetAuthEngineTime
Snmpv3GetBufferData
Snmpv3GetSecurityLevel
Snmpv3GetTrapSecurityLevel
Snmpv3Init
Snmpv3Invoke
Snmpv3InitializeUserDataBase
DNS_TIMEOUT macro
DNS_TYPE_A macro
DNS_TYPE_MX macro
DNSBeginUsage function
DNSDiscardName function
DNSEndUsage function
DNSHostName variable
DNSHostNameROM variable
DNSIsResolved function
DNSPutROMString function
DNSPutString function
DNSResolve function
DNSResolveROM function
dwInternalTicks variable
dwLastUpdateTick variable
dwLFSRRandSeed variable
dwSNTPSeconds variable
dwUpdateAt variable
Dynamic DNS Client
Dynamic DNS Internal Members
Dynamic DNS Public Members
Dynamic DNS Stack Members
Dynamic Variables

SOCKADDR type
sockaddr_in structure
SOCKADDR_IN type
socket function
SOCKET type
Socket Types
SOCKET_CNXN
SOCKET_DISC
SOCKET_ERROR
SOCKET_INFO
Sockets
SocketWithRxDa
SoftAP Network
Software
SSL
Files
SSL_Internal_Mem
SSL_Public_Mem
SSL_Stack_Memb
SSL_ALERT macro
SSL_ALERT_LEVEL enumeration
SSL_APPLICATION macro
SSL_BASE_BUFF
SSL_BASE_HAS
SSL_BASE_KEY
SSL_BASE_SES
SSL_BASE_STUB
SSL_BUFFER-uri
SSL_BUFFER_S
SSL_CERT variable
SSL_CERT_LEN
SSL_CHANGE_CIPHER_SPEC
SSL_HANDSHA
ExtractURLFields function

fatCache variable
fatCacheID variable
FIN macro
FindEmailAddress function
FindMatchingSocket function
FindOIDsFromSnmpV3Request function
FindOIDsInRequest function
FindROMEmailAddress function
Flag_ImageUpdate_running variable
Flags variable
FormatNetBIOSName function
Forms using GET
Forms using POST

g_AppPowerSaveModeEnabled variable
g_powerSaveState variable
g_psPollActive variable
g_sleepNeeded variable
gAutoPortNumber variable
GenerateHashRounds function
GenerateRandomDWORD function
GenerateSessionKeys function
Generating Server Certificates
Generic TCP Client
  Variables
Generic TCP Server
  Macros
  GENERIC_TRAP_NOTIFICATION_TYPE enumeration
GenericTCPClient function

SSL_HASH_SIZE
SSL_HASH_SPACE
SSL_INVALID_ID
SSL_KEYS structure
SSL_KEYS_SIZE
SSL_KEYS_SPACE
SSL_MESSAGE
SSL_MIN_SESSION
SSL_PKEY_INFO
SSL_RSA_CLIENT
SSL_RSA_EXPORT_WITH_ARCFOUR_40_MD5
SSL_RSA_LIFETIME
SSL_RSA_LIFETIME_EXTENSION
SSL_RSA_SESSION
SSL_RSA_WITH
SSL_SESSION
SSL_SESSION_SIZE
SSL_SESSION_SPACE
SSL_SESSION_STUB
SSL_SESSION_TYPE enumeration
SSL_STATE enumeration
SSL_STUB structure
SSL_STUB_SIZE
SSL_STUB_SPACE
SSL_SUPPLEMENTARY_DATA_TYPES enumeration
SSL_VERSION
SSL_VERSION_HI
SSL_VERSION_LO
SSLBufferAlloc function
SSLBufferFree function
sslBufferID variable
SSLBufferSync function
SSLClientSize.h
SSLFinishPartialRecord
SSLFlushPartialRecord
GenericTCP server function

GET_BULK_REQUEST macro

GET_NEXT_REQUEST macro

GET_REQUEST macro

GET_RESPONSE macro

GetAppPowerSaveMode function

GetDataTypeInfo function

gethostname function

GetNextLeaf function

GetOIDStringByAddr function

GetOIDStringByID function

getSnmpV2GenTrapOid function

GetTickCopy function

Getting Help

Getting Started

getZeroInstance variable

gGenericTrapNotification variable

gOIDCorrespondingSnmpMibID variable

Google PowerMeter

gSendTrapFlag variable

gSendTrapSMstate variable

gSetTrapSendFlag variable

gSnmpNonMibRecInfo variable

gSnmpV3InPduWholeMsgBuf variable

gSnmpV3OUTPduWholeMsgBuf variable

gSNMPv3PduHeaderBuf variable

gSNMPv3ScopedPduDataPos variable

gSNMPv3ScopedPduRequestBuf variable

gSNMPv3ScopedPduResponseBuf variable

gSnmpv3TrapConfigData variable

gSNMPv3TrapMsgHeaderBuf variable

gSNMPV3TrapSecurityLevel variable

sslHash variable

SSLHashAlloc function

SSLHashFree function

sslHashID variable

SSLHashSync function

sslKeys variable

sslKeysID variable

SSLKeysSync function

SSLPeriodic function

SSLRSASecurity function

SSLRxAlert function

SSLRxAntiqueClientHello function

SSLRxCCS function

SSLRxClientHello function

SSLRxClientKeyExchange function

SSLRxFinished function

SSLRxHandshake function

SSLRxRecord function

SSLRxServerCertificate function

SSLRxServerHello function

sslSession variable

sslSessionID variable

sslSessionSync function

sslSessionUpdated variable
How to determine new IP address assigned
How to fix MRF24WB0M / MRF24WG0M transmission rates
How to increase TCP throughput
HSEnd function
HSGet function
HSGetArray function
HSGetWord function
HSPut function
HSPutArray function
HSPutROMArray function
HSPutWord function
HSStrart function
HTTP Configuration
HTTP_CACHE_LEN macro
HTTP_CONN structure
HTTP_FILE_TYPE enumeration
HTTP_IO_RESULT enumeration
HTTP_MAX_DATA_LEN macro
HTTP_MAX_HEADER_LEN macro
HTTP_MIN_CALLBACK_FREE macro
HTTP_PORT macro
HTTP_READ_STATUS enumeration
HTTP_STATUS enumeration
HTTP_STUB structure
HTTP_TIMEOUT macro
HTTP2 Authentication
HTTP2 Compression
HTTP2 Cookies
HTTP2 Dynamic Variables
HTTP2 Features
HTTP2 Form Processing
HTTP2 Internal Members
HTTP2 Public Members
strncpy_m function
strPassword variable
strSpaces variable
strTitle variable
STRUCTURE macro
strupr function
SW License Agreement
SwapARPPacket
swapl function
swaps function
SwapTCPHeader
SYN macro
SyncTCB function
SyncTCBStub macro
SYNQueue variable
TCB structure
TCB_STUB struct
TCBStubs variable
TCP Functions
TCP Internal Members
TCP Public Members
TCP Stack Members
TCP/IP Configuration
TCP_ADJUST_GIVE_REST_TO_RX macro
TCP_ADJUST_GIVE_REST_TO_TX macro
TCP_ADJUST_PRESERVE_RX macro
TCP_ADJUST_PRESERVE_TX macro
TCP_AUTO_TRANSMIT_TIMEOUT_VAL macro
TCP_CLOSE_WAIT_TIMEOUT macro
TCP_DELAYED_ACK_TIMEOUT macro
TCP_FIN_WAIT_2_TIMEOUT macro
HTTPSendFile function
HTTPServer function
httpStubs variable
HTTPURLDecode function
HW_ETHERNET macro

ICMP
ICMP Internal Members
ICMP Public Members
ICMP_PACKET structure
ICMP_TIMEOUT macro
ICMPBeginUsage function
ICMPEndUsage function
ICMPFlags variable
ICMPPingReply function
ICMPProcess function
ICMPSendPing function
ICMPSendPingToHost function
ICMPSendPingToHostROM function
ICMPSendPingToHostROM macro
ICMPState variable
ICMPTimer variable
ifconfig Commands
ImageUpdate_Addr variable
ImageUpdate_Checksum variable
ImageUpdate_Size variable
in_addr structure
INADDR_ANY macro
incomingPdu variable
incomingSnmpPDUmsgID variable
INDEX_INFO union
Infrastructure Network
Initialization

TCPFindEx function
TCPFindROMArray
TCPFindROMArrayEx
TCPFlush function
TCPGet function
TCPGetArray function
TCPGetRemote
TCPGetRxFIFO
TCPGetRxFIFOEx
TCPGetTxFIFO
TCPGetTxFIFOEx
TCPInit function
TCPIP Demo App
TCPIsConnected
TCPIsGetReady
TCPIsPutReady
TCPListen function
TCPOpen function
TCPPeek function
TCPPeekArray function
TCPPerformance
TCPPut function
TCPPutArray function
TCPPutROMArray
TCPPutROMString
TCPPutString function
TCPRAMCopy function
TCPRAMCopyROM
TCPRequestSSL
TCPRXPerformance
TCPSSLDecrypt
TCPSSSLGetPenc
Initialization Structure
INOUT_SNMP_PDU enumeration
Internet Bootloader
Internet Radio
Introduction
INVALID_INDEX macro
INVALID_SOCKET macro
INVALID_TCP_PORT macro
INVALID_UDP_PORT macro
INVALID_UDP_SOCKET macro
IP Address
IP_ADDR_ANY macro
iperf Example
IPPROTO_IP macro
IPPROTO_TCP macro
IPPROTO_UDP macro
IS_AGENT_PDU macro
IS ASN_INT macro
IS ASN_NULL macro
IS_GET_NEXT_REQUEST macro
IS_GET_REQUEST macro
IS_GET_RESPONSE macro
IS OCTET_STRING macro
IS_OID macro
IS SET_REQUEST macro
IS SNMPV3_AUTH_STRUCTURE macro
IS_STRUCTURE macro
IS TRAP macro
IsASNNull function
isBufferUsed variable
isHashUsed variable
isMPFSLocked variable
IsSnmpV3ASNNull function
IsSnmpv3ValidOID function
TCPSSLHandleIncoming
TCPSSLHandshake
TCPSSLInPlaceMACEncrypt
TCPStartSSLClient
TCPStartSSLClientEx
TCPStartSSLServer
TCPTick
TCPTXPerformance
TCPWasReset
Telnet
Telnet Internal Members
Telnet Public Members
Telnet Stack Members
TELNET_PASSWORD macro
TELNET_PORT macro
TELNETS_PORT macro
TELNET_USERNAME macro
TFTP
TFTP ACCESS
TFTP ARP_TIMEOUT
TFTP ARP_TIMEOUT VAL
TFTP BLOCK_SIZE
TFTP BLOCK_SIZE_MSB
TFTP CLIENT_PORT macro
TFTP_END_OF_FILE
TFTP_ERROR
TFTP_ERROR_ACCESS_VIOLATION
TFTP_ERROR_DISK_FULL
TFTP_ERROR__
TFTP_ERROR_ACCESS_VIOLATION
TFTP_ERROR_DISK_FULL
TFTP_ERROR__
TFTP_ERROR_ACCESS_VIOLATION
TFTP_ERROR_DISK_FULL
TFTP_ERROR__
TFTP_ERROR_ACCESS_VIOLATION
TFTP_ERROR_DISK_FULL
TFTP_ERROR__
TFTP_ERROR_ACCESS_VIOLATION
TFTP_ERROR_DISK_FULL
TFTP_ERROR__
MAC Address
Main File
Main Loop
Management Scan Message Conflict
masks variable
MAX_FILE_NAME_LEN macro
MAX_REG_APPS macro
MAX_TELNET_CONNECTIONS macro
MAX_TRY_TO_SEND_TRAP macro
Maximum Scan Results
MD5AddData function
MD5AddROMData function
MD5Calculate function
MD5HashBlock function
MD5Initialize function
md5LocalizedAuthKey variable
Memory Allocation
Memory Usage
MIB Browsers
MIB Files
MIB_INFO union
Microchip TCP/IP Discoverer
Missing DHCP Client Name
MPFS_FAT_RECORD structure
MPFS_HANDLE type
MPFS_INVALID macro
MPFS_INVALID_FAT macro
MPFS_INVALID_HANDLE macro
MPFS_PTR type
MPFS_SEEK_MODE enumeration
MPFS_STUB structure
MPFS_WRITE_PAGE_SIZE macro
TFTPIsFileClose
TFTPIsFileOpen
TFTPIsFileOpenFile
TFTPIsGetReady
TFTPIsOpenedFile
TFTPIsPutReady
TFTPOpen function
TFTPOpenFileFunction
TFTPOpenROMFunction
TFTPPut function
TFTPUploadFragmentedRAMFileToHost
TFTPUploadRAMFileToHost
Tick Internal Members
Tick Module
Functions
Tick Public Members
Tick Stack Functions
TICK variable
TICK_HOUR macro
TICK_MINUTE macro
TICK_SECOND macro
TickConvertToMilliseconds
TickGet function
TickGetDiv256Function
TickGetDiv64KFunction
TickInit function
TICKS_PER_SECOND macro
TickUpdate function
TransportState variable
TRAP macro
TRAP_COMMUNITY_MAX_LEN macro
TRAP_INFO structure
TRAP_TABLE_SIZE macro
trapInfo variable
MPFS2
MPFS2 Command Line Options
MPFS2 Internal Members
MPFS2 Public Members
MPFS2 Stack Members
MPFS2 Utility
MPFS2_FLAG_HASINDEX macro
MPFS2_FLAG_ISZIPPED macro
MPFSClose function
MPFSFormat function
MPFSGet function
MPFSGetArray function
MPFSGetBytesRem function
MPFSGetEndAddr function
MPFSGetFilename function
MPFSGetFlags function
MPFSGetID function
MPFSGetLong function
MPFSGetMicrotime function
MPFSGetPosition function
MPFSGetSize function
MPFSGetStartAddr function
MPFSGetTimestamp function
MPFSInit function
MPFSOpen function
MPFSOpenID function
MPFSOpenROM function
MPFSPutArray function
MPFSPutEnd function
MPFSSeek function
MPFSStubs variable
MPFSTell macro
MRF24WB0M advertised supported rates of 1, 2, 5.5 and 11 Mbps

UDP
UART-to-TCP Bridge
UDP Types
UDP Internal Members
UDP Public Members
UDP Sockets
UDP Stack Members
UDP_HEADER_enum
UDP_OPEN_IP_interface
UDP_OPEN_NO_connect
UDP_OPEN_READ
UDP_OPEN_RDWR
UDP_OPEN_SEI
UDP_PORT_type
UDP_SOCKET_type
UDP_STATE_enum
UDPClose function
UDPDiscard function
UDPFlush function
UDPGet function
UDPGetArray function
UDPInit function
UDPIsGetReady function
UDPIsOpened function
UDPSocktype
UDPSocket_t
U
MRF24WB0M assert failures when using `<iwconfig scan>` command
MRF24WB0M Compatibility with AP/Routers
MSG_AUTHORITATIVE_HEADER_LEN macro
MSGGLOBAL_HEADER_LEN macro
MsgProcModPrepareDataElements structure
MsgProcModPrepareOutgoingMessage structure
MsgProcModPrepareResponseMessage structure
msgSecrtyParamLenOffSet variable
Multicast Filters: Hardware vs Software
MutExVar variable
MySocket variable
MyTCB variable
MyTCBStub variable

NBNS
NBNS Stack Members
NBNS_HEADER structure
NBNS_PORT macro
NBNSGetName function
NBNSPutName function
NBNSTask function
Network Switch or Change
NOTIFY_COMMUNITY_LEN macro
NTP_EPOCH macro
NTP_FAST_QUERY_INTERVAL macro
NTP_PACKET structure
NTP_QUERY_INTERVAL macro
NTP_REPLY_TIMEOUT macro
NTP_SERVER macro
NTP_SERVER_PORT macro
Null String ESSID
numFiles variable

UDPIsPutReady function
UDPIsPutReady macro
UDPOpen macro
UDPOpenEx function
UDPPacket function
UDPPut function
UDPPutArray function
UDPPutROMArray function
UDPPutROMStr function
UDPPutString function
UDPRxCount variable
UDPSetRxBuffer function
UDPSetTxBuffer function
UDPSocketInfo variable
UDPTask function
UDPTxCount variable
uitoa function
ultoa function
UnencodeURL function
UNKNOWN_SOCKET macro
unregisterContextEngineID function
uploadChunkDescriptor function
uploadChunkDescriptorForRetransmit function
Uploading Pre-built MPFS2 Images
Uploading Web Pages
URG macro
Use of macro #define
USER_SECURITY_NAME_LEN macro
Using the Bootloader
Using the Stack
USM_SECURITY_LEVEL enumeration
VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE enumeration
<table>
<thead>
<tr>
<th>O</th>
<th>vUploadFilename</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCTET_STRING macro</td>
<td>vUploadRemoteFile</td>
</tr>
<tr>
<td>OID_INFO structure</td>
<td></td>
</tr>
<tr>
<td>OID_MAX_LEN macro</td>
<td></td>
</tr>
<tr>
<td>OIDLookup function</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W</th>
<th>Web Page Demo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td></td>
</tr>
<tr>
<td>Variables</td>
<td></td>
</tr>
<tr>
<td>WebVend</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pbkdf2_sha1 function</td>
<td></td>
</tr>
<tr>
<td>PDU_INFO structure</td>
<td></td>
</tr>
<tr>
<td>Performance Test Internal Members</td>
<td></td>
</tr>
<tr>
<td>Performance Test Stack Members</td>
<td></td>
</tr>
<tr>
<td>Performance Tests</td>
<td></td>
</tr>
<tr>
<td>PERFORMANCE_PORT macro</td>
<td></td>
</tr>
<tr>
<td>Peripheral Usage</td>
<td></td>
</tr>
<tr>
<td>PIC18 Explorer</td>
<td></td>
</tr>
<tr>
<td>PIC18F97J60 Config</td>
<td></td>
</tr>
<tr>
<td>PIC24FJ256DA210 Dev Board</td>
<td></td>
</tr>
<tr>
<td>PIC32MX7XX Config</td>
<td></td>
</tr>
<tr>
<td>PICDEM.net 2</td>
<td></td>
</tr>
<tr>
<td>Ping (ICMP) Demo</td>
<td></td>
</tr>
<tr>
<td>Macros</td>
<td></td>
</tr>
<tr>
<td>PingDemo function</td>
<td></td>
</tr>
<tr>
<td>Power Save Internal Members</td>
<td></td>
</tr>
<tr>
<td>Power Save Public Members</td>
<td></td>
</tr>
<tr>
<td>PRIV_LOCALIZED_PASSWORD_KEY_LEN macro</td>
<td></td>
</tr>
<tr>
<td>ProcessGetBulkVar function</td>
<td></td>
</tr>
<tr>
<td>ProcessGetNextVar function</td>
<td></td>
</tr>
<tr>
<td>ProcessGetSetHeader function</td>
<td></td>
</tr>
<tr>
<td>ProcessGetVar function</td>
<td></td>
</tr>
<tr>
<td>ProcessHeader function</td>
<td></td>
</tr>
<tr>
<td>processResponsePdu structure</td>
<td></td>
</tr>
<tr>
<td>ProcessSetVar function</td>
<td></td>
</tr>
<tr>
<td>ProcessSnmpv3MsgData function</td>
<td></td>
</tr>
<tr>
<td>vUploadRemoteFile</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W</th>
<th>Web Page Demo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td></td>
</tr>
<tr>
<td>Variables</td>
<td></td>
</tr>
<tr>
<td>WebVend</td>
<td></td>
</tr>
</tbody>
</table>
RF Module Firmware Update
ROMStringToIPAddress function
ROMStringToIPAddress macro
RST macro
RTOS
RX_PERFORMANCE_PORT macro
RXParserState variable

SaveOffChip function
Scan Operation and Scan Results
Scan Public Members
securityPrimitivesOfIncomingPdu variable
SecuritySysGenerateRequestMsg structure
SecuritySysGenerateResponseMsg structure
SecuritySysProcessIncomingMsg structure
send function
SendNotification function
SendPowerModeMsg function
SendTCP function
SENDTCP_KEEP_ALIVE macro
SENDTCP_RESET_TIMERS macro
sendto function
SERVER_PORT macro
ServerName variable
ServerPort variable
session_key variable
SET_REQUEST macro
SetAppPowerSaveMode function
SetErrorStatus function
SetEventNotificationMask function
SetPowerSaveState function
SHA1AddData function
SHA1AddROMData function

WF_CPSetSsidType function
WF_CPSetWepKeyType function
WF_CPUpdatePI function
WF_DisableModule function
WF_EnableSWM function
WF_FixTxRateWithMaxPower function
WF_GetDeviceIn function
WF_GetMacAddress function
WF_GetMacStats function
WF_GetMultiCast function
WF_GetPowerState function
WF_GetRtsThreshold function
WF_HibernateEnable function
WF_MulticastSet function
WF_ProcessEvent function
WF_PsPollDisable function
WF_PsPollEnable function
WF_SaveWPSCredentials function
WF_Scan function
WF_ScanGetResult function
WF_SetLinkDownThreshold function
WF_SetMacAddress function
WF_SetMultiCast function
WF_SetPSK function
WF_SetRegional function
WF_SetRtsThreshold function
WF_TxPowerGet function
WF_TxPowerGet function
WF_TxPowerGet function
WF_TxPowerGet function
WF_TxPowerSet function
WF_TxPowerSet function
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 function to include the functionality you'd like. You may insert or change states in TelnetTask as needed.

### Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telnet Public Members</td>
<td>Functions and variables accessible by the stack application</td>
</tr>
<tr>
<td>Telnet Stack Members</td>
<td>Functions and variables intended to be accessed only by the stack</td>
</tr>
<tr>
<td>Telnet Internal Members</td>
<td>Functions and variables internal to the Telnet module</td>
</tr>
</tbody>
</table>

Stack API > Telnet
# Web Page Demos Functions

## Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPPostImage</td>
<td>This is function HTTPPostImage.</td>
</tr>
<tr>
<td>HTTPPostConfig</td>
<td>Processes the configuration form on config/index.htm</td>
</tr>
<tr>
<td>HTTPPostSNMPCommunity</td>
<td>This is function HTTPPostSNMPCommunity.</td>
</tr>
<tr>
<td>HTTPPostDDNSConfig</td>
<td>Parsing and collecting http data received from http form.</td>
</tr>
<tr>
<td>HTTPPostEmail</td>
<td>Processes the e-mail form on email/index.htm</td>
</tr>
<tr>
<td>HTTPPostLCD</td>
<td>Processes the LCD form on forms.htm</td>
</tr>
<tr>
<td>HTTPPostMD5</td>
<td>Processes the file upload form on upload.htm</td>
</tr>
</tbody>
</table>

Demo Information > Available Demos > Demo App > Demo Modules > Web Page Demos
## Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag_ImageUpdate_running</td>
<td>This is variable Flag_ImageUpdate_running.</td>
</tr>
<tr>
<td>ImageUpdate_Addr</td>
<td>Processes the file upload form on upload_image.htm</td>
</tr>
<tr>
<td>DDNSData</td>
<td>RAM allocated for DDNS parameters</td>
</tr>
<tr>
<td>ImageUpdate_Checksum</td>
<td>This is variable ImageUpdate_Checksum.</td>
</tr>
<tr>
<td>ImageUpdate_Size</td>
<td>This is variable ImageUpdate_Size.</td>
</tr>
<tr>
<td>lastFailure</td>
<td>Stick status message variable. See lastSuccess for details.</td>
</tr>
<tr>
<td>lastSuccess</td>
<td>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.</td>
</tr>
</tbody>
</table>
## Generic TCP Client Variables

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoteURL</td>
<td>Defines the URL to be requested by this HTTP client</td>
</tr>
<tr>
<td>ServerName</td>
<td>Defines the server to be accessed for this application</td>
</tr>
<tr>
<td>ServerPort</td>
<td>Note that if HTTPS is used, the ServerName and URL must change to an SSL enabled server.</td>
</tr>
</tbody>
</table>

-Demo Information > Available Demos > Demo App > Demo Modules > Generic TCP Client

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
## Generic TCP Server Macros

### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVER_PORT</td>
<td>Defines which port the server will <em>listen</em> on</td>
</tr>
</tbody>
</table>

[Demo Information > Available Demos > Demo App > Demo Modules > Generic TCP Server](#)
## Ping (ICMP) Demo Macros

### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST_TO_PING</td>
<td>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.</td>
</tr>
</tbody>
</table>

[Demo Information] > [Available Demos] > [Demo App] > [Demo Modules] > [Ping (ICMP) Demo]
## SNMP Server (Agent) Functions

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendNotification</td>
<td>Prepare, validate remote node which will receive trap and send trap pdu.</td>
</tr>
<tr>
<td>SNMPGetTimeStamp</td>
<td>Obtains the current Tick value for the SNMP time stamp.</td>
</tr>
</tbody>
</table>

[Demo Information > Available Demos > Demo App > Demo Modules > SNMP Server (Agent)]

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
### SNMP Server (Agent) Macros

#### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_TRY_TO_SEND_TRAP</td>
<td>Update the Non record id OID value which is part of CustomSnmpDemo.c file</td>
</tr>
<tr>
<td>SNMP_MAX_NON_REC_ID_OID</td>
<td>Default STACK_USE_SMIV2 is enabled. For Stack V5.31, STACK_USE_SMIV2 should be disabled.</td>
</tr>
<tr>
<td>STACK_USE_SMIV2</td>
<td></td>
</tr>
</tbody>
</table>
## SNMP Server (Agent) Variables

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gSendTrapSMstate</td>
<td>This is variable gSendTrapSMstate.</td>
</tr>
<tr>
<td>gSnmpNonMibRecInfo</td>
<td>OLD snmp.mib file with SMIv1 standard</td>
</tr>
<tr>
<td>gSnmpv3UserSecurityName</td>
<td>This is variable gSnmpv3UserSecurityName.</td>
</tr>
<tr>
<td>gtrapSMStateUpdate</td>
<td>This is variable gtrapSMStateUpdate.</td>
</tr>
</tbody>
</table>

[Demo Information] > [Available Demos] > [Demo App] > [Demo Modules] > [SNMP Server (Agent)]
# Helpers Functions

## Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFSRRand</td>
<td>Returns a pseudo-random 16-bit unsigned integer in the range from 0 to 65535 (0x0000 to 0xFFFF).</td>
</tr>
<tr>
<td>LFSRSeedRand</td>
<td>Seeds the LFSR random number generator invoked by the LFSRRand() function. The prior seed is returned.</td>
</tr>
<tr>
<td>strncpy_m</td>
<td>Copies multiple strings to a destination</td>
</tr>
</tbody>
</table>

[Stack API > Helpers](#)
### Helpers Variables

#### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwLFSRRandSeed</td>
<td>Default Random Number Generator seed. 0x41FE9F9E corresponds to calling LFSRSeedRand(1)</td>
</tr>
</tbody>
</table>

Stack API > Helpers

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
## SNMP Files

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SNMPv3.c</strong></td>
<td>- Simple Network Management Protocol (SNMP) Version 3 Agent</td>
</tr>
<tr>
<td></td>
<td>- Module for Microchip TCP/IP Stack</td>
</tr>
<tr>
<td></td>
<td>- Provides SNMPv3 API for doing stuff</td>
</tr>
<tr>
<td></td>
<td>- Reference: RFCs 3410, 3411, 3412, 3413, 3414</td>
</tr>
<tr>
<td><strong>SNMPv3.h</strong></td>
<td>- Simple Network Management Protocol (SNMP) Version 3 Agent</td>
</tr>
<tr>
<td></td>
<td>- Module for Microchip TCP/IP Stack</td>
</tr>
<tr>
<td></td>
<td>- Provides SNMPv3 API for doing stuff</td>
</tr>
<tr>
<td></td>
<td>- Reference: RFCs 3410, 3411, 3412, 3413, 3414</td>
</tr>
<tr>
<td><strong>SNMPv3USM.c</strong></td>
<td>- Simple Network Management Protocol (SNMP) Version 3 Agent</td>
</tr>
<tr>
<td></td>
<td>- Module for Microchip TCP/IP Stack</td>
</tr>
<tr>
<td></td>
<td>- Provides SNMPv3 API for doing stuff</td>
</tr>
</tbody>
</table>
| SNMP.c | - Simple Network Management Protocol (SNMP) Version 1 Agent  
|        | - Simple Network Management Protocol (SNMP) Version 2 community based Agent  
|        | - Module for Microchip TCP/IP Stack  
|        | - Provides SNMP API for doing stuff  
|        | - Reference: RFC 1157 (for SNMP V1)  
|        | - RFC 3416 (for SNMPv2C)  |

| SNMP.h | - SNMP Defs for Microchip TCP/IP Stack  |

- Reference: RFCs 3410, 3411, 3412, 3413, 3414
## SNMP Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IsSNMPv3ValidStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>_Snmpv3IsValidAuthStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>_Snmpv3IsValidInt</td>
<td>Verifies variable datatype as INT and retrieves its value.</td>
</tr>
<tr>
<td>FindOIDSFromSnmpV3Request</td>
<td>Finds number of varbinds in the varbind list received in a SNMPv3 pdu.</td>
</tr>
<tr>
<td>getSnmpV2GenTrapOid</td>
<td>Resolves generic trap code to generic trap OID.</td>
</tr>
<tr>
<td>IsSnmpV3ASNNull</td>
<td>Verifies the value type as ASN_NULL.</td>
</tr>
<tr>
<td>IsSnmpv3ValidOID</td>
<td>Populates OID type, length and oid string from the received pdu.</td>
</tr>
<tr>
<td>IsSNMPv3ValidStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>ProcessGetBulkVar</td>
<td>This routine process the SNMPv2c Get Bulk Request.</td>
</tr>
<tr>
<td>ProcessGetNextVar</td>
<td>Retrieves next node from the MIB database.</td>
</tr>
<tr>
<td>ProcessGetVar</td>
<td>Processes snmp Get request pdu.</td>
</tr>
<tr>
<td>ProcessSnmpv3MsgData</td>
<td>This routine processes the snmpv3 request and parallelly creates the response pdu.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SNMPGetExactIndex</td>
<td>To search for exact index node in case of a Sequence variable.</td>
</tr>
<tr>
<td>SNMPGetTrapTime</td>
<td>Returns trap resolve get time.</td>
</tr>
<tr>
<td>SNMPldRecrdValidation</td>
<td>Used to Restrict the access dynamic and non dynamic OID string for A particular SNMP Version.</td>
</tr>
<tr>
<td>SNMPIsValidSetLen</td>
<td>Validates the set variable data length to data type.</td>
</tr>
<tr>
<td>Snmpv3AESDecryptRxedScopedPdu</td>
<td>Incoming SNMPv3 scoped PDU decryption using AES decryption protocol.</td>
</tr>
<tr>
<td>Snmpv3AESEncryptResponseScopedPdu</td>
<td>OutGoing SNMPv3 scoped PDU Encryption using AES encryption protocol.</td>
</tr>
<tr>
<td>Snmpv3AuthenticateRxedPduForData Integrity</td>
<td>Authenticate an incoming SNMPV3 USM PDU using MD5 or SHA</td>
</tr>
<tr>
<td>Snmpv3AuthenticateTxPduForData Integrity</td>
<td>Authenticate to an outgoing SNMPV3 USM PDU using MD5 or SHA</td>
</tr>
<tr>
<td>Snmpv3AuthKeyZeroing2HmacBufLen64</td>
<td>Pad zero to the authentication key localized buffer.</td>
</tr>
<tr>
<td>Snmpv3BufferPut</td>
<td>Copies BYTE data to dynamically allocated memory buffer.</td>
</tr>
<tr>
<td>Snmpv3CmprTrapSecNameAndSecLvlWithUSMdB</td>
<td>Routine to find the index of the user name in the user data base table.</td>
</tr>
<tr>
<td>Snmpv3ComputeHMACIpadOpadForAuthLoclzedKey</td>
<td>Compute HMAC inner and outer pad for authorization localized key.</td>
</tr>
<tr>
<td>Snmpv3ComputeHmacMD5Digest</td>
<td>Compute HMAC - MD5 authentication code</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Snmpv3ComputeHmacShaDigest</td>
<td>Compute HMAC - SHA authentication code</td>
</tr>
<tr>
<td>Snmpv3ComputeMd5HmacCode</td>
<td>Compute HMAC - MD5 authentication code</td>
</tr>
<tr>
<td>Snmpv3ComputeShaHmacCode</td>
<td>Compute HMAC - SHA authentication code</td>
</tr>
<tr>
<td>Snmpv3FormulateEngineID</td>
<td>Formulates the <code>snmpEngineID</code> for the SNMPV3 engine.</td>
</tr>
<tr>
<td>Snmpv3FreeDynAllocMem</td>
<td>Allocated dynamic memory freeing is done by this routine.</td>
</tr>
<tr>
<td>Snmpv3GetAuthEngineTime</td>
<td>Updates the snmp engine time variable <code>snmpEngineTime</code> for the SNMPV3 engine.</td>
</tr>
<tr>
<td>Snmpv3GetBufferData</td>
<td>Reads BYTE data from dynamically allocated memory buffer.</td>
</tr>
<tr>
<td>Snmpv3GetSecurityLevel</td>
<td>Get Security level from authentication and Privacy type.</td>
</tr>
<tr>
<td>Snmpv3GetTrapSecurityLevel</td>
<td>Routine to find the report, auth and privacy flags settings in the TRAP.</td>
</tr>
<tr>
<td>Snmpv3Init</td>
<td>SNMPv3 initialization.</td>
</tr>
<tr>
<td>Snmpv3InitializeUserDataBase</td>
<td>Initialize default SNMPv3 global user database.</td>
</tr>
<tr>
<td>Snmpv3IsValidAuthStructure</td>
<td>Decode variable length structure.</td>
</tr>
<tr>
<td>Snmpv3IsValidInt</td>
<td>Verifies variable datatype as INT and retrieves its value.</td>
</tr>
<tr>
<td>Snmpv3MsgProcessingModelProcessPDU</td>
<td>This routine collects or populates the message processing model infomation from the received SNMPv3</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Snmpv3Notify</td>
<td>Creates and Sends SNMPv3 TRAP pdu.</td>
</tr>
<tr>
<td>Snmpv3Pswd2LocalizedAuthKeyMD5Hashing</td>
<td>Convert MD5 Auth password to the localized Key using SNMPEngineID.</td>
</tr>
<tr>
<td>Snmpv3Pswd2LocalizedAuthKeySHAHashing</td>
<td>Convert SHA Auth password to the localized Key using SNMPEngineID.</td>
</tr>
<tr>
<td>Snmpv3ScopedPduProcessing</td>
<td>This routine collects the scoped pdu header information from the received SNMPv3 request PDU or populates to the response PDU respectively.</td>
</tr>
<tr>
<td>Snmpv3SetErrorStatus</td>
<td>Set snmpv3 error status in the response pdu.</td>
</tr>
<tr>
<td>Snmpv3TrapScopedpdu</td>
<td>TRAP PDU scoped pdu header construction.</td>
</tr>
<tr>
<td>Snmpv3UserSecurityModelProcessPDU</td>
<td>This routine collects or populates the security model parameters information from the received SNMPv3 request PDU or to the response PDU respectively.</td>
</tr>
<tr>
<td>Snmpv3UsmAesEncryptDecryptInitVector</td>
<td>AES Encryption and decryption init vector. (RFC 3826)</td>
</tr>
<tr>
<td>Snmpv3UsmOutMsgAuthenticationParam</td>
<td>Both MD5 and SHA1 is used for the outgoing message authentication.</td>
</tr>
<tr>
<td>Snmpv3USMOutMsgPrivParam</td>
<td>SNMP USM out message uses Privacy protocol (RFC 3826)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Snmpv3UsmSnmpEngnAuthPrivPswdLocalization</td>
<td>Convert Auth and Priv password to the localized Key using SNMPEngineID.</td>
</tr>
<tr>
<td>Snmpv3ValidateEngineId</td>
<td>Validate engine ID.</td>
</tr>
<tr>
<td>Snmpv3ValidateSecNameAndSecLv</td>
<td>Validate security name with Security level.</td>
</tr>
<tr>
<td>Snmpv3ValidateSecurityName</td>
<td>Validate SNMPV3 user name or security name.</td>
</tr>
</tbody>
</table>

Stack API > SNMP

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc.  All rights reserved.

[Contents] [Index] [Home]
## SNMP Macros

### Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH LOCALIZED PASSWORD KEY LEN</td>
<td>SNMPv3 Authentication length size</td>
</tr>
<tr>
<td>INVALID_INDEX</td>
<td>This is macro INVALID_INDEX.</td>
</tr>
<tr>
<td>IS_SNMPV3_AUTH_STRUCTURE</td>
<td>This is macro IS_SNMPV3_AUTH_STRUCTURE.</td>
</tr>
</tbody>
</table>
| MSG_AUTHORITATIVE_HEADER_LEN                     | Length of SNMPv3 authoritative msg header length = Header length (2 bytes) +
                                                       engine ID (snmpEngnIDLength engine boot (4 bytes) +
                                                       engine time (4 bytes) +
                                                       security name (securityPrimitive +
                                                       authentication parameters (snmpOutMsgAuthParamLen +
                                                       parameters (snmpOutMsgAuthParamLen |
| MSGGLOBAL_HEADER_LEN                             | Length of the SNMPv3 msg header (2 bytes) +
                                                       MSGID: length of MsgID value +
                                                       msgMAXSIZE: value +
                                                       msg flag: value +
                                                       security model type: value +
                                                       security model length of value +
                                                       #define |
<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIV_LOCALIZED_PASSWORD_KEY_LEN</td>
<td>SNMPv3 Privacy Key Length for Validation</td>
</tr>
<tr>
<td>REPORT_RESPONSE</td>
<td>This is macro REPORT_RESPONSE</td>
</tr>
<tr>
<td>SNMP_ENGINE_MAX_MSG_SIZE</td>
<td>SNMP ENGINE MAX MSG SIZE is determined as the minimum of the max msg size values supported among all transports available to and supported by the engine.</td>
</tr>
<tr>
<td>SNMP_MAX_MSG_SIZE</td>
<td>SNMP MIN and MAX message size is 484 bytes in size</td>
</tr>
<tr>
<td>SNMP_MAX_OID_LEN_MEM_USE</td>
<td>This macro will be used to avoid SNMP OID memory buffer corruption</td>
</tr>
<tr>
<td>SNMP_trap_community_max_len_mem_use</td>
<td>This macro will be used to avoid SNMP OID memory buffer corruption</td>
</tr>
<tr>
<td>SNMP_MAX_OID_LEN_MEM_USE</td>
<td>SNMP_MAX_OID_LEN_MEM_USE</td>
</tr>
<tr>
<td>SNMP_trap_community_max_len_mem_use</td>
<td>SNMP_trap_community_max_len_mem_use</td>
</tr>
<tr>
<td>SNMP_V3</td>
<td>This is macro SNMP_V3</td>
</tr>
<tr>
<td>SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE</td>
<td>SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE</td>
</tr>
<tr>
<td>SNMPV3_H</td>
<td>This is macro SNMPV3_H</td>
</tr>
<tr>
<td>SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE</td>
<td>SNMPV3 PRIV LOCALIZED PASSWORD KEY LEN MEM USE</td>
</tr>
<tr>
<td>SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE</td>
<td>User security name length for memory validation</td>
</tr>
<tr>
<td>SNMPV3_USM_MAX_USER</td>
<td>User Security Model should have at least 1 user. Default is 3. User should change as per the requirement.</td>
</tr>
<tr>
<td>SNMPV3_MSG_AUTHENTICATION_FAIL</td>
<td>This is macro SNMPV3_MSG_AUTHENTICATION_FAIL</td>
</tr>
<tr>
<td>SNMPV3_MSG_AUTHENTICATION_SUCCESS</td>
<td>This is macro SNMPV3_MSG_AUTHENTICATION_SUCCESS</td>
</tr>
<tr>
<td>USER_SECURITY_NAME_LEN</td>
<td>SNMPv3 User Security Name Length</td>
</tr>
</tbody>
</table>
### SNMP Structs, Records, Enums

#### Structs, Records, Enums

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **AccessCtrlSubSysIsAccessAllowed**       | Applications are the typical clients of the service(s) of the Access Control Subsystem. The following primitive is provided by the Access Control Subsystem to check if access is allowed:  
  `statusInformation` = -- success or errorIndication                                                                                       |
| **dispatcherProcessPdu**                  | Process Incoming Request or Notification PDU  
  Dispatcher provides the following primitive to pass an incoming snmp pdu to an application.                                               |
| **dispatcherStatusInfo**                  | Generate Outgoing Request or Notification  
  `statusInformation` = -- sendPduHandle if success -- errorIndication if failure                                                                 |
| **dispatcherReturnResponsePdu**           | Generate Outgoing Response  
  The PDU Dispatcher provides the following primitive for an application to return an SNMP Response PDU to the PDU Dispatcher:  
  result = SUCCESS or FAILURE                                                                                                                  |
| **MsgProcModPrepareDataElements**        | Prepare Data Elements from an Incoming SNMP Message  
  The Message Processing Subsystem provides this service primitive for preparing the abstract data elements from an incoming SNMP message:  
  result = -- SUCCESS or errorIndication                                                                                                       |
| **MsgProcModPrepareOutgoingMessage**     | Prepare Outgoing SNMP Request or Notification Message  
  The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Request or Notification Message                  |
<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MsgProcModPrepareResponseMessage</td>
<td>Prepare an Outgoing SNMP Response Message</td>
</tr>
<tr>
<td></td>
<td>The Message Processing Subsystem provides this service primitive for</td>
</tr>
<tr>
<td></td>
<td>preparing an outgoing SNMP Response Message: result = -- SUCCESS or</td>
</tr>
<tr>
<td></td>
<td>FAILURE</td>
</tr>
<tr>
<td>processResponsePdu</td>
<td>Process Incoming Response PDU</td>
</tr>
<tr>
<td></td>
<td>The PDU Dispatcher provides the following primitive to pass an incoming</td>
</tr>
<tr>
<td></td>
<td>SNMP Response PDU to an application:</td>
</tr>
<tr>
<td>registerContextEngineID</td>
<td>success or errorIndication</td>
</tr>
<tr>
<td>SecuritySysGenerateRequestMsg</td>
<td>This is record SecuritySysGenerateRequestMsg.</td>
</tr>
<tr>
<td>SecuritySysGenerateResponseMsg</td>
<td>Generate a Response Message</td>
</tr>
<tr>
<td></td>
<td>The Security Subsystem provides the following primitive to generate a</td>
</tr>
<tr>
<td></td>
<td>Response message:</td>
</tr>
<tr>
<td>StateRelease</td>
<td>Release State Reference Information</td>
</tr>
<tr>
<td></td>
<td>All Subsystems which pass stateReference information also provide a</td>
</tr>
<tr>
<td></td>
<td>primitive to release the memory that holds the referenced state information</td>
</tr>
<tr>
<td>unregisterContextEngineID</td>
<td>This is record unregisterContextEngineID.</td>
</tr>
</tbody>
</table>
# SNMP Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INOUT_SNMP_PDU</td>
<td>This is type INOUT_SNMP_PDU.</td>
</tr>
<tr>
<td>REPORT_FLAG_AND_SECURITY_LEVEL_FLAGS</td>
<td>This is type REPORT_FLAG_AND_SECURITY_LEVEL_FLAGS.</td>
</tr>
<tr>
<td>SecuritySysProcessIncomingMsg</td>
<td>This is type SecuritySysProcessIncomingMsg.</td>
</tr>
<tr>
<td>SNMP_ENGNID_OCTET_IDENTIFIER_VAL</td>
<td>The fifth octet indicates how the rest (6th and following octets) are formatted. Refer to RFC3411 section5 Page# 41</td>
</tr>
<tr>
<td>SNMPNONMIBRECDINFO</td>
<td>This is type SNMPNONMIBRECDINFO.</td>
</tr>
<tr>
<td>SNMPV3_HMAC_HASH_TYPE</td>
<td>Type of hash being calculated.</td>
</tr>
<tr>
<td>SNMPV3_MSG_AUTH_SEC_PARAM_RESULT</td>
<td>This is type SNMPV3_MSG_AUTH_SEC_PARAM_RESULT.</td>
</tr>
<tr>
<td>SNMPV3_MSG_PRIV_SEC_PARAM_RESULT</td>
<td>This is type SNMPV3_MSG_PRIV_SEC_PARAM_RESULT.</td>
</tr>
<tr>
<td>SNMPV3_PRIV_PROT_TYPE</td>
<td>This is type SNMPV3_PRIV_PROT_TYPE.</td>
</tr>
<tr>
<td>SNMPV3_REQUEST_WHOLEMSG</td>
<td>This is type SNMPV3_REQUEST_WHOLEMSG.</td>
</tr>
<tr>
<td>SNMPV3_RESPONSE_WHOLEMSG</td>
<td>This is type SNMPV3_RESPONSE_WHOLEMSG.</td>
</tr>
<tr>
<td>snmpV3EngnUserDataBase</td>
<td>This is type snmpV3EngnUserDataBase.</td>
</tr>
<tr>
<td>SNMPV3MSGDATA</td>
<td>SNMPv3</td>
</tr>
<tr>
<td>snmpV3TrapConfigDataBase</td>
<td>snmpv3 target configuration</td>
</tr>
<tr>
<td>statusInformation</td>
<td>success or errorIndication</td>
</tr>
<tr>
<td>STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL</td>
<td>This is type STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL.</td>
</tr>
<tr>
<td>STD_BASED_SNMP_SECURITY_MODEL</td>
<td>Snmp Message Processing Model.</td>
</tr>
</tbody>
</table>
STD_BASED_SNMPV3_SECURITY_LEVEL

USM_SECURITY_LEVEL

This is type STD_BASED_SNMPV3_SECURITY_LEVEL.

This is type USM_SECURITY_LEVEL.

Stack API > SNMP
# SNMP Variables

## Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authKey_iPad</td>
<td>This is variable authKey_iPad.</td>
</tr>
<tr>
<td>authKey_oPad</td>
<td>This is variable authKey_oPad.</td>
</tr>
<tr>
<td>authoritativeSnmpEngineBoots</td>
<td>The number of times that the authoritative SNMP engine has (re-)initialized itself since its snmpEngineID was last configured.</td>
</tr>
<tr>
<td>authoritativeSnmpEngineTime</td>
<td>The number of seconds since the value of the authoritativeSnmpEngineBoots object last changed</td>
</tr>
<tr>
<td>cipher_text</td>
<td>This is variable cipher_text.</td>
</tr>
<tr>
<td>deciphered_text</td>
<td>This is variable deciphered_text.</td>
</tr>
<tr>
<td>getZeroInstance</td>
<td>This variable is used for gext next request for zero instance</td>
</tr>
<tr>
<td>gSnmpV3InPduWholeMsgBuf</td>
<td>Dynamic memory stub and PDU details for Incoming stored PDU</td>
</tr>
<tr>
<td>gSnmpV3OUTPduWholeMsgBuf</td>
<td>Dynamic memory stub details and constructed outgoing stored PDU details</td>
</tr>
<tr>
<td>gSNMPv3PduHeaderBuf</td>
<td>Response PDU construction offset details</td>
</tr>
<tr>
<td>gSNMPv3ScopedPduDataPos</td>
<td>Offset to read scoped PDU data bytes for processing from dynamic memory stub</td>
</tr>
<tr>
<td>gSNMPv3ScopedPduRequestBuf</td>
<td>Stored request scoped pdu details</td>
</tr>
<tr>
<td>gSNMPv3ScopedPduResponseBuf</td>
<td>Processed response scoped pdu details</td>
</tr>
<tr>
<td>gSnmpv3TrapConfigData</td>
<td>SNPv3 global configuration database to be used for trap notification</td>
</tr>
<tr>
<td></td>
<td>TRAP message PDU header construction</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>gSNMPv3TrapMsgHeaderBuf</td>
<td>Offset details</td>
</tr>
<tr>
<td>gSnmpV3TrapOUTPduWholeMsgBuf</td>
<td>Dynamic memory stub details and constructed trap PDU details</td>
</tr>
<tr>
<td>gSNMPv3TrapScopedPduResponseBuf</td>
<td>TRAP scoped PDU construction offset details</td>
</tr>
<tr>
<td>gSNMPV3TrapSecurityLevel</td>
<td>This is variable gSNMPV3TrapSecurityLevel.</td>
</tr>
<tr>
<td>gSnmpv3UserDBIndex</td>
<td>Index to the particular reference configured in User security model data base snmpV3UserDataBase.</td>
</tr>
<tr>
<td>gUsmStatsEngineID</td>
<td>Global variable to find out how many times SNMPv3 engine id has been validated</td>
</tr>
<tr>
<td>hmacAuthKeyBuf</td>
<td>This is variable hmacAuthKeyBuf.</td>
</tr>
<tr>
<td>HmacMd5Digest</td>
<td>This is variable HmacMd5Digest.</td>
</tr>
<tr>
<td>HmacSHADigest</td>
<td>This is variable HmacSHADigest.</td>
</tr>
<tr>
<td>incomingPdu</td>
<td>Incoming PDU details</td>
</tr>
<tr>
<td>incomingSnmpPDUmsgID</td>
<td>Retrieved Incoming Msg ID value from PDU</td>
</tr>
<tr>
<td>ivEncrptKeyOut</td>
<td>This is variable ivEncrptKeyOut.</td>
</tr>
<tr>
<td>md5LocalizedAuthKey</td>
<td>This is variable md5LocalizedAuthKey.</td>
</tr>
<tr>
<td>msgSecrtyParamLenOffset</td>
<td>This is variable msgSecrtyParamLenOffset.</td>
</tr>
<tr>
<td>securityPrimitivesOfIncomingPdu</td>
<td>Incoming PDU Security primitive details.</td>
</tr>
<tr>
<td>session_key</td>
<td>This is variable session_key.</td>
</tr>
<tr>
<td>sha1LocalizedAuthKey</td>
<td>This is variable sha1LocalizedAuthKey.</td>
</tr>
<tr>
<td>snmpEngineBoots</td>
<td>The number of times that the SNMP engine has (re-)initialized itself since snmpEngineID was last configured.</td>
</tr>
<tr>
<td>snmpEngineID</td>
<td>Reserving 32 bytes for the snmpEngineID as the octet string length can vary form 5 to</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>snmpEngineMaxMessageSize</strong></td>
<td>The maximum message size the SNMP engine can handle.</td>
</tr>
<tr>
<td><strong>snmpEngineMsgProcessModel</strong></td>
<td>Type of Message processing model used. Value Maximum range (2^31-1), RFC3411</td>
</tr>
<tr>
<td><strong>snmpEngineSecurityModel</strong></td>
<td>Type of security model used. Value Maximum range (2^31-1), RFC3411</td>
</tr>
<tr>
<td><strong>snmpEngineTime</strong></td>
<td>The number of seconds since the value of the <code>snmpEngineBoots</code> object last changed</td>
</tr>
<tr>
<td><strong>snmpEngineTimeOffset</strong></td>
<td>Stores the time value in seconds since SNMP Engine reset</td>
</tr>
<tr>
<td><strong>snmpEngnIDLength</strong></td>
<td>Engine ID length of the SNMP Engine</td>
</tr>
<tr>
<td><strong>snmpInMsgAuthParamLen</strong></td>
<td>Incoming SNMPv3 msg authentication parameters string is 12 bytes long.</td>
</tr>
<tr>
<td><strong>snmpInMsgAuthParamStrng</strong></td>
<td>Reserving 12 bytes for the incoming SNMPv3 msg authentication parameters.</td>
</tr>
<tr>
<td><strong>snmpInMsgPrivParamLen</strong></td>
<td>Incoming SNMPv3 msg privacy parameters string is 8 bytes long.</td>
</tr>
<tr>
<td><strong>snmpInMsgPrvParamStrng</strong></td>
<td>Reserving 8 bytes for the incoming SNMPv3 msg privacy parameters.</td>
</tr>
<tr>
<td><strong>snmpMsgBufSeekPos</strong></td>
<td>Offset to read PDU data bytes for processing from dynamic memory stub</td>
</tr>
<tr>
<td><strong>snmpOutMsgAuthParamLen</strong></td>
<td>Outgoing SNMPv3 msg authentication parameters string is 12 bytes long.</td>
</tr>
<tr>
<td><strong>snmpOutMsgAuthParamStrng</strong></td>
<td>Reserving 12 bytes for the outgoing SNMPv3 msg authentication parameters.</td>
</tr>
<tr>
<td><strong>snmpOutMsgPrivParamLen</strong></td>
<td>Outgoing SNMPv3 msg privacy parameters string is 8 bytes long.</td>
</tr>
<tr>
<td><strong>snmpOutMsgPrvParamStrng</strong></td>
<td>Reserving 8 bytes for the outgoing SNMPv3 msg privacy parameters.</td>
</tr>
<tr>
<td><strong>snmpResponseSecurityFlag</strong></td>
<td>Type of Security for outgoing message in response to the incoming message.</td>
</tr>
<tr>
<td><strong>snmpSecurityLevel</strong></td>
<td>Type of security. noAuthNoPriv(0), AuthNoPriv(1), AuthPriv(3)</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>snmpTrapTimer</strong></td>
<td>This is variable snmpTrapTimer.</td>
</tr>
<tr>
<td><strong>snmpV3AesDecryptInitVector</strong></td>
<td>128 Bit</td>
</tr>
<tr>
<td><strong>snmpV3AesEncryptInitVector</strong></td>
<td>128 Bit</td>
</tr>
<tr>
<td><strong>snmpV3UserDataBase</strong></td>
<td>This is variable snmpV3UserDataBase.</td>
</tr>
</tbody>
</table>

Stack API > SNMP

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
## Files

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLClientSize.h</td>
<td>This is file SSLClientSize.h.</td>
</tr>
</tbody>
</table>

Stack API > SSL
## TCP Functions

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFGetTCBSIZE</td>
<td>Returns number of bytes available in TCP Control Block (TCB) so higher-layer code can determine if the number of bytes available can support the structures designated to be stored in the TCB.</td>
</tr>
</tbody>
</table>
**strTitle Variable**

```c
ROM BYTE strTitle[] = "\x1b[2J\x1b[31m\x1b[1m" "Microchip Telnet Server"
```

**Description**

Demo title string

[Stack API > Telnet > Internal Members > strTitle Variable](#)
## Tick Module Functions

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>attribute</strong></td>
<td>6-byte value to store Ticks. Allows for use over longer periods of time.</td>
</tr>
</tbody>
</table>

Stack API > Tick
## UDP Types

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP_STATE</td>
<td>UDP States</td>
</tr>
</tbody>
</table>

Stack API > UDP
## Wi-Fi Power Save Functions

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GetAppPowerSaveMode</code></td>
<td>Returns state of power save mode</td>
</tr>
<tr>
<td><code>SetAppPowerSaveMode</code></td>
<td>Enable or disable power save mode</td>
</tr>
</tbody>
</table>

*Wi-Fi API > Wi-Fi Power Save*
### Wi-Fi Power Save Types

#### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tWFPsPwrMode</td>
<td>Enumeration of valid values for WFSetPowerSaveMode()</td>
</tr>
<tr>
<td>tWFPwrModeReg</td>
<td>Power Save Mode Request Structure</td>
</tr>
</tbody>
</table>

[Wi-Fi API > Wi-Fi Power Save](URL)
### Wi-Fi Power Save Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_AppPowerSaveModeEnabled</td>
<td>Enable or Disable Power Save Mode</td>
</tr>
<tr>
<td>g_powerSaveState</td>
<td>Power save states</td>
</tr>
<tr>
<td>g_psPollActive</td>
<td>Status of PS-Poll</td>
</tr>
<tr>
<td>g_sleepNeeded</td>
<td>TRUE if need to put device back into PS-Poll sleep mode. else FALSE</td>
</tr>
</tbody>
</table>
MAX_TELNET_CONNECTIONS Macro

C

```c
#define MAX_TELNET_CONNECTIONS (1u)
```

Description

Number of simultaneously allowed Telnet 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.
**strAuthenticated Variable**

ROM BYTE `strAuthenticated[] = "\r\nLogged in successfully\r\n\" "\r

**Description**

Successful authentication message

[Stack API > Telnet > Internal Members > strAuthenticated Variable](#)
strDisplay Variable

C

ROM BYTE strDisplay[] = "\n\nSNTP Time: (disabled)" "\nAnalog: 1023"

Buttons: 3 2 1 0

LEDs: 7 6 5 4 3 2 1 0

Description

Demo output string

Stack API > Telnet > Internal Members > strDisplay Variable

Microchip TCP/IP Stack 5.42.08 - June 15, 2013
Copyright © 2012 Microchip Technology, Inc. All rights reserved.
strGoodBye Variable

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM BYTE strGoodBye[] = &quot;\r\n\nGoodbye!\r\n&quot;;</td>
</tr>
</tbody>
</table>

**Description**

Demo disconnection message
strPassword Variable

C

ROM BYTE strPassword[] = "Password: \xff\xfd\x2d";

Description

DO Suppress Local Echo (stop telnet client from printing typed characters) Access denied message
strSpaces Variable

C
ROM BYTE strSpaces[] = "          ";

Description

String with extra spaces, for Demo
The following functions and variables are designated as internal to the Telnet module.

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strSpaces</td>
<td>String with extra spaces, for Demo</td>
</tr>
<tr>
<td>strAuthenticated</td>
<td>Successful authentication message</td>
</tr>
<tr>
<td>strDisplay</td>
<td>Demo output string</td>
</tr>
<tr>
<td>strGoodBye</td>
<td>Demo disconnection message</td>
</tr>
<tr>
<td>strPassword</td>
<td>DO Suppress Local Echo (stop telnet client from printing typed characters) Access denied message</td>
</tr>
<tr>
<td>strTitle</td>
<td>Demo title string</td>
</tr>
</tbody>
</table>
Telnet Public Members

The following functions and variables are available to the stack application.

Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_TELNET_CONNECTIONS</td>
<td>Number of simultaneously allowed Telnet 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.</td>
</tr>
<tr>
<td>TELNET_PASSWORD</td>
<td>Default Telnet password</td>
</tr>
<tr>
<td>TELNET_PORT</td>
<td>Default local listening port for the Telnet server. Port 23 is the protocol default.</td>
</tr>
<tr>
<td>TELNETS_PORT</td>
<td>Default local listening port for the Telnet server when SSL secured. Port 992 is the telnets protocol default.</td>
</tr>
<tr>
<td>TELNET_USERNAME</td>
<td>Default username and password required to login to the Telnet server.</td>
</tr>
</tbody>
</table>

Module

Telnet

Stack API > Telnet > Public Members
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.

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TelnetTask</td>
<td>Performs Telnet Server related tasks. Contains the Telnet state machine and state tracking variables.</td>
</tr>
</tbody>
</table>

### Module

**Telnet**

Stack API > Telnet > Stack Members
TELNET_PASSWORD Macro

C

#define TELNET_PASSWORD "microchip"

Description

Default Telnet password
TELNET_PORT Macro

C

#define TELNET_PORT 23

Description

Default local listening port for the Telnet server. Port 23 is the protocol default.
TELNET_USERNAME Macro

```
C
#define TELNET_USERNAME "admin"
```

**Description**

Default username and password required to login to the Telnet server.
**TELNETS_PORT Macro**

```c
#define TELNETS_PORT 992
```

**Description**

Default local listening port for the [Telnet](https://en.wikipedia.org/wiki/Telnet) server when SSL secured. Port 992 is the telnets protocol default.
TelnetTask Function

C

void TelnetTask();

Description

Performs Telnet Server related tasks. Contains the Telnet state machine and state tracking variables.

Preconditions

Stack is initialized()

Returns

None

Side Effects

None

Remarks

None